GaitNet: Greedy, Dynamic Quadruped Gait Generation

Owen Sullivan

Supervisors:
Prof. Mahdi Agheli

Worcester Polytechnic Institute

October 20, 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
MCTS	**M**onte **C**arlo **T**ree **S**earch

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 significant challenge, particularly when traditional gait cycles prove inadequate. While many existing systems rely on periodic gait patterns or centralized planners, recent advances in learning-based methods have enabled more adaptive, non-gaited approaches. This study proposes a novel method for generating non-gaited, dynamic footstep plans using a neural network-based greedy planner, aiming to achieve a balance between computational efficiency and dynamic performance.

1.1 Motivation and Vision

Quadruped control pipelines can be categorized into several approaches, most of which can be broadly classified as traditional, neural network-based, or hybrid methods.

Traditional methods often rely on MPC for optimization, combined with lower-level controllers to track the desired motions. Many aspects of quadruped locomotion—such as joint torques and ground reaction forces—are smooth and can be efficiently optimized. However, these methods become less practical when optimizing over the robot's discrete contact states. Typically, they employ pre-defined gait sequences or embed discrete contact decisions into a more complex optimization problem. This trade-off performs well in general settings, but ultimately limits the unique capabilities of quadruped robots.

Neural network-based methods have gained traction in recent years, particularly with the advancement of reinforcement learning techniques. These approaches often replace the MPC component of traditional pipelines, directly mapping state information to joint commands. In doing so, they implicitly address the discrete contact problem, allowing the network to learn the mixed-integer optimization implicitly. Although this provides greater flexibility, such methods sacrifice the formal guarantees of traditional control, demand extensive training data, and frequently exhibit poor generalization to novel environments.

Hybrid methods, which combine elements of both approaches, form the focus of this thesis. These methods leverage neural networks to address specific challenges—such as contact or foothold planning—while relying on traditional controllers for carrying out higher level tasks. This synergy enables the neural network to concentrate on the more complex and non-smooth aspects of locomotion, while maintaining the stability and robustness offered by model-based control frameworks.

1.2 Research Gap

Traditional methods have continually faced challenges in footstep planning. Some approaches attempt to incorporate the discrete aspects of contact planning directly into the optimization problem (TOWR), but this introduces significant computational complexity, making real-time solutions difficult. Other methods rely on pre-defined gait sequences (Raibert-style gaits, etc.), which constrain the robot's ability to fully exploit the versatility of quadruped locomotion. Although substantial progress has been made in MCTS-based footstep planners for classical controllers, these approaches remain computationally demanding.

Despite the advantages and recent advancements in hybrid control methods, a notable gap persists in the literature. Specifically, there is a lack of hybrid approaches capable of achieving the robust contact planning performance of neural network-based systems while preserving the stability and reliability of traditional control frameworks. Recent work (Contactnet) has demonstrated promise in generating footstep plans

through machine learning; however, this method is constrained to moving one leg at a time for a pre-defined duration—falling short of the fully dynamic and acyclic gaits observed in end-to-end learning systems.

This motivates a central research question:

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

1.3 Hypothesis

To address the identified research gap, we hypothesize that a hybrid control pipeline incorporating a greedy, neural network-based footstep planner can effectively generate dynamic, non-gaited locomotion for quadruped robots in challenging environments. By leveraging the advancements introduced by ContactNet and extending its capabilities to support multi-leg actions, we propose that such a planner can produce footstep sequences that enable the robot to traverse complex terrains while maintaining both stability and efficiency.

The proposed approach employs a modified implementation of ContactNet's architecture, adapted specifically to generate footstep candidates. These candidates will be evaluated and ranked by a novel planner, *GaitNet*, which selects the most appropriate action at each time step. By integrating this planner within a traditional control framework, we aim to harness the advantages of both learning-based and model-based methods, yielding a robust and adaptable quadruped locomotion system.

1.4 Contributions

- Development of a novel footstep planner utilizing greedy planning with a neural network, for use in hybrid control of quadruped robots.
- Demonstration that the proposed planner can generate dynamic, acyclic gaits in challenging environments.
- Empirical evaluation showing that the planner achieves comparable diverse expressiveness with fast inference times.

1.5 Thesis Structure

- **Chapter 2: Background** - Covers foundational concepts in quadruped locomotion, gait generation, and machine learning techniques relevant to this work.
- **Chapter 3: Methodology** - Details the design and implementation of the CNN-based greedy planner, including network architecture and training procedures.
- **Chapter 4: Results and Discussion** - Presents experimental results, evaluates the performance of the proposed planner, and discusses its strengths and limitations.
- **Chapter 5: Conclusions and Future Work** - Summarizes the key findings of the thesis and outlines potential directions for future research.

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.

3 Methodology

3.1 System Overview

This work presents a control framework for quadruped robots capable of generating dynamic, acyclic gaits in challenging environments. The framework is hierarchical, processing robot state and terrain data to produce footstep commands for 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, a gait generation network (ContactNet), takes these candidates $\mathbf{f_c}$, ranks them, and outputs the optimal footstep action $\mathbf{f_a}$ to the robot controller.

