Fast Footstep Planning on Uneven Terrain Using Deep Sequential Models

Hersh Sanghvi
School of Engineering and Applied Science
University of Pennsylvania
Email: hsanghvi@seas.upenn.edu

Camillo Jose Taylor
School of Engineering and Applied Science
University of Pennsylvania
Email: cjtaylor@seas.upenn.edu

Abstract— One of the fundamental challenges in realizing the potential of legged robots is generating plans to traverse challenging terrains. Control actions must be carefully selected so the robot will not crash or slip. The high dimensionality of the joint space makes directly planning low-level actions from onboard perception difficult, and control stacks that do not consider the low-level mechanisms of the robot in planning are ill-suited to handle fine-grained obstacles. One method for dealing with this is selecting footstep locations based on terrain characteristics. However, incorporating robot dynamics into footstep planning requires significant computation, much more than in the quasi-static case. In this work, we present an LSTM-based planning framework that learns probability distributions over likely footstep locations using both terrain lookahead and the robot's dynamics, and leverages the LSTM's sequential nature to find footsteps in linear time. Our framework can also be used as a module to speed up sampling-based planners. We validate our approach on a simulated one-legged hopper over a variety of uneven terrains.

I. INTRODUCTION

Legged robots have significant mobility advantages over their wheeled counterparts. They can traverse stairs, uneven terrain and other hazards that would be impassible for more conventional ground robots. However, these attractive dynamic capabilities come at a cost. First, legged robots with limited perceptual lookahead must carefully plan motions to move across potentially rough terrain without crashing. This involves developing an effective mapping from perception to feasible actions that incorporates terrain geometry and the robot's high-dimensional and hybrid state dynamics. Second, factors such as model mismatch and perception error force frequent online replanning. These two requirements significantly constrain the possible approaches. Searching over the full state space would certainly result in a feasible plan, but would take far too long to be used online, while a planner using many approximations would operate quickly, but with riskier actions.

One commonly used approach is to plan footstep locations [1], [2], [3]. Footstep planning frames the problem as a discrete-time search and also allows the planner to incorporate both the geometry of the terrain and the robot's dynamics while considerably lowering the dimensionality of the search space. Making this search fast enough to run online in dynamic situations is still a significant challenge.

In this paper, we develop a framework that learns from offline plans to give explicit and interpretable probability
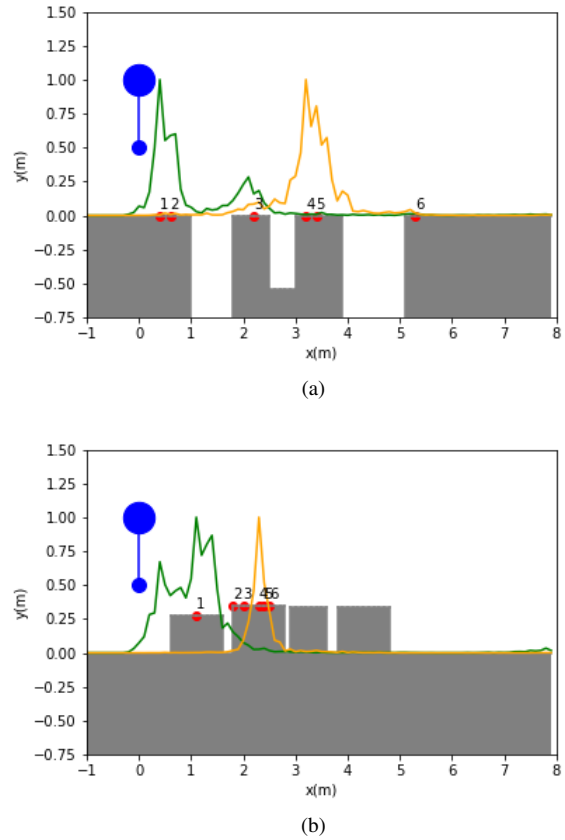

(a)

(b)

Fig. 1: Our footstep planner in action. The green and orange surfaces represent probability distributions over the terrain for the first and fourth steps, respectively. Note in particular how the planner accounts for the loss in kinetic energy on the elevated terrain.

distributions over the terrain conditioned both on perceptual input and the robot's dynamics for an arbitrary number of steps. This distribution can be used directly through its mode (argmax), or to speed up a sampling-based planner by suggesting informed samples. We demonstrate the merits of our approach using a simulated one-legged hopper on a variety of terrains. The proposed framework provides a computationally efficient approach to relating look-ahead perception to action for these systems.

