GaitNet: Greedy, Dynamic Quadruped Gait Generation

Owen Sullivan

Supervisors:
Dr. Mohammad Mahdi Agheli Hajiabadi

Worcester Polytechnic Institute

October 17, 2025

Contents

List of Figures

List of Tables

List of Abbreviations

TODO: Make sure these are all used in the text, with nothing missing

TODO: Order alphabetically

MPC	**M**odel **P**redictive **C**ontrol
RL	**R**einforcement **L**earning
ML	**M**achine **L**earning
MLP	**M**ulti-**L**ayer **P**erceptron
DNN	**D**eep **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
DQN	**D**eep **Q** **N**etwork
PPO	**P**roximal **P**olicy **O**ptimization

List of Symbols

$\mathbf{f}_c$	Footstep candidate Vector of *leg index* (or -1 for no leg), *dx*, *dy*, and *cost*. With *dx* and *dy* being offsets from the nominal foot position in the base frame.	$\begin{bmatrix}\text{-} & \text{m} & \text{m} & \text{-}\end{bmatrix}^T$
$\mathbf{f}_a$	Footstep action Vector of *leg index* (or -1 for no leg), *dx*, *dy*, and *swing duration*. With *dx* and *dy* being offsets from the nominal foot position in the base frame.	$\begin{bmatrix}\text{-} & \text{m} & \text{m} & \text{s}\end{bmatrix}^T$
$\mathbf{r}_w$	COM position in world frame	m
$\mathbf{q}_{rw}$	Root orientation (quaternion) in world frame	-
$\mathbf{v}_b$	Root linear velocity in base frame	$\mathrm{m\,s^{-1}}$
ω_b	Root angular velocity in base frame	$\mathrm{rad\,s^{-1}}$
$\mathbf{p}_{ib}$	End-effector i position in base frame	m
$\mathbf{g}$	Gravity vector in base frame	$\mathrm{m\,s^{-2}}$
$\mathbf{c}_i$	End-effector i contact state	-
$\mathbf{q}$	Joint positions	-
$\dot{\mathbf{q}}$	Joint velocities	-
$\ddot{\mathbf{q}}$	Joint accelerations	-
τ	Joint torques	$\mathrm{N\,m}$
$\mathbf{u}$	Control Input Vector of v_x, v_y, and ω	$\begin{bmatrix}\mathrm{m\,s^{-1}} & \mathrm{m\,s^{-1}} & \mathrm{rad\,s^{-1}}\end{bmatrix}^T$

Note: A symbol defined with a subscript i, but displayed without it indicates the concatenation of all i elements, e.g. $\mathbf{c} = \begin{bmatrix}\mathbf{c}_1 & \mathbf{c}_2 & \mathbf{c}_3 & \mathbf{c}_4\end{bmatrix}^T$

1 Introduction

Quadruped locomotion in unstructured environments remains a complex challenge, particularly when traditional gait cycles are insufficient. While many systems rely on periodic gait patterns or centralized planners, recent advances in learning-based methods have opened the door to more adaptive, non-gaited approaches. This proposal investigates a novel method for generating non-gaited, dynamic footstep plans using a CNN-based greedy planner, aiming for a balance between computational efficiency and dynamic behaviors.

1.1 Problem Statement

1.2 Motivation

1.3 Research Question

> Can a machine learning planner generate dynamic, acyclic gaits for quadruped robots in challenging environments?

1.4 Hypothesis

1.5 Contributions

1.6 Thesis Structure

2 Background

Recent advances in legged locomotion have largely focused on either implicit gait-based policies or perception-driven foothold selection modules. However, these approaches often trade off between agility, expressiveness, and computational cost. In contrast to gaited methods that impose rhythmic structure, non-gaited planning allows for more versatile, terrain-adaptive behaviors—but typically at the expense of increased planning complexity. Bridging this gap, our proposed approach draws on greedy methods for their computational efficiency and on convolutional neural networks (CNNs) for terrain-aware generalization, aiming to achieve dynamic, non-gaited footstep planning in real time. This section reviews the foundations upon which our approach builds: learning-based locomotion strategies, footstep planners, greedy selection mechanisms, and foothold classification techniques. Each informs a different aspect of our planner's design and situates our method within the broader landscape of quadruped control.

2.1 Locomotion Planning Paradigms

2.2 Learning-Based Locomotion Strategies

Recent work has shown the potential of deep learning to generate agile, robust locomotion policies, often without explicit footstep planning. Shi et al. [2] use a neural network to modulate trajectory generator parameters in real time for energy-efficient walking. Xie et al. [3] train reinforcement learning policies on centroidal dynamics models to output desired body accelerations, assuming a fixed foot-placement heuristic and gait pattern. Duan et al. [4] learn step-to-step transitions using proprioception, generating joint targets and varying step frequency for terrain-adaptive behaviors. Siekmann et al. [5] focus on blind locomotion by training an LSTM policy to handle randomized stairs using only proprioceptive feedback. Lee et al. [6] employ a temporal convolutional network to infer terrain structure from proprioceptive history, using an automated curriculum to adapt to progressively harder environments. While these approaches enable robust locomotion across diverse terrains, they generally rely on fixed or implicit gait patterns and lack explicit control over individual footstep selection, making them less suitable for non-gaited or footstep-specific planning.

2.3 Learning-Based Footstep Planners

Learning-based footstep planners combine perception and control through data-driven models, often using a learned component upstream of a model-predictive controller. FootstepNet [7] uses deep reinforcement learning to generate heuristic-free plans but is limited to bipeds in simple 2D environments. ContactNet [1] employs a CNN to produce non-gaited footstep sequences using greedy selection, though it is constrained to one-leg-at-a-time motion and coarse foothold discretization. DeepGait [8] separates footstep planning and motion optimization, enabling more modular design but still relying on static gaits. Omar et al. [9] use a CNN to identify safe stepping regions, which are then decomposed into convex regions and passed to an MPC for downstream planning. While efficient and fully onboard, the method relies on fixed gait sequences and is primarily focused on safe terrain classification. Villarreal et al. [10] similarly use a CNN for foothold classification, replacing expert heuristics and enabling real-time performance with significantly faster inference, but their method also depends on a fixed gait schedule. DeepLoco [11] introduces a hierarchical policy for footstep and motion planning, but its coarse terrain input and focus on gaited footstep generation

for bipeds limit dynamic adaptability. Taouil et al. [12] use Monte Carlo Tree Search (MCTS) to explore dynamic and non-gaited stepping combinations, making them first to generate dynamic, non-gaited footstep plans in real time. Their approach does suffer from a high computational cost due to its high branching factor, especially on certain types of terrain [1]. Overall, while these approaches advance learning-based planning, few simultaneously support both non-gaited and dynamically unstable footstep generation.