This hierarchical design enables strict control over the range of possible actions the robot can execute. Between the two stages, candidate actions are filtered to ensure that only valid, prescribed movements are permitted.

TODO: update Figure 3.1

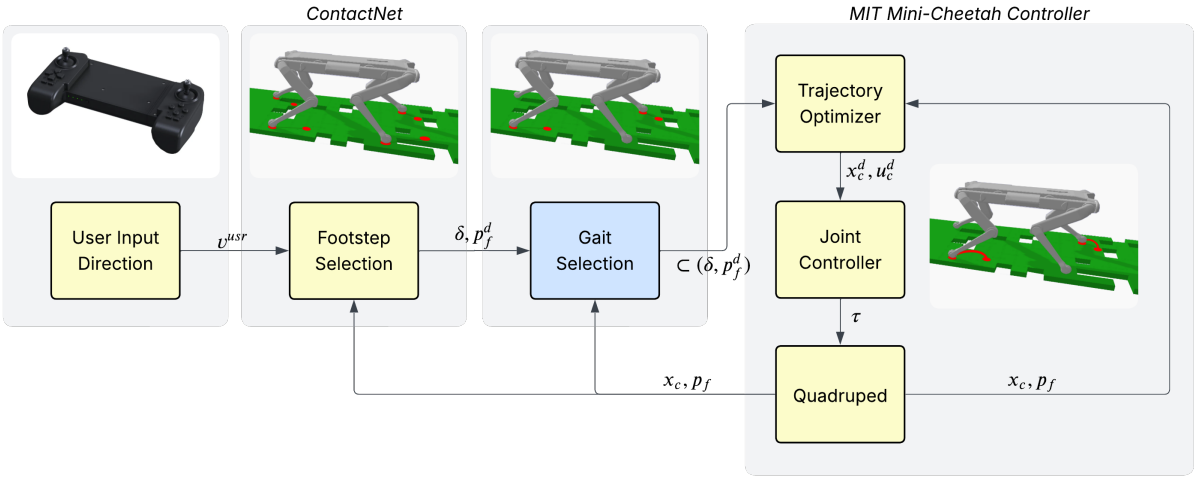


Figure 3.1: A block diagram of the proposed framework. The user defines an input direction v^{usr} which the *footstep selector* [1] uses along with, the robot state x_c, and the current foot positions p_f to generate the swing durations δ and touchdown points p_f^d for all currently grounded feet. The *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 framework was selected for several reasons. It is a modern platform with a Python interface designed specifically for machine learning applications and GPU parallelism. The use of GPU parallelism enables significantly faster simulation and data collection, albeit at the cost of increased programming complexity and reduced compatibility with older hardware. Furthermore, the extensive collection of example and community projects provides valuable references for implementing the simulation features required in this work.

Although both simulation and learning processes are executed on the GPU, the robot controllers operate on the CPU. Figure 3.2 illustrates the overall processing flow. The simulation environment runs entirely on the GPU, where multiple robots are simulated in parallel. Meanwhile, the robot MPCs execute on the CPU, with each controller running concurrently. The CPU and GPU communicate at every simulation step to exchange robot states and corresponding control actions.

TODO: add in learning step

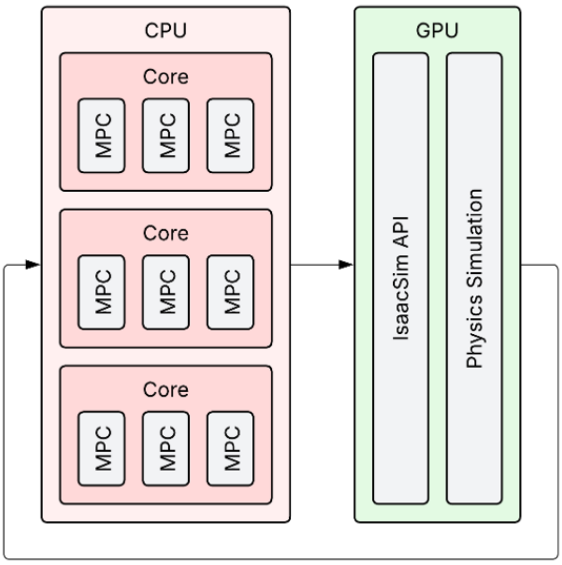


Figure 3.2: A block diagram showing the programming tasks computed on the CPU vs GPU. The full simulation is run in parallel using Nvidia Isaac Lab on the GPU, while the robot MPCs are run in parallel on the CPU.