II. Related Work

A. Graph Search and Optimization for Footstep Planning

Methods for footstep planning that do not use machine learning are typically based on solving either graph search or optimization problems. These methods often generate quasi-static trajectories or are too slow to be used online. On the graph search side, [4] uses best-first search over input angles for a one-legged robot, sampling possible leg angles at each impact to find the next node. For multi-legged robots, [5], [6], [7], [8] use a variety of graph search algorithms to plan only a Center of Mass (CoM) trajectory and use terrain heuristics to find footholds along that trajectory, while [1], [9], [10] explicitly search over footsteps based on stable stance transitions.

On the optimization side, most approaches make use of some relaxation or approximation of the underlying dynamics in order to speed up computation time. For example, [11] express step lengths as gait phases to find center-of-mass and footstep locations using nonlinear optimization, at the cost of not enforcing some swing foot collisions. [12] formulates footstep planning as a mixed-integer quadratic program and can solve for short footstep plans in under a second, but does not consider the robot dynamics. [13] uses a convex relaxation of the dynamics in order to generate 2 to 4 step plans in roughly one second. [2], [14] consider more complete dynamic models but the whole body optimizations take on the order of minutes to solve. Most of these optimization schemes could benefit from an independent, robust system for initializing the footstep locations which our method could provide.

B. Learning for Footstep Planning

Researchers have also utilized to learning-based methods to plan footstep locations. One popular method is to use Reinforcement Learning (RL) and learn a policy from visual input to select the optimal next footstep or action [15], [16], [17], [18], [19]. These methods are powerful in that they can leverage the full robot model, but usually only plan footsteps or motion primitives over a short horizon, and sometimes rely on hand-selected terrain features in the reward function. Variational Autoencoders have also been explored for footstep planning to deal with multimodality: the fact that there are many feasible choices for steps, depending on the desired behavior [20].

Another common approach is to incorporate learned modules into the more traditional planning frameworks. Previous methods include learning cost functions to speed up optimization or search [21], [22] learning forward dynamics models [23], adaptation of next footholds based on visual information and heuristics [24], [25], and direct regression of footsteps [26].

Recurrent Neural Networks (RNNs) are an interesting architecture for this problem as they naturally handle the sequential, discrete-time nature of the problem. While RNNs have been used in policies for blind low-level control of legged robots [27] and in other planning domains [28], [29],

to the authors' knowledge this is the first application of RNNs to the footstep planning problem. Our framework has a number of advantages over other learned footstep planners: no terrain heuristics are used for training, the plan can be arbitrarily long up to the visual horizon, multimodality is handled by outputting a probability density map over all possible step locations, and it can be used as a sub-module in a traditional sampling planner.

III. Problem Formulation and Approach

We envision a scenario where a legged robot has a limited perceptual lookahead and must plan its steps in order to cross the terrain in front of it without crashing. Specifying these step locations can provide enough information to generate a corresponding CoM trajectory [1]. The locations of the footsteps must be carefully chosen to be reachable while also avoiding collisions.

First, we set up our problem by describing the dynamics of our simulated robots. Next, we establish two baseline search-based planners that use these dynamics to search for footsteps over broken terrain. We then show how we can use these planners to train a recurrent model to output similar plans in a fraction of the computational time. Lastly, we demonstrate the performance of our recurrent planner on a variety of terrains.