2.4 Greedy and Heuristic Footstep Planners

Greedy planners offer computationally efficient alternatives to exhaustive search, often prioritizing goal-directed heuristics or local stability metrics. Gao et al. [13] propose GH-QP, a greedy-heuristic hybrid that incorporates expected robot speed into a footstep planning heuristic, though it is limited to bipedal systems and lacks support for dynamic footstep plans. Zucker et al. [14] use A* with a learned terrain cost and Dubins car heuristic for footstep planning, enabling effective search in SE(2) but restricted to static, gated locomotion. Kalakrishnan et al. [**?**] combine greedy terrain-cost search with a coarse-to-fine pose optimization pipeline, but rely on fixed gait patterns and struggle with highly constrained environments. While these works demonstrate the potential of greedy methods, none support both dynamically unstable and non-gaited footstep generation. Notably, some prior methods already discussed—such as ContactNet [1], DeepLoco [11], and Villarreal et al. [10]—also incorporate greedy elements in their pipelines.

2.5 Foothold Classification Algorithms

Foothold classification algorithms focus on determining safe stepping regions, often using learned terrain models to support downstream planners that enforce safety constraints. Asselmeier et al. [15] generate steppability maps from visual inputs to guide a trajectory-optimization-based planner, validating their perception-to-control pipeline in simulation. Omar et al. [16] propose a two-stage classifier that first identifies steppable regions before selecting footholds, emphasizing safety over expressiveness by evaluating only candidate swing-foot locations. Similarly, Omar et al. [9] use a CNN to identify safe footholds, which are grouped into convex clusters and passed to an MPC for real-time planning, but the approach relies on fixed gait sequences. These methods provide robust terrain understanding but are typically designed as standalone perception modules, separate from the full footstep generation process, and thus are not directly applicable to dynamic or non-gaited motion planning.

2.6 Synthesis

3 Methodology

3.1 System Overview

This work presents a control framework for quadruped robots that can generate dynamic, acyclic gaits in challenging environments. The framework is hierarchal, taking in robot state and terrain data and outputting footstep commands to the robot controller. The first stage of the framework is a footstep evaluation network, similar to ContactNet from [1], which estimates footstep candidates $\mathbf{f}_c$. The second stage is a gait generation network (ContactNet), which takes in footstep candidates $\mathbf{f}_c$, ranks them, and finally sends the best footstep action $\mathbf{f}_a$ to the robot controller.

This hierarchal framework allows for strict control over the possible actions the robot can take. Between the two stages, the candidate actions are filtered to ensure that the robot can only take prescribed actions.

TODO: update Figure 3.1

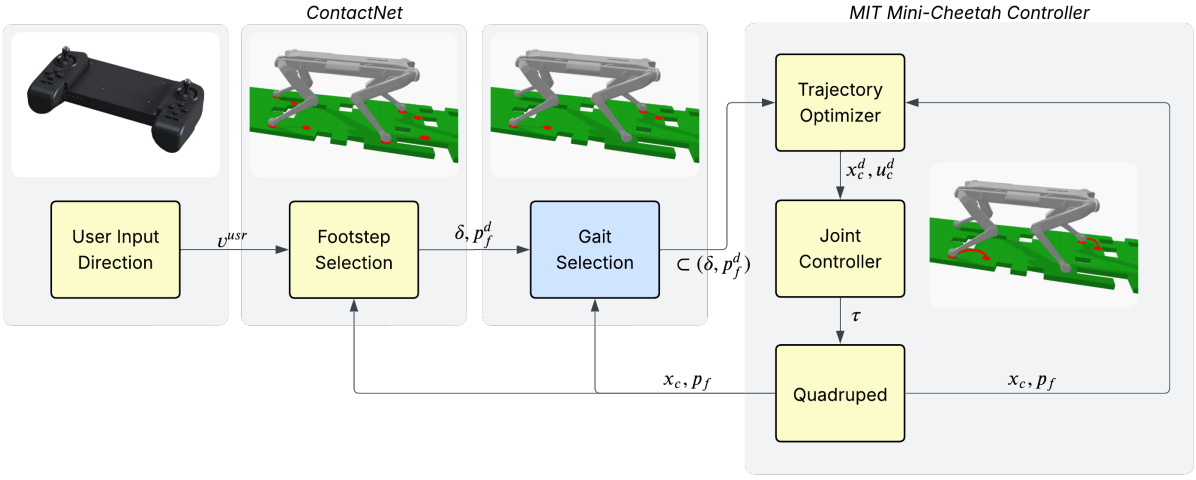


Figure 3.1: A block diagram of the proposed framework. The user defines an input direction v^{usr} which the *footstep selector* [1] uses along with, the robot state x_c, and the current foot positions p_f to generate the swing durations δ and touchdown points p_f^d for all currently grounded feet. The *gait selector* (novel) takes these desired foot movements and selects an appropriate subset $\subset (\delta, p_f^d)$ based on x_c, p_f, and the terrain data. $\subset (\delta, p_f^d)$ is then passed into the MIT Mini-Cheetah Controller as MPC constraints to perform lower level control.

3.2 Simulation Environment

The simulation environment used for this project is NVIDIA Isaac Lab [17]. This was chosen for a number of reasons. It is a modern framework with a Python interface specifically designed for machine learning, and GPU parallelism. The GPU parallelism enabled much faster simulation and data-collection at the cost of more complicated programming and reduced compatibility with older hardware. Additionally, there are a large number of example and community projects demonstrating many of the simulation features needed for this project.

Despite the simulation and learning being run on the GPU, the robot controllers are run on the CPU. Figure 3.2 shows a block diagram of the processing flow. The simulation environment is run entirely on the GPU, with each robot being simulated in parallel. The robot MPCs are run on the CPU, with each robot controller being run in parallel. The CPU and GPU communicate at each simulation step to exchange the robot state and the control actions.