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

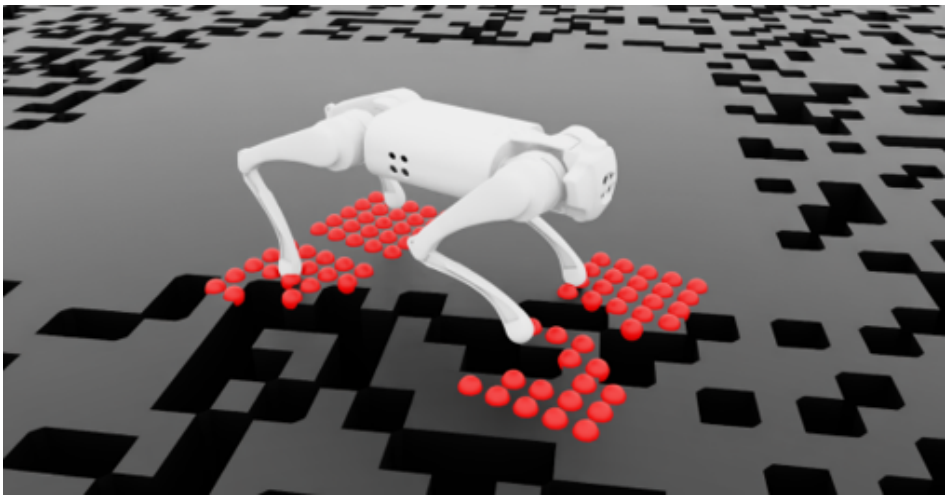


Figure 3.3: Image of a Unitree Go 1 navigating the terrain. Red spheres show the hit location of raycasts. Black regions show voids in the terrain.

3.3 Footstep Evaluation Network

The purpose of the footstep evaluation network is to generate footstep candidates for the GaitNet model. Importantly, the precise output of this network is not critical; rather, it is essential that the network provides high-quality candidates when sampled. It achieves this by estimating the cost associated with potential footsteps given the current robot state. The network's architecture and training process are largely based on ContactNet [1], with several key modifications, which are described in detail below.

3.3.1 Architecture

The footstep evaluation network is responsible for generating a set of footstep candidates $\mathbf{f_c}$ based on the robot state $\mathbf{x_c}$:

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

Here, $\mathbf{p}_{b,xy}$ represents the x and y positions of all end effectors in the base frame, stacked into a single vector, and $\mathbf{r}_{w,z}$ denotes the height of the robot's center of mass in the world frame. The inclusion of ω_b distinguishes this formulation from that in [1]. This was with the intent to improve the model's performance in situations with high rotational velocities.

The network is trained on heuristically computed footstep cost maps (Figure 3.7b), which estimate the cost of moving each foot to candidate positions given the current robot state. These maps are represented as four 5×5 grids, one for each leg. This choice also differs from the implementation in [1]. This grid size was selected to provide more information for each foot, important later when sampling footstep candidates.

The architecture of the footstep evaluation network is shown in Figure 3.4. It consists of a feedforward neural network that initially 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 8×8 spatial representation and processed by a convolutional layer with two output channels, a 3×3 kernel, stride 1, and padding 1, followed by a ReLU nonlinearity. The output is flattened and passed through a final fully connected layer

before being reshaped into a 5×5 grid. Again, this specific architecture differs from [1], where a larger network without the convolutional layer was used. These specific changes were found to provide better results for the 5×5 output grid.

To produce the four footstep candidate maps, this network is replicated four times—once per leg—with weights not shared between legs. This design allows the network to learn leg-specific behaviors, accommodating potential asymmetries in the robot's dynamics.

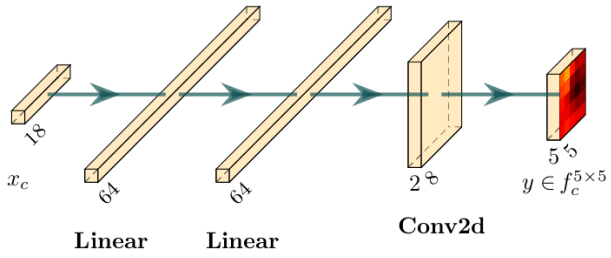


Figure 3.4: Footstep evaluation neural network architecture.

3.3.2 Training

The footstep evaluation network is trained to predict heuristic footstep cost maps, as described in [1]. Training data is generated using the simulation environment outlined in section 3.2.

During training, 100 robots are simulated in parallel, divided into four groups of 25. Each group tests a grid of footstep positions for one foot at a time; the front-left group is illustrated in Figure 3.5. An *iteration* is considered complete once all robots have either successfully completed or failed their assigned motions. Importantly, all robots begin each iteration from the same initial state.