IV. Modeling

A. Spring-Loaded Inverted Pendulum

For our work, we consider the classic planar Spring Loaded Inverted Pendulum (SLIP) hopper [30]: a one legged machine modeled as a massless, spring-loaded leg and foot attached to a point mass body. This model is used as a template [31] to plan trajectories of higher degree of freedom robots such as bipeds [32].

The SLIP hopper is a hybrid system with two modes. When the hopper is in the *flight* mode, the foot is not touching the ground and the only force experienced by the body is gravity. When the hopper is in the *stance* mode, the foot is in contact with the ground and the body experiences gravity and a spring force acting through the leg. The instantaneous transition from stance to flight is referred to as *liftoff*, and the transition from flight to stance is *touchdown*.

The passive SLIP model has one degree of freedom: the angle of the leg with respect to the ground. This input is applied at the apex of flight, where the vertical velocity is 0, and will affect the body's trajectory during the following stance phase by controlling the tradeoff between apex height and forward velocity. The touchdown transition occurs when the foot makes contact with the ground. A prototypical trajectory for the SLIP hopper is shown in Figure 2. For the full dynamical equations, we refer the reader to [33], whose formulation we used for this work.

An important property of the SLIP model is that there is no closed form solution for the stance phase dynamics, requiring either an approximation or explicit forward integration in order to calculate the next state after applying a control input [34].
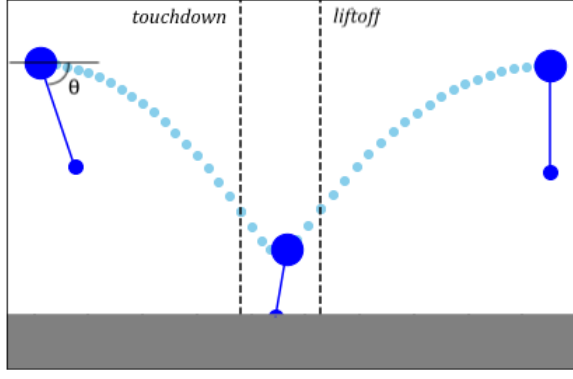
Fig. 2: A typical SLIP hopper trajectory

B. SLIP Failure Cases

Although the passive SLIP models are relatively simplistic, there are a number of failure cases that make planning for the SLIP hopper nontrivial over broken terrain. The first is the standard kinematic case where the body collides with the terrain, or the foot collides with a vertical surface (although this is technically a friction cone violation, we consider it a collision). Second, the leg angle can exit the friction cone during stance phase, causing the foot to slip. Third, if the hopper's apex height is too high above the terrain, the spring will fully compress during touchdown and will bottom-out, also causing a crash. Finally, because the energy of the passive SLIP model is fixed, there is an inverse relationship between the apex velocity and apex height. If the hopper travels too fast, the apex height will not be sufficient for the foot to swing to its position for the next step, causing the robot to "trip" on the ground.

The intersection of the kinematically feasible and dynamically reachable sets of footsteps is not immediately apparent just by analyzing the terrain. Therefore, when planning a trajectory, one must either precompute a set of feasible motion primitives for varying terrains, perform search by sweeping over control inputs and simulating the dynamics, or solve a nonlinear optimization problem.

V. BASELINE PLANNERS

A. Angle-space Planner

We formulate our first baseline planner as a best-first search over a tree that is dynamically constructed by sampling the possible actions at each leaf, adapted from the approach in [4]. Each node N is a tuple $(a, x, \theta, P, c, \{C_1...C_N\})$, where a is the apex state at which the leg angle θ is applied to reach x, the footprint of the CoM at the subsequent touchdown location. The other terms contain information about the graph structure. P is the parent of N, c is the cost of N, and $\{C_1...C_N\}$ is the list of the children of N.

Our search algorithm begins with a root node, which is the inevitable touchdown point from the initial apex. From that node, we sample all the available actions and

simulate the dynamics equations. If an action is feasible, a node corresponding to the current apex state and subsequent touchdown location is added to the tree as a leaf. Afterwards, the node with the lowest cost is fetched from a priority queue and expanded. This process repeats until a desired number of nodes exceed the goal distance. We refer to this planner as the "Angle-space" planner. Psuedocode for this algorithm is shown in Algorithm 1.