TODO: add in learning step

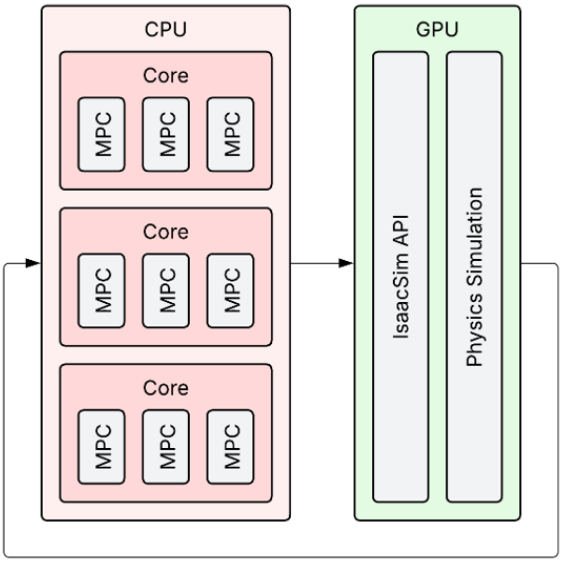


Figure 3.2: A block diagram showing the programming tasks computed on the CPU vs GPU. The full simulation is run in parallel using Nvidia Isaac Lab on the GPU, while the robot MPCs are run in parallel on the CPU.

NVIDIA Isaac Lab uses a declarative system of python dataclasses to define the simulation environment. For this work, a custom environment is used (Figure 3.3), including:

TODO: update

- Unitree Go 1—Configured to use force control for each joint.
- Terrain raycasts—Measure the height of the terrain at each possible footstep location. This emulates the lidar processing steps of a full vision pipeline.
- Custom terrain—Planar terrain with sections missing on a grid pattern. The underlying grid is 8cm square, and the void density is tuned to create a challenging but possible environment. Multiple difficulty levels of the terrain are generated for the robots to progress through as they learn. The terrain is designed to mirror that used in [1] through as they learn.

TODO: update Figure 3.3

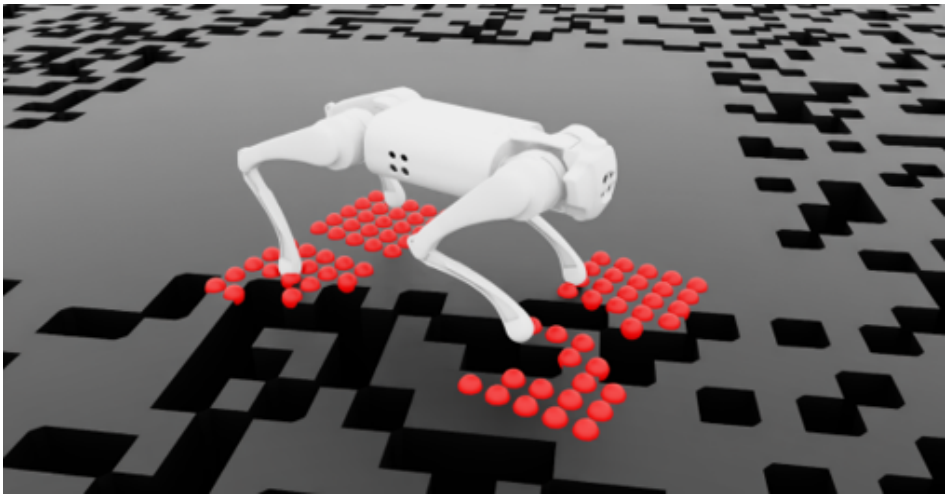


Figure 3.3: Image of a Unitree Go 1 navigating the terrain. Red spheres show the hit location of raycasts. Black regions show voids in the terrain.

3.3 Footstep Evaluation Network

The purpose of the footstep evaluation network is to provide footstep candidates to the GaitNet model. Importantly, the exact output of this network is not critical, just that it provide good candidates when sampled from. It accomplishes this by estimating the cost of potential footsteps given the robot state. The architecture and training process are similar to ContactNet [1], but with some key differences detailed below.

3.3.1 Architecture

The footstep evaluation network is tasked with generating a set of footstep candidates $\mathbf{f}_c$ based on $\mathbf{x}_c$:

$$\mathbf{x_c} = \begin{bmatrix} \mathbf{p}_{b,xy} \\ \mathbf{r}_{w,z} \\ \mathbf{v}_b \\ \omega_b \\ \mathbf{u} \end{bmatrix}$$

where $\mathbf{p}_{b,xy}$ is the x and y position all end effectors in the base frame stacked into a single vector, and $\mathbf{r}_{w,z}$ is the height of the robot's COM in the world frame. Here, the inclusion of ω_b differs from [1].

The model is trained on heuristically calculated footstep cost maps (Figure 3.7b), which estimate the cost associated with moving each foot to certain positions based on the robot state. These are represented as 4 5x5 grids of footstep candidates; one for each leg.

The footstep evaluation network architecture is shown in Figure 3.4. It consists of a feedforward neural network that first maps the input through two fully connected layers of 64 nodes each, both with ReLU activations. The resulting 64-dimensional feature vector is reshaped into an 8x8 spatial representation and processed by a convolutional layer with two output channels, a 3x3 kernel, with stride 1, and padding 1, followed by a ReLU nonlinearity. The output is flattened and passed to a final fully connected layer before being reshaped into a 5x5 grid. In order to produce the 4 footstep candidate maps, this entire network is replicated 4 times, once for each leg. The weights are not shared between legs. This design choice was made to allow the network to learn leg-specific behaviors, as dynamics may differ between legs due to asymmetries in the robot.

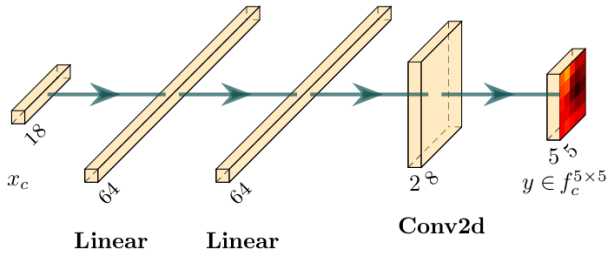


Figure 3.4: Footstep evaluation neural network architecture.

3.3.2 Training

The footstep evaluation network is trained to predict the heuristic footstep cost maps used in [1]. The training data is generated in the simulation environment described in section 3.2, During the training sessions, 100 robots are run in parallel, being split up into 4 groups of 25. Each group tests a grid of different footstep positions for one foot at a time; the front left group is shown in Figure 3.5. After every robot has finished (or failed) it's motion, an *iteration* is complete. It is important to note that all robots start each iteration from the same state.