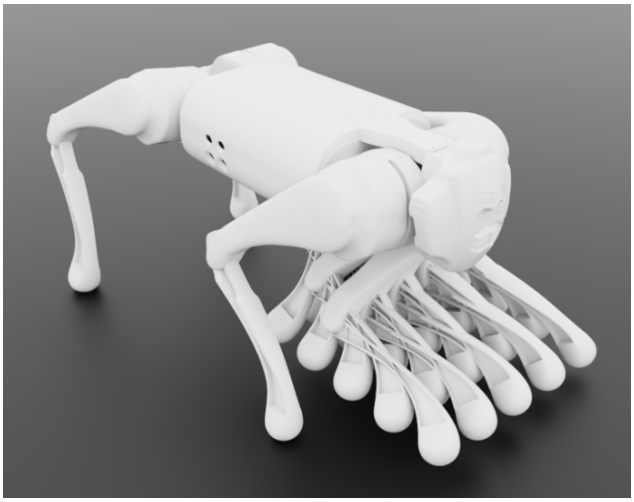


Figure 3.5: Snapshot showing 25 robots testing different footstep positions in parallel. For real data generation, 100 are run in parallel, testing 25 footstep positions for each foot at a time.

Data collection proceeds by chaining multiple iterations together, with control inputs periodically updated. After each iteration, the top 10 actions are inserted into a tree structure as edges, with the resulting state as the leaf node. The tree is explored by randomly selecting leaves for expansion, and this process continues until a predefined maximum number of iterations is reached. To ensure data quality, iterations in which more than 50% of the robots fall are discarded, along with their two parent nodes in the tree. This approach promotes the collection of diverse, yet successful, training data. This implementation differs in the specifics to [1], while still remaining faithful to the overall methodology.

Figure 3.6 illustrates the distribution of foot positions in the training dataset. Darker regions correspond to discrete points on the 5×5 footstep grids; a foot occupies a given position if it was moved there in that iteration.

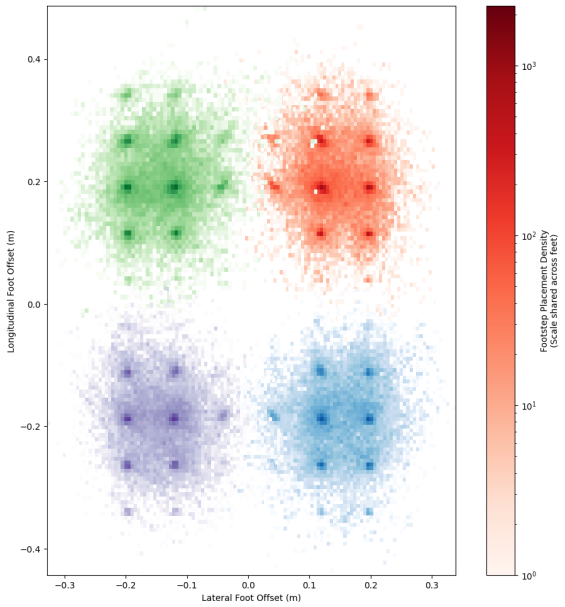
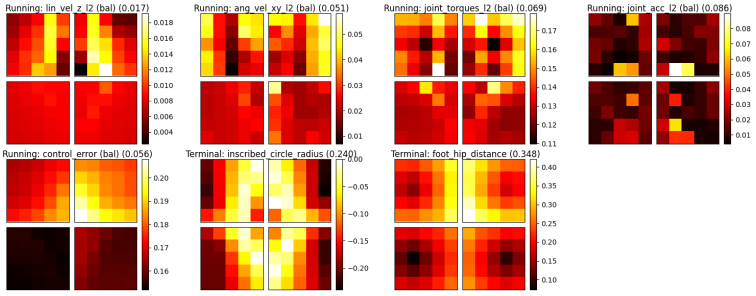


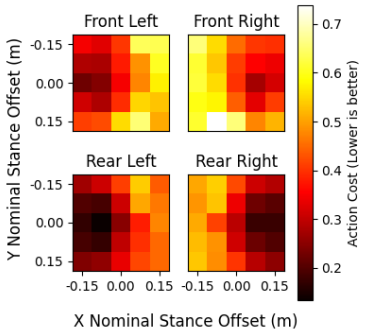
Figure 3.6: Foot placement heatmaps showing distribution of foot positions in the GaitNet training data. Note that the histograms are overlaid in some places, obscuring data underneath.

The purpose of this data collection is to assign a cost to each potential footstep, forming the candidate set $\mathbf{f_c}$. Costs are computed heuristically based on simulation outcomes, balancing stability and efficiency. While similar to the approach in [1], our heuristic differs in several aspects to better suit the output for the footstep candidate sampling. Figure 3.7a shows the individual factors used to construct the footstep candidate maps, and Figure 3.7b displays the resulting combined cost map. These factors are described 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.