Algorithm 1: Angle-space Planner

Inputs : Initial apex state a, terrain model t, goal g, cost function h, forward model f, branching factor B, desired number of sequences d

Output: Up to d sequences of feasible touchdown locations

1 Set $N_{\text{root}} = (a, x_0)$;
2 Initialize empty priority queue q;
3 $N_s \leftarrow N_{root}$;
4 **while** *# successful solns $< d$ and q not empty* **do**
5     $a_s = N_s.a$;
6   Generate $\{\theta_1, ..., \theta_B\}$ within friction limits of t;
7   **for** $i = 1$ **to** B **do**
8   **if** *successful($f(a_s, t, \theta_i)$)* **then**
9           $a_{s+1}, x_{s+1} \leftarrow f(a_s, t, \theta_i)$;
10          $c \leftarrow h(x_{s+1})$;
11          $N_{s+1} \leftarrow (a_{s+1}, x_{s+1}, \theta_i, N_s, c, \{\})$;
12   Add N_{s+1} as a child of N_s;
13   Add N_{s+1} to q;
14   **end**
15    $N_s \leftarrow q.pop()$;
16   **if** *atGoal(N_s, g)* **then**
17   # successful solns++;
18   Add N_s to goal list;
19       $N_s \leftarrow q.pop()$;
20 **end**
21 **for** N *in goal list* **do**
22   Output all x on the path from N_{root} to N;
23 **end**

To evaluate the cost of each transition, we use a function that takes the child node as input. This cost function is formulated as

$$h(N_i) = w_1|x_i - x'| + w_2|\dot{x}_i| + w_3 D(N_{i-1}, N_i, N_{i+1}) \quad (1)$$

Where x_i is the node N_i's location and x' is the goal location. This cost function penalizes the distance of the node to the goal, high node apex velocity, and the node's "isolation". $D(N_{i-1}, N_i, N_{i+1})$ is a function calculating the spread between the nearest neighbors of N_i. If the N_i obtained by applying θ_i is far away from from its closest fellow child, that means a small deviation from θ_i is more likely to cause a crash in the presence of obstacles. (w_1, w_2, w_3) are the relative weights of these terms.
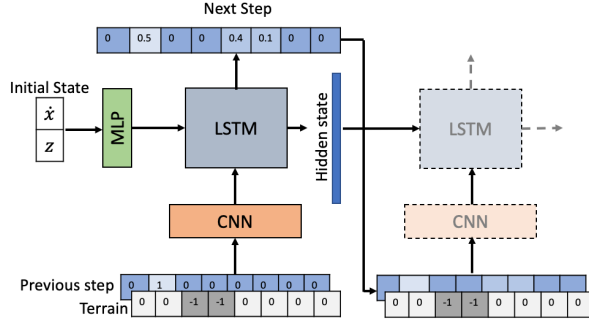
Fig. 3: A diagram of our LSTM framework unrolled over time

B. Low-level Controller

While the planner shown in Algorithm 1 produces the input angles necessary to hit its output sequence, it is inadvisable to use these in practice, since the open-loop angles will be incorrect as soon as there is any deviation from the reference sequence. Our approach is similar in spirit to [35]; we train a simple Multi-Layer Perceptron (MLP) in a supervised fashion to calculate the angle required to hit the desired step length given the current apex height.

Naturally, the learned controller does not perfectly achieve the desired step lengths. It can produce errors of up to 0.5m on our simulated robot, since small angle errors can cause large position errors at high velocities. This controller acts as a stand-in for the more complex controllers commonly used on higher dimensional legged robots.

C. Step-space Planner

One fault of the planner shown in Algorithm 1 is that it has no knowledge of the underlying controller and will output sequences that are physically achievable by the robot, but may cause significant errors if provided as inputs to the actual system consisting of the controller composed with the dynamics.