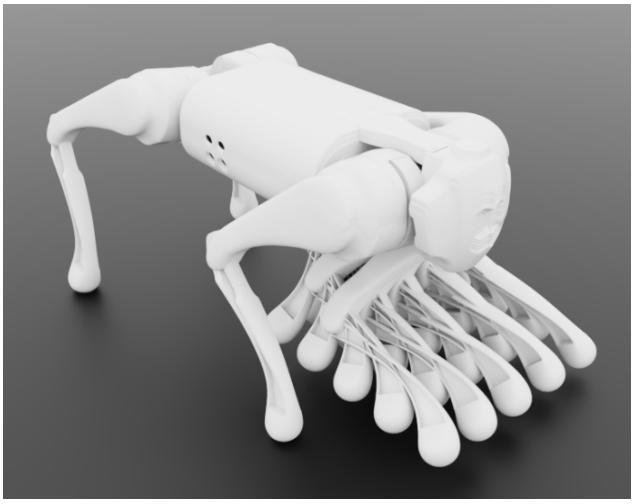


Figure 3.5: Snapshot showing 25 robots testing different footstep positions in parallel. For real data generation, 100 are run in parallel, testing 25 footstep positions for each foot at a time.

During the data collection process, we chain together many iterations, periodically changing the control inputs. After each one, the best 10 actions are placed into a tree structure as edges, with the resultant state as the leaf. The program then explores the tree by randomly selecting leaves to expand. This process is repeated until a maximum number of iterations is reached. To avoid collecting poor data, iterations which result in more than 50% of the robots falling over are discarded, as well as their 2 parents up the tree. This method ensures that a diverse set of data is collected, while still focusing on the more successful actions. Figure 3.6, shows how the foot positions are distributed in the training data. The darker spots are the discrete points on the 5x5 grid of tested footstep positions. For any given datapoint a foot is in one of these positions because it was just moved there.

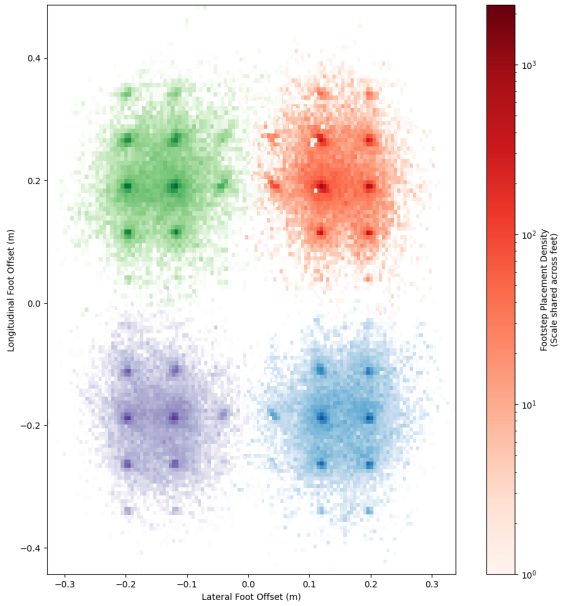
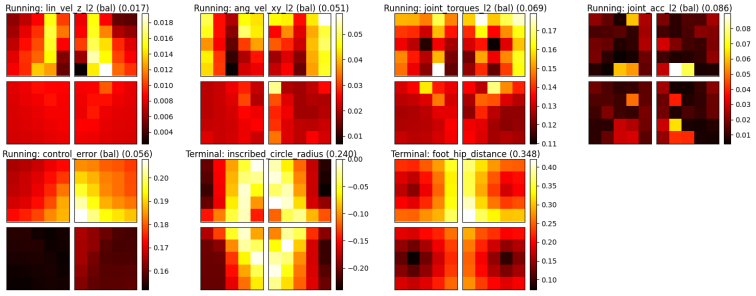


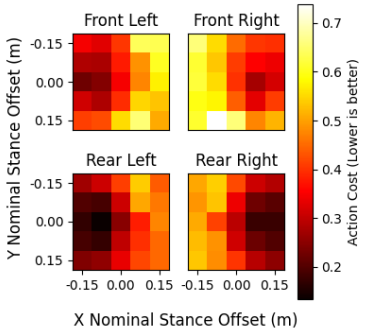
Figure 3.6: Foot placement heatmaps showing distribution of foot positions in the GaitNet training data. Note that the histograms are overlaid in some places, obscuring data underneath.

The purpose of this data collection is to generate the associated cost of moving a foot to a specific position, a set of $\mathbf{f}_c$. The cost of each footstep position is calculated heuristically from the simulations. The heuristic was designed to encourage a balance of stability and efficiency. The heuristic includes many similar elements to the one used in [1], but with some differences to better suit our system, which will eventually need to move multiple feet at once. The specific factors that make up the footstep candidate maps in the training data are shown in Figure 3.7a, while the combined cost map is shown in Figure 3.7b. The factors are justified below.

- *lin_vel_z_l2*—Penalizes high vertical velocity of the trunk.
- *ang_vel_xy_l2*—Penalizes high angular velocity of the trunk in the horizontal axes.
- *joint_torques_l2*—Penalizes high joint torques.
- *joint_acc_l2*—Penalizes high joint accelerations.
- *control_error*—Penalizes errors between the control input and actual robot motion.
- *inscribed_circle_radius*—Measures the distance from the COM to the nearest edge of the support polygon. This encourages the robot to keep its COM in a stable position.
- *foot_hip_distance*—Measures the distance from the nominal foot position. This encourages the robot to keep its feet moving along with the body.



(a) Factors influencing footstep candidate maps. *(bal)* indicates that the values for each leg were balanced to have a lower spread, mitigating factors that consistently prefer one leg over another. The last number in parenthesis indicates the total range of the data, the most important factor for the combined cost map.



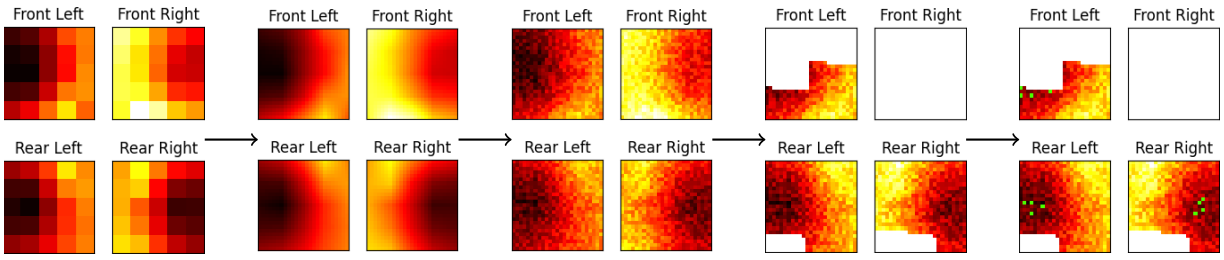
(b) Combined cost map from factors in (a) (not normalized).

Figure 3.7: Footstep candidate map composition. (a) shows the individual factors that make up the footstep candidate maps, while (b) shows the combined cost map.