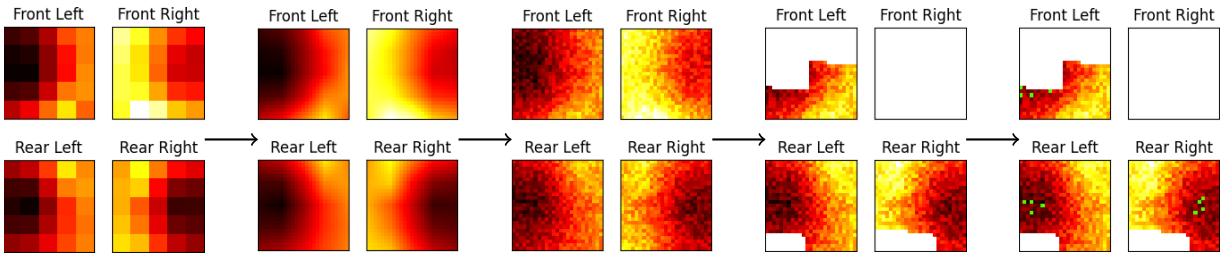
As in [1], the cost maps are normalized to enhance training performance. Our approach differs in that the maps are normalized directly to the range $[0, 1]$, rather than preserving only the relative ordering of costs as in [1]. This direct normalization is crucial for providing the upstream GaitNet model with maximal information.

3.3.3 Post-Processing

The primary purpose of the footstep evaluation network is to provide high-quality footstep candidates to the GaitNet model. Figure 3.8 illustrates the processing pipeline.

The raw cost map output from the footstep evaluation network (Figure 3.8a) 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 a more diverse set of footstep positions; without this noise, all candidates would cluster in close proximity.

The cost map is then filtered based on the robot state and terrain data (Figure 3.8d) to mask out invalid actions. This includes positions that are too close to terrain edges and movements that would attempt to reposition a leg already in the swing phase (as illustrated by the Front Right leg in the figure). Finally, the top 8 candidates from each leg are selected (Figure 3.8e) for processing by GaitNet. If fewer than 8 valid candidates exist for a given leg, no-action candidates are added to maintain consistent tensor dimensions.



(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 optimal footstep action from a discrete set of candidates. By restricting selection to a predefined set of continuous actions, the algorithm enforces constraints on the robot's motion, strictly preventing invalid foot placements and limiting the range of possible gaits. The inclusion of a no-action candidate, combined with continuous re-evaluation, distinguishes this approach: it enables the greedy generation of acyclic gaits in which multiple feet can be in the swing phase simultaneously.

3.4.1 Architecture

GaitNet is responsible for ranking footstep candidates based on the one-hot encoded leg index $\mathbf{f_c}'$ 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, compared to the footstep evaluation network, are $\mathbf{g}$ and $\mathbf{c}$. Using $\mathbf{x_g}$ and $\mathbf{f_c}'$, GaitNet outputs a logit associated with each footstep and the desired swing duration if that step is selected.

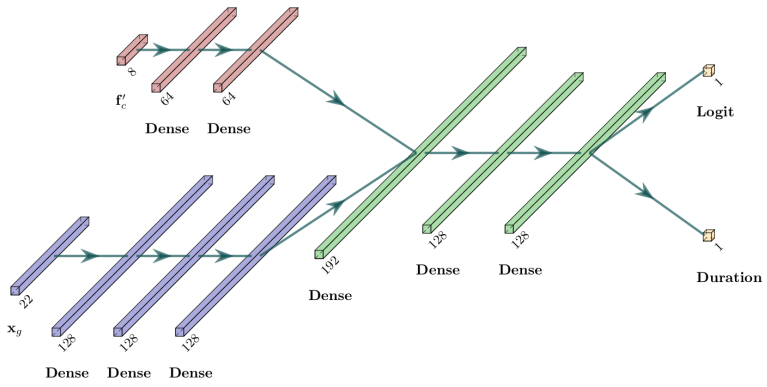


Figure 3.9: Gait net architecture. Sections of the diagram include the footstep candidate encoder in red, robot state encoder in blue, and shared trunk in green. The final output is a logit encoding the value of this option and the desired swing duration if this action is taken.

TODO: Use [18] somewhere

The 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 network with intermediate dimensionality of 128 and ReLU activations. This produces a fixed-dimensional latent representation of the current robot state.
- Footstep encoder: Each footstep candidate is encoded by a two-layer feedforward network with hidden size 64 and ReLU activations. No-action candidates are represented through a fixed embedding.

- Shared trunk: The concatenated robot state and footstep embeddings are processed by a three-layer feedforward trunk, reducing dimensionality while applying Layer Normalization and ReLU nonlinearity.
- Output heads: Two parallel prediction heads are applied to the shared trunk:
  - Logit head: A single-layer network outputs the predicted reward value as a logit.
  - Duration head: A single-layer network with sigmoid activation outputs a normalized swing duration, which is then scaled to a valid temporal range.

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

3.4.2 Training

Due to the inclusion of the $\mathbf{c}$ term in the input, GaitNet cannot be effectively trained using supervised learning because of the high dimensionality of the problem. Instead, it is trained using Proximal Policy Optimization (PPO) [**?**]. In this setup, the actor network is GaitNet (Figure 3.9), while the critic follows a similar architecture but omits the duration head. All logits from the critic are passed through a final multi-layer perceptron (MLP) to produce a single value estimate. This MLP consists of two hidden layers of size 64, with ReLU activations applied to all layers except the output.