To incorporate the controller model, we alter the sampling procedure used in line 4 of Algorithm 1. Instead of directly sampling leg angles, we instead sample from a grid of possible step locations, pass those into the controller to obtain the corresponding θ, and use that in the planning algorithm. The branching factor for this variant corresponds to the size and spacing of the grid. The final sequence output by this modified planner uses the sampled step lengths x_L, as those feedforward inputs are known to result in feasible actions from the controller. This approach has the advantage that steps that encounter terrain obstacles can be avoided a priori, but requires a model for the low-level controller which is not always known in practice. We refer to this modification as the "Step-space planner".

VI. LEARNED RECURRENT PLANNER

Both of the search planners described in the previous section operate too slowly to be used online with frequent replanning. For a fast, online footstep planner, we instead use a learned sequential framework to generate plans that mimic the ones output by the search planners.

We use an LSTM [36], a variant of RNNs, as the core of our learned planner, with a Convolutional Neural Network (CNN) that jointly processes the terrain and previous step location to create an encoded vector as input to the LSTM.

Just like in sentence completion (for which RNNs are widely used), a large history of sequences over a wide variety of terrains and initial apexes induces a probability distribution over possible next step locations:

$$p(s_i|t, a, s_1, ..., s_{i-1}) \qquad (2)$$

where t is a discretized encoding of the perceived terrain, a is the initial apex prior to step s_1, and $s_1, ..., s_{i-1}$ are the past steps. This distribution can be used for planning either by sampling next nodes from the distribution as in [28], [37], or by taking the mode of the distribution as the next footfall location.

All together, our full LSTM-CNN model is governed by the following equations:

$$h_0 = g_0(a)$$
$$v_i = \text{CNN}([\hat{t}, \hat{s}_{i-1}])$$
$$\hat{s}_i, h_i = \text{LSTM}(h_{i-1}, v_i)$$

Where h_0 is the initial apex a projected onto the hidden state space of the LSTM by g_0, a single linear layer. v_i is the encoded input vector, calculated by the CNN by processing the previous step $\hat{s}_{i-1}$ and a discretized array representation of the terrain $\hat{t}$.

An important point is that $\hat{s}_{i-1}$ and $\hat{s}_i$ are discrete probability densities that share the same underlying support as the terrain heightmap $\hat{t}$, so finding the correct step location becomes a classification problem among the discrete cells. By encoding the step location in the terrain, the CNN can use co-located visual features to process both together as channels of a single image. A diagram of our model is shown in Figure 3.

The LSTM architecture allows the learned planner to capture the behavior of the hopper over time without requiring simulation of the forward dynamics, which is a key to handling the complex dynamics of the system without excessive computation. Meanwhile the CNN allows the planner to extract relevant terrain features and to marry that with an aligned representation of the planned footsteps.

A. LSTM-Guided Sampling

Mirroring the Step-space planner, we can generate an LSTM-Guided sampling planner where for each node expansion we instead sample from the output distribution of the LSTM. To generate these samples, we fetch the path from the root node to the current node, and pass the path into the LSTM as the past steps, along with the initial apex and terrain.

Sampling from the output of the LSTM allows us to incorporate controller knowledge into the LSTM's planning while also reducing the required branching factor of the

TABLE I: Training data generation parameters

Parameter	Value
# terrains	360
# initial states per terrain	8
Min # steps	8
# desired goal sequences per instance	8
Branching Factor	15
Coefficient of friction	0.6
w_0, w_1, w_2	1, 1, 2

TABLE II: Results on the Ditch Test Suite

Planner	Success % ↑	ODE Calls ↓	$t_{\mathbf{plan}}$ ↓
Heuristic	24	**0**	**2e-5s**
Angle-space	58.6	22661	0.15s
Step-space	**86.6**	30857	0.29s
LSTM (Ours)	56	**0**	0.007s
LSTM-Guided (Ours)	84	4885	0.04s

TABLE III: Results on the Step Test Suite

Planner	Success % ↑	ODE Calls ↓	$t_{\mathbf{plan}}$ ↓
Heuristic	56	**0**	**2e-5s**
Angle-space	60	27021	0.15s
Step-space	85	41146	0.34s
LSTM (Ours)	49	**0**	0.007s
LSTM-Guided (Ours)	**96**	6932	0.04s

planner by sampling from an informed distribution. We use temperature scaling [38] to widen the distribution and make it more suitable for sampling.