As in [1], the cost maps are normalized to improve training performance. Our approach differs in how the cost maps are normalized though. We directly normalize the cost maps to the range $[0, 1]$, whereas [1] normalizes to $[0, 1]$ in such a way that only the relative ordering of costs are preserved. This difference is critical to our system to provide the upstream GaitNet model with as much information as possible.

3.3.3 Post-Processing

Ultimately, the purpose of the footstep evaluation network is to provide footstep candidates to the GaitNet model. Figure 3.8 shows the processing pipeline. The raw cost map output (Figure 3.8a) from the footstep evaluation network is first upsampled (Figure 3.8b) to increase the resolution of possible footstep positions, and to match the resolution of the terrain data. Next, noise is added (Figure 3.8c) to encourage exploration of more varied footstep positions. Without noise, all of the footstep candidates would be right next to each other. The cost map is then filtered based on the robot state and terrain data (Figure 3.8d) to mask out invalid actions, including positions that are too close to terrain edges, and actions which would attempt to move a leg already in the swing state (seen in the Front Right leg in this figure). Finally, the top 8 candidates from each leg are selected (Figure 3.8e) to be processed by GaitNet. In the case where there are fewer than 8 valid candidates for a leg, no-action candidates are added to the set to maintain the size of the tensors.



(a) Raw (b) Upsample (c) Noise (d) C-space (e) TopK (green)

Figure 3.8: Cost map processing pipeline. Shows how the raw cost map is processed to produce the final footstep candidates.

3.4 GaitNet

GaitNet is designed to select the best footstep action from a discrete set of footstep candidates. By choosing from a predefined set of continuous actions, we are able to enforce constraints on the robot's motion, strictly

prohibiting invalid foot placements, and limiting the possible gaits. The addition of a no-action candidate and continuous re-evaluation makes this algorithm novel, as it is able to greedily generate acyclic gaits with multiple feet in swing at once.

3.4.1 Architecture

GaitNet is tasked with ranking footstep candidates based on the footstep candidates with the leg index switched to one-hot encoding $\mathbf{f}'_c$ and the robot state $\mathbf{x}_g$:

$$\mathbf{x_g} = \begin{bmatrix} \mathbf{p}_{b,xy} \\ \mathbf{r}_{w,z} \\ \mathbf{v}_b \\ \omega_b \\ \mathbf{u} \\ \mathbf{g} \\ \mathbf{c} \end{bmatrix}$$

Descriptions of $\mathbf{p}_{b,xy}$ and $\mathbf{r}_{w,z}$ can be found in subsection 3.3.1. The additional terms included here in comparison to the footstep evaluation network are $\mathbf{g}$ and $\mathbf{c}$. Using $\mathbf{x}_g$ and $\mathbf{f}'_c$, it then outputs a logit associated with the footstep and the desired swing duration if that step is chosen.

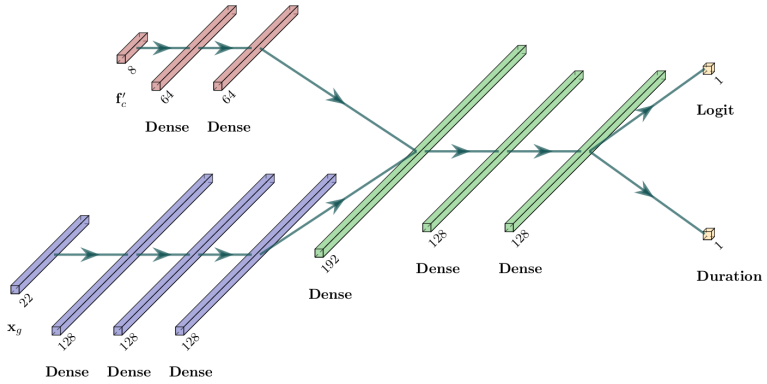


Figure 3.9: Gait net architecture. Sections of the diagram include the footstep candidate encoder in red, robot state encoder in blue, and shared trunk in green. The final output is a logit encoding the value of this option and the desired swing duration if this action is taken.

TODO: Use [18] somewhere

The proposed model is designed to jointly evaluate robot state and footstep candidates. The architecture consists of two encoders, a shared trunk, and two task-specific output heads:

- Robot state encoder. The robot state vector is processed by a three-layer feedforward encoder with intermediate dimensionality of 128, each layer has a ReLU activation. This produces a fixed-dimensional latent representation of the current robot state.
- Footstep encoder. Each footstep candidate is encoded by a similar two-layer network with hidden size 64, again using ReLU activations. No-action candidates are handled through a fixed embedding.
- Shared trunk. The concatenated robot state and footstep embeddings are processed by a three-layer feedforward "trunk," reducing the dimensionality while applying Layer Normalization and ReLU non-linearity.
- Output heads. From the shared trunk, two parallel prediction heads are applied

- a one-layer "logit head" outputs the predicted reward value as a logit.
- a one-layer "duration head" with sigmoid activation outputs a normalized swing duration, which is then scaled to a valid temporal range.

This design allows the network to evaluate multiple footstep candidates sequentially, assigning each both an expected value and a feasible swing duration, while explicitly supporting a "no-action" option via a dedicated embedding.

3.4.2 Training

By including the **c** term in the input, GaitNet cannot reasonably be trained with supervised learning because of the high dimensionality of the problem. Instead, it is trained using PPO [**?**]. For PPO, the actor network is GaitNet (Figure 3.9), and the critic follows the same architecture without the duration head, and with all of the logits passed through a final MLP to produce a single value estimate. The final MLP has two hidden layers of size 64, with ReLU activations on all but the last layer.

A custom actor critic implementation was also necessary to handle the dual outputs of GaitNet. The logits are used to define a categorical distribution over the footstep candidates, and the duration output is treated as a separate normal distribution with a fixed standard deviation of 0.01 s for each action. For the logits, all but one no-action candidates are removed (set to $-\infty$) to avoid overly influencing the selection process.

The total log probability of an action is the sum of the categorical log probability of the footstep candidate and the log probability of the duration under the normal distribution. To sample an action, a footstep candidate is first sampled from the categorical distribution, and then the duration is sampled from the normal distribution associated with that footstep candidate. The duration and footstep candidate are then combined to form a single footstep action.

TODO: detail rewards

TODO: detail events

TODO: detail terminations

4 Results and Discussion

4.1 Footstep Evaluation Network

TODO: This section will show the results of training the Footstep Evaluation Network. Specifically, we need to show that our implementation is sufficiently similar to [1]

The results of the Footstep Evaluation Network are very promising, with the model able to predict footstep candidate maps with high accuracy. Figure 4.1 shows the model output and ground truth for typical data sample. Figure 4.2 shows a particularly challenging data sample, and the model is still able to identify the best positions for each leg, particularly the back left leg, which needs to be far from the nominal position to maintain stability in that state. As it is used in this work, the specific accuracy of this model is not critical, as it is only used to sample the continuous space of foot positions to generate candidate actions for the GaitNet policy.