A custom actor-critic implementation was necessary to handle GaitNet's dual outputs. The logits define a categorical distribution over footstep candidates, while the duration output is treated as a separate normal distribution with a fixed standard deviation of 0.01 s for each action. To prevent multiple no-action candidates from skewing the selection, all but one no-action candidate are removed (set to $-\infty$).

The total log probability of an action is computed as the sum of the categorical log probability of the selected footstep candidate and the log probability of the duration under the normal distribution. During action sampling, a footstep candidate is first sampled from the categorical distribution, followed by sampling the duration from its associated normal distribution. The footstep candidate and duration are then combined to form a complete 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 highly promising, with the model able to predict footstep candidate maps with strong accuracy. Figure 4.1 shows the model output alongside the ground truth for a typical data sample.

Figure 4.2 illustrates a particularly challenging sample, in which the model still successfully identifies the most suitable positions for each leg. Notably, the back-left leg must be positioned far from its nominal location to maintain stability in this scenario, and the model correctly predicts this adjustment.

In the context of this work, the precise accuracy of the model is not critical. Its primary role is to sample the continuous space of foot positions to generate candidate actions for the GaitNet policy.

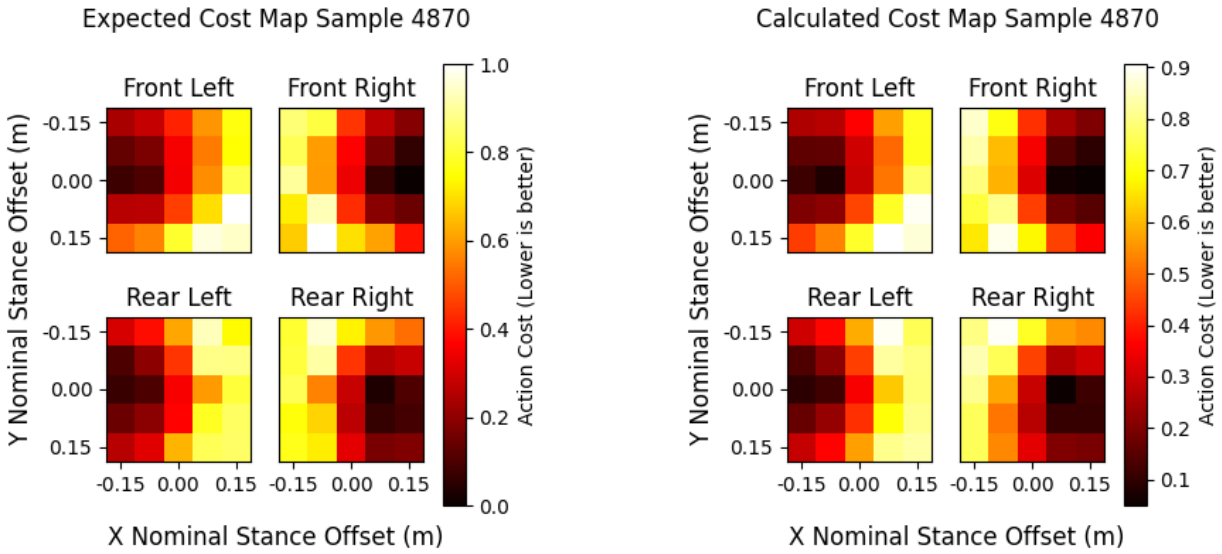


Figure 4.1: Typical data samples showing calculated (left) and expected (right) quadruped images.

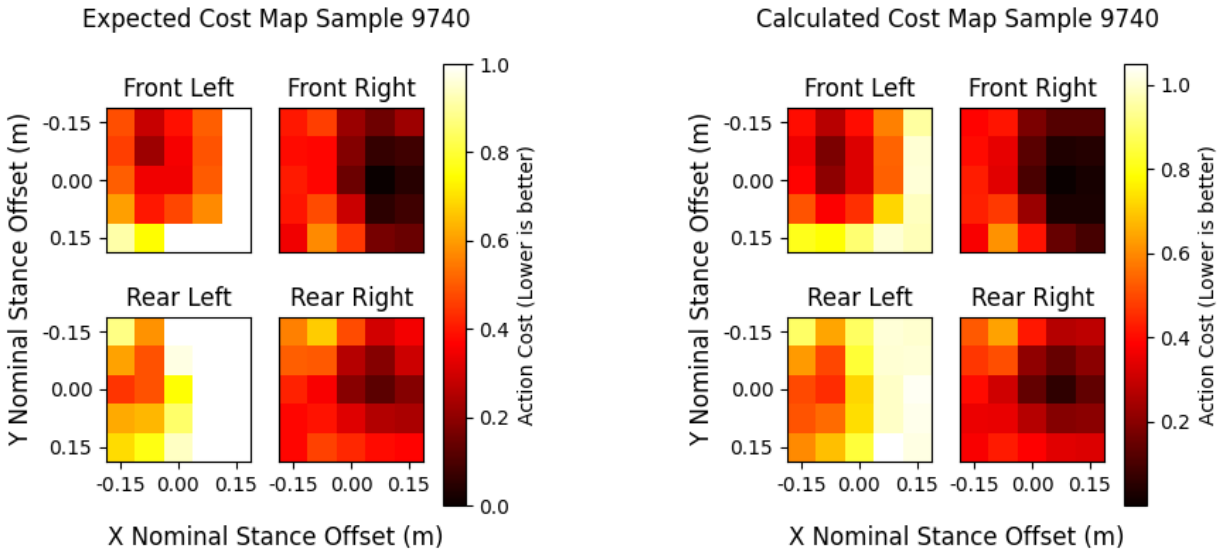


Figure 4.2: Particularly challenging data samples showing calculated (left) and expected (right) quadruped images.

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 objective of GaitNet is to generate fully dynamic and acyclic gaits for quadruped robots. An example swing sequence is shown in Figure 4.3, illustrating a non-repetitive, dynamic gait in which the robot performs motions with up to two feet off the ground simultaneously. Swing durations are non-uniform, with some legs remaining in the swing phase longer than others. These results demonstrate that the system can synthesize a wide variety of actions, which are analyzed further below.

Examining Figure 4.3, a short step is observed for the front-left leg (green) at 3 s, while the rear-right leg (red) executes a longer step at 5 s. Although the differences in duration are subtle, they indicate that the system is capable of dynamically varying step timing.

Another important observation from Figure 4.3 is the system's response to changing control inputs. Between 0-7.5 s, the robot is commanded to follow a slow input, and the system takes a cautious approach, moving one foot at a time. After 7.5 s, the input speed increases, prompting a more dynamic gait. During this faster phase, the robot occasionally executes two steps simultaneously and does not follow a strict alternating pattern, further highlighting the system's flexibility in generating acyclic gaits.

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

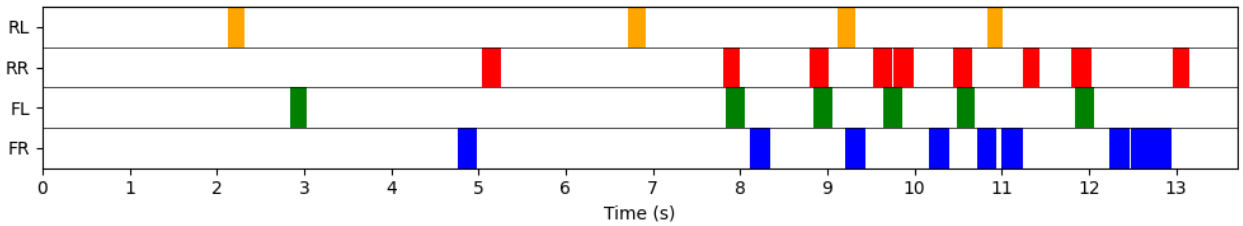


Figure 4.3: Example swing schedule for a single gait cycle. Each row represents a leg, with color indicating the leg is in swing phase.

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 illustrates the performance of GaitNet relative to the baseline method across a range of terrain difficulties and commanded velocities. The results indicate that GaitNet outperforms the baseline in most scenarios, particularly on more challenging terrains and at higher commanded speeds. This demonstrates the effectiveness of GaitNet in generating robust gaits capable of adapting to varying conditions.

A closer examination of the graphs reveals that GaitNet maintains strong performance when the commanded velocity is below 0.15 m/s or terrain difficulty is under 5%. In contrast, ContactNet's performance begins to degrade for commanded velocities above 0.05 m/s and deteriorates further as terrain difficulty increases. GaitNet's superior performance in these scenarios underscores the benefits of its dynamic, acyclic gait generation capabilities.