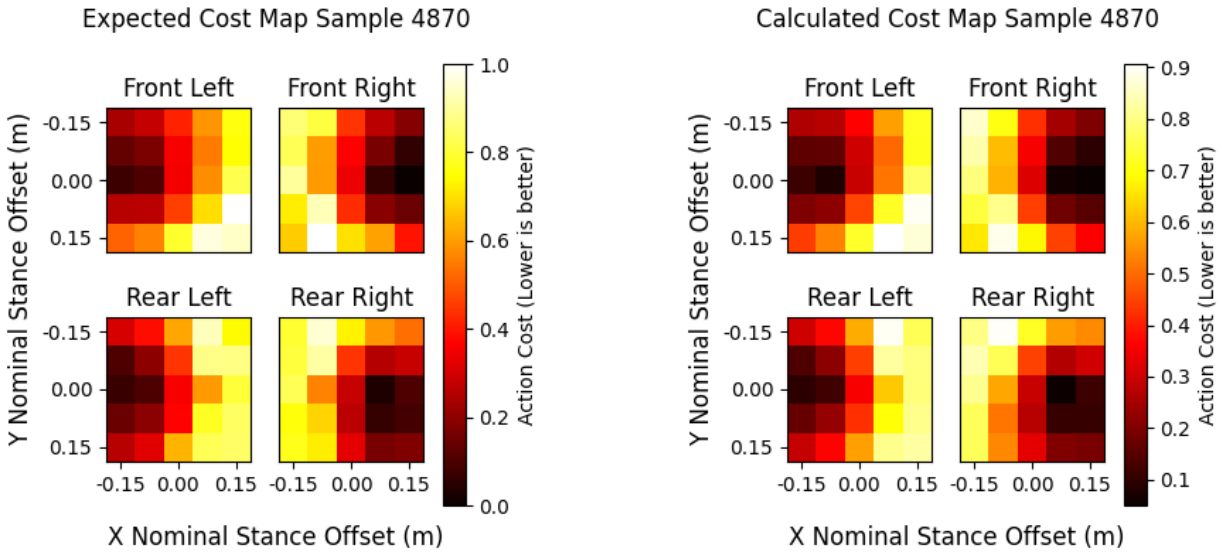


Figure 4.1: Typical data samples showing calculated (left) and expected (right) quadruped images.

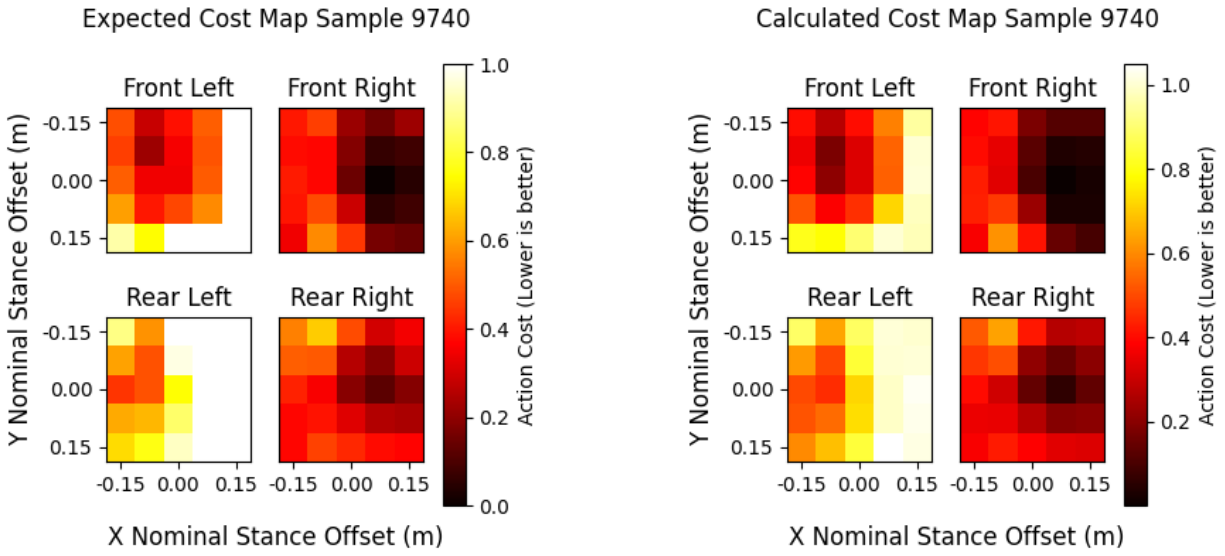


Figure 4.2: Particularly challenging data samples showing calculated (left) and expected (right) quadruped images.

TODO: Provide results showing how the model performs in situations like those in [1] (single foot steps on our terrain). This should highlight that this is a correct implementation of their method.

4.2 GaitNet

TODO: this section will show the results of training GaitNet. specifically, we need to show that the model is able to generate dynamic and acyclic gaits. We will not be comparing with the baseline method in this section.

The primary goal of GaitNet is to generate fully dynamic and acyclic gaits for quadruped robots. An example swing sequence can be seen in Figure 4.3. This shows an acyclic gait and dynamic gait; it is non-repetitive and the robot performs dynamically unstable motions with up to two feet off of the ground at once. Additionally, the swing durations are non-uniform, with some legs swinging for longer than others. It is clear that the system is able to synthesize a wide variety of actions, the value of which will be evaluated in this section.

Further analyzing Figure 4.3, we can see that a particularly short step was taken by the front left leg (green) at 3s, while a longer step was taken by the rear right leg (red) at 5s. Although the difference in duration is minor, this shows that the system is capable of dynamically varying the step duration.

The other takeaway from Figure 4.3 is the system's response to changing control inputs. Between 0-7.5s, the system is commanded to follow a slow control input. After 7.5s, the control input is changed to a faster speed. In the slow section, the system is very cautious, only taking one step at a time. However, in the fast section, the system exhibits a much more dynamic gait. At multiple points the system can be seen to take two steps at a time, while also not adhering to a strict alternating pattern.

TODO: Modify figure Figure 4.3 to show the terrain/robot at multiple time stamps.

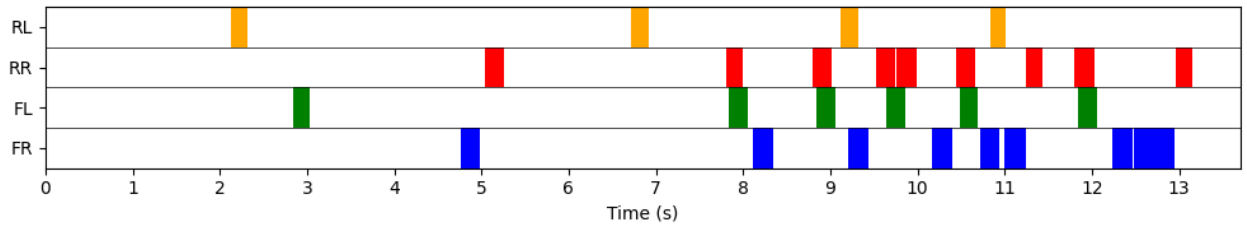


Figure 4.3: Example swing schedule for a single gait cycle. Each row represents a leg, with color indicating the leg is in swing phase.

TODO: Include figure showing the distribution of swing durations.

TODO: Include training figures from tensorboard detailing how the model improves over time.

4.3 Swing Duration Analysis

TODO:
quantify the improvement from dynamic swing duration

- compare the training time and final performance of a model with fixed swing durations vs dynamic swing durations

4.4 Action Cost Ablation Study

TODO:
explore the correlation between logits and candidate action costs

- starts off very high 0.8, drops to 0.4 during training
- compare training time and final performance of a model trained with and without the action cost in the reward function
- hypothesis: The action cost is useful at the beginning of training to help the model quickly learn the robot dynamics.
- note: if the action cost and logits are very poorly correlated, then there is possible improvement in the candidate action sampling method.

4.5 Baseline Comparison

Figure 4.4 shows the performance of GaitNet compared to the baseline method across a variety of terrain difficulties and commanded velocities. The results indicate that GaitNet outperforms the baseline method in most scenarios, particularly in more challenging terrains and at higher commanded speeds. This demonstrates the effectiveness of GaitNet in generating robust gaits that can adapt to varying conditions.

Further analyzing these graphs, we can see that GaitNet maintains excellent performance when the commanded velocity is under 0.15 m/s or the terrain difficulty is below 5%. ContactNet on the other hand, begins to degrade in performance for commanded velocities above 0.05 m/s and increasingly with terrain difficulties. GaitNet's superior performance in these scenarios highlights the advantages of its dynamic and acyclic gait generation capabilities.

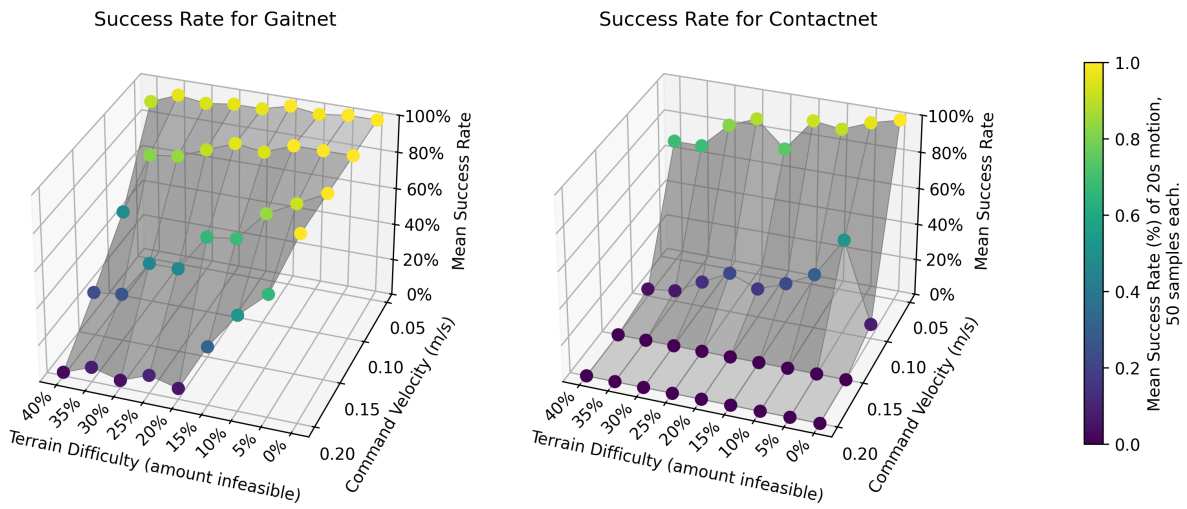


Figure 4.4: Evaluation of GaitNet and the baseline method for different terrain difficulties and commanded velocities.

5 Conclusions and Future Work

5.1 Summary of Findings

5.2 Limitations

- only proven in simulation
- no-lookahead limits performance at higher speeds
- currently appears to be limited by accuracy of footstep placements by low level controller
- GaitNet limited by the effectiveness of the underlying action candidate sampling method.

5.3 Future Work

- Real-World Deployment
- Swing re-planning
  - Currently, GaitNet doesn't re-plan at all during swings.
  - This would be trivial to add to GaitNet, but would require a more robust low level controller.
- Long horizon planning
  - Currently GaitNet is only trained to output actions for a single point in time.
  - The current implementation causes issues with the MPC not accurately knowing the future contact state. It looks like after every swing the robot will return to a nominal stance.
  - This does not create significant issues at the low speeds being explored in this work, but would be problematic at faster speeds.
- GPU accelerated MPC
  - Currently the MPC is run on the CPU, which seriously limits the speed of RL learning.
  - Some works explore GPU accelerated MPCs, which could be used to greatly speed up training.
- Simultaneous actions
  - Currently GaitNet only selects one action at a time.
  - A method of selecting the two best actions at once was briefly explored, but was abandoned due to poor learning, and implementation challenges.
  - specifically, I don't think I was able to get the gradients to flow properly through the two action selections.
- Sampler improvements

- Currently GaitNet relies on a sampling based method to select actions.
- It should be possible to directly search the action space. This could be done with projected gradient ascent to handle the sparse action space, or some sort of diffusion?
- This would increase the quality of solutions at the cost of compute time.
- Unsure of how this would work with learning.
- Could also train candidate selection network in tandem with GaitNet to improve sampling quality.

5.4 Final Remarks

References

[1] A. Bratta, A. Meduri, M. Focchi, L. Righetti, and C. Semini, "ContactNet: Online multi-contact planning for acyclic legged robot locomotion,"

[2] H. Shi, Q. Zhu, L. Han, W. Chi, T. Li, and M. Q.-H. Meng, "Terrain-aware quadrupedal locomotion via reinforcement learning."