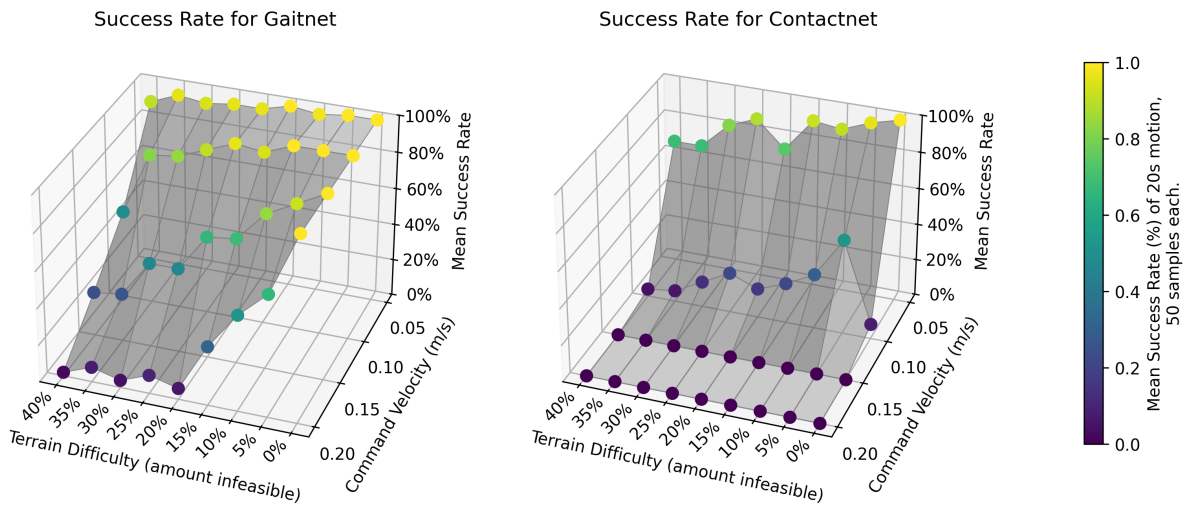


Figure 4.4: Evaluation of GaitNet and the baseline method for different terrain difficulties and commanded velocities.

5 Conclusions and Future Work

5.1 Summary of Findings

5.2 Limitations

While the results presented in this work are promising, several limitations should be acknowledged. First, the proposed framework has only been validated in simulation. Real-world deployment may introduce additional challenges, such as unmodeled dynamics, sensor noise, and hardware constraints, which could affect performance. Second, GaitNet operates in a greedy, no-lookahead fashion, limiting its effectiveness at higher speeds or in highly dynamic environments where predictive planning could improve stability and efficiency. Additionally, the system's performance is constrained by the accuracy of the low-level controller in executing footstep placements. Errors in tracking or positioning can degrade the overall effectiveness of the generated gaits. Finally, GaitNet's performance is inherently tied to the quality of footstep candidates produced by the footstep evaluation network; suboptimal candidate sampling may restrict both the diversity and effectiveness of the generated actions.

These limitations highlight clear directions for future research, including real-world testing on physical hardware, the incorporation of predictive planning strategies, and improvements to both candidate generation and low-level control precision.

5.3 Future Work

Several avenues exist to extend and improve the current framework. First, real-world deployment remains a crucial next step. Testing GaitNet on physical hardware would expose the system to real-world dynamics, sensor noise, and hardware constraints, providing valuable insights for further refinement.

Swing re-planning is another potential improvement. Currently, GaitNet does not re-plan during the swing phase. Incorporating real-time re-planning would allow the robot to adjust to unexpected disturbances or terrain changes, though this would require a more robust low-level controller capable of accurately tracking the modified footstep trajectories.

Long-horizon planning also presents an opportunity for enhancement. GaitNet is presently trained to generate actions for a single point in time, which limits the MPC's ability to predict future contact states accurately; the MPC expects the robot to revert to a nominal stance after each swing. While this is acceptable at low speeds, it could become problematic at higher velocities, where predictive planning would improve stability and performance.

Currently, the MPC runs on the CPU, limiting the speed of reinforcement learning. Leveraging GPU-accelerated MPCs, as explored in recent works [**?**], could significantly accelerate training.

GaitNet also selects only one action at a time, leading to slightly staggered swing start times. Initial attempts to select multiple simultaneous actions encountered difficulties with gradient flow and learning stability. Developing a reliable method for multi-action selection could enable more fluid and dynamic gait patterns, though it would require careful network and training design.

Finally, improvements to the action candidate sampler could enhance GaitNet's performance. At present, action selection relies on sampling-based methods, which may not always identify the highest-quality actions. Directly searching the action space using techniques such as projected gradient [**?**] ascent or other optimization

strategies could improve solution quality at the expense of additional computation. Another possibility is to train a candidate selection network jointly with GaitNet to improve the diversity and relevance of sampled actions.

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

Designing the reward functions for training GaitNet (see Table A.1) proved to be a challenging task. Various combinations of rewards often resulted in undesirable behaviors, such as the robot frequently lifting its feet while idle or dragging feet during locomotion. Achieving a natural gait required careful balancing, particularly between *op_reward* and *mdp.foot_slip_penalty*.

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, several other reward functions were explored but ultimately not used.

The *a_foot_in_swing* reward was intended to encourage the agent to frequently lift its feet during early learning. However, it proved unnecessary, as the initial network weights already favored foot movement. Furthermore, including this reward caused the agent to move its feet excessively when idle.

The *no_op_reward* was designed to encourage the agent to remain idle when no useful actions were available. In practice, this reward required an excessively high weighting to have any effect, at which point it would overshadow other rewards. The *op_reward* was found to be a more effective alternative for achieving the desired behavior.

A.2 Termination Functions

The termination functions used in training GaitNet are summarized in Table A.2. These functions help define the conditions under which an episode ends, either applying the termination penalty or simply signaling an invalid state.

[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 for training GaitNet are summarized in Table A.3. While not all parameters were rigorously tuned, they were found to perform well in practice. The discount factor *gamma* was specifically selected in relation to the agent observation frequency of 25,Hz to provide a reasonable effective horizon. The *learning_rate* was chosen relatively high to compensate for the slower 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.