[3] *GLiDE: Generalizable Quadrupedal Locomotion in Diverse Environments with a Centroidal Model*, pp. 523–539. Springer International Publishing. ISSN: 2511-1256, 2511-1264.

[4] H. Duan, A. Malik, J. Dao, A. Saxena, K. Green, J. Siekmann, A. Fern, and J. Hurst, "Sim-to-real learning of footstep-constrained bipedal dynamic walking," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 10428–10434, IEEE.

[5] J. Siekmann, K. Green, J. Warila, A. Fern, and J. Hurst, "Blind bipedal stair traversal via sim-to-real reinforcement learning."

[6] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," vol. 5, no. 47.

[7] C. Gaspard, G. Passault, M. Daniel, and O. Ly, "FootstepNet: an efficient actor-critic method for fast on-line bipedal footstep planning and forecasting," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 13749–13756, IEEE.

[8] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, "DeepGait: Planning and control of quadrupedal gaits using deep reinforcement learning."

[9] S. Omar, L. Amatucci, G. Turrisi, V. Barasuol, and C. Semini, "Fast convex visual foothold adaptation for quadrupedal locomotion,"

[10] O. Villarreal, V. Barasuol, M. Camurri, L. Franceschi, M. Focchi, M. Pontil, D. G. Caldwell, and C. Semini, "Fast and continuous foothold adaptation for dynamic locomotion through CNNs," vol. 4, no. 2, pp. 2140–2147.

[11] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "DeepLoco: dynamic locomotion skills using hierarchical deep reinforcement learning," vol. 36, no. 4, pp. 1–13. Publisher: Association for Computing Machinery (ACM).

[12] I. Taouil, L. Amatucci, M. Khadiv, A. Dai, V. Barasuol, G. Turrisi, and C. Semini, "Non-gaited legged locomotion with monte-carlo tree search and supervised learning," vol. 10, no. 2, pp. 1265–1272. Publisher: Institute of Electrical and Electronics Engineers (IEEE).

[13] Z. Gao, X. Chen, Z. Yu, C. Li, L. Han, and R. Zhang, "Global footstep planning with greedy and heuristic optimization guided by velocity for biped robot," vol. 238, p. 121798. Publisher: Elsevier BV.

[14] M. Zucker, N. Ratliff, M. Stolle, J. Chestnutt, J. A. Bagnell, C. G. Atkeson, and J. Kuffner, "Optimization and learning for rough terrain legged locomotion," vol. 30, no. 2, pp. 175–191. Publisher: SAGE Publications.

[15] M. Asselmeier, Y. Zhao, and P. A. Vela, "Steppability-informed quadrupedal contact planning through deep visual search heuristics."

[16] S. Omar, L. Amatucci, V. Barasuol, G. Turrisi, and C. Semini, "SafeSteps: Learning safer footstep planning policies for legged robots via model-based priors," in *2023 IEEE-RAS 22nd International Conference on Humanoid Robots (Humanoids)*, pp. 1–8, IEEE.

[17] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg, "Orbit: A unified simulation framework for interactive robot learning environments," vol. 8, no. 6, pp. 3740–3747.

[18] D. Feng, C. Haase-Schütz, L. Rosenbaum, H. Hertlein, C. Gläser, F. Timm, W. Wiesbeck, and K. Dietmayer, "Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1341–1360, 2021.

A GaitNet Training Configuration

A.1 Reward Function Analysis

The reward functions used to train GaitNet (see Table A.1) proved challenging to design. With many combinations of reward functions, the robot would exhibit undesirable behaviors, such as frequently lifting its feet when idle, or dragging feet behind while walking. The specific balance of *op_reward* and *mdp.foot_slip_penalty* were particularly important to achieve a natural gait.

Function[1]	**Weight**	**Description**
mdp.is_alive	0.4	Reward for being alive
mdp.track_lin_vel_xy_exp	0.5	Reward for tracking linear velocity in the XY plane
mdp.track_ang_vel_z_exp	0.5	Reward for tracking angular velocity around the Z axis
op_reward	-0.1	Penalty for performing actions
mdp.is_terminated	-200	Penalty for termination
mdp.lin_vel_z_l2	-2.5	Penalty for linear velocity in the Z direction
mdp.ang_vel_xy_l2	-0.1	Penalty for angular velocity in the XY plane
mdp.flat_orientation_l2	-8	Penalty for non-flat orientation
mdp.foot_slip_penalty	-6	Penalty for foot slip

Table A.1: Final reward functions used to train GaitNet.

In addition to the rewards listed in Table A.1, the following reward functions were explored but ultimately not used:

a_foot_in_swing
A reward for a foot being in the swing phase. This was intended to help the agent learn to frequently lift its feet, in the early stages of learning. In practice, this reward was found to be unnecessary, as initial network weights already heavily favored moving the feet. Additionally, it would cause the agent to always move its feet when idle.

no_op_reward
A reward for not performing any actions. This was intended to encourage the agent to remain idle when no useful actions were available. In practice, this reward would not provide useful information unless very highly weighted, at which point it would drown out other rewards. *op_reward* was found to be a more suitable alternative to achieve the desired behavior.

A.2 Termination Functions

The termination functions used in training GaitNet are detailed below.

[1]Functions named "mdp.*" are built-in functions provided by the NVIDIA Isaac Lab framework.
[2]Functions named "mdp.*" are built-in functions provided by the NVIDIA Isaac Lab framework.
[3]Time Out indicates whether the termination applies the mdp.is_terminated penalty.

Function[2]	Time Out[3]	Description
mdp.time_out	True	Terminate at the end of the episode
mdp.bad_orientation	False	Terminate if the robot's orientation is too far from upright
mdp.root_height_below_minimum	False	Terminate if the robot's base height is too low
mdp.terrain_out_of_bounds	True	Terminate if the robot leaves the terrain bounds
foot_in_void	False	Terminate if any foot steps into the void

Table A.2: Final termination functions used to train GaitNet.

A.3 PPO Hyperparameters

The PPO hyperparameters used to train GaitNet are detailed below. These were not all rigorously tuned, but were found to work well in practice. *gamma* was specifically chosen in relation to the agent observation frequency of 25 Hz, providing a reasonable discount length. The *learning_rate* was also specifically chosen to be relatively high, to make up for the slow data collection speed.

Hyperparameter	Value
clip_param	0.3
num_learning_epochs	8
num_mini_batches	4
value_loss_coef	0.5
entropy_coef	0.02
learning_rate	3e-4
max_grad_norm	1.0
use_clipped_value_loss	True
gamma	0.99
lam	0.95

Table A.3: Final hyperparameters used for PPO training.