VII. EXPERIMENTS

We test our approach on the robot described in Section IV. For all experiments, we use a hopper whose body mass is 7kg, leg length is 0.5m, and spring constant is 3200 Nm^{-1}. For simulation, we use a custom simulation of the SLIP model with a timestep of 0.01 seconds. All of the sequence generation, model training, experiments, and simulations were done in Python on a machine with an AMD Ryzen 9 3600 processor with 32 GB of RAM and an NVidia RTX 2080 Super GPU.

Our LSTM model contains 2 layers each of 110 hidden units, and the input CNNs use 2 layers of 7x1 filters before being encoded into a 110-dimensional feature vector that acts as input to the LSTM, making our overall architecture relatively small. Because we frame planning as a classification problem, we use Cross-Entropy Loss to train our model. All models are implemented and trained in Pytorch.

A. Training

We generate training data for our LSTM by running the Angle-space planner with the parameters described in Table I. A single run of the Angle-space planner can give multiple training sequences as any path from the root node to a leaf is a feasible sequence of footsteps. In order to still provide reasonably good training sequences, we only add sequences that make a certain amount of progress towards the goal to the training set.

Each terrain is either a ditches or steps scenario. We model the terrain 8 meters in front of the robot and 3 meters behind, discretized in segments of 10cm. We also use a lower coefficient of friction during training in order to encourage further robustness by the LSTM. This process generates roughly 50,000 sequences of varying lengths (although a few of the sequences on the same terrain are usually highly similar to each other) and takes 1.5 hours with multithreading.

B. Testing

Our testing data consists of 15 random ditch-world and step-world terrains coupled with 5 different initial states swept over varying initial velocities (150 total test cases). The coefficient of friction used for all the test experiments is 0.8. Similar to training, the goal for the robot is to cross a certain x threshold, set to 10 meters.

We test 5 different planners: the Angle-space planner, Step-space planner, LSTM planner, LSTM-Guided sampling planner, and a terrain-heuristic planner. The terrain-heuristic planner is meant to mimic the approaches of [25], [24], where steps are selected based on a nominal gait dependent on the initial apex velocity, and then the planned steps are moved if they land in ditches. All planners are tested in a receding-horizon fashion, where the planned sequence is executed by the low-level controller for 3 steps before replanning occurs.

VIII. RESULTS

We measure a variety of metrics on both the ditch and step test suites. The first metric is the success rate. We show that, on its own, the LSTM planner is comparable to the Angle-space planner while using a fraction of the computation time and superior to the heuristic planner. In order to confirm our hypothesis that the LSTM-Guided planner can cut down on the planning time, we also measure the number of calls to the ordinary differential equation (ODE) integrator per successful episode, which is a measure proportional to the amount of computation required by each planner. Lastly, we show some qualitative cases and discuss how our planners fall short.

A. Quantitative Results

The success rate of all 5 planners on the ditch and step world scenarios are shown in Tables II and III. In these results, the Angle-space planner is used with an initial branching factor of 20 with a fallback branching factor of 30 if the first try fails. The Step-space planner is used with a horizon of 5m and spacing of 0.5 meters (branching factor of 10), and the LSTM-Guided planner is used with a branching factor of 3 and fallback factor of 5.

We can see that the Step-space and LSTM-Guided planners have the best overall performance, succeeding on over 80% of the terrains from both suites. This makes intuitive sense as they incorporate the controller's behavior into the planner. The pure LSTM planner performs slightly worse than the baseline Angle-space planner in both the ditches and steps scenarios; at low velocities, it tends to suggest steps that either cause the controller to violate the friction cone or that are very close to the edges of ditches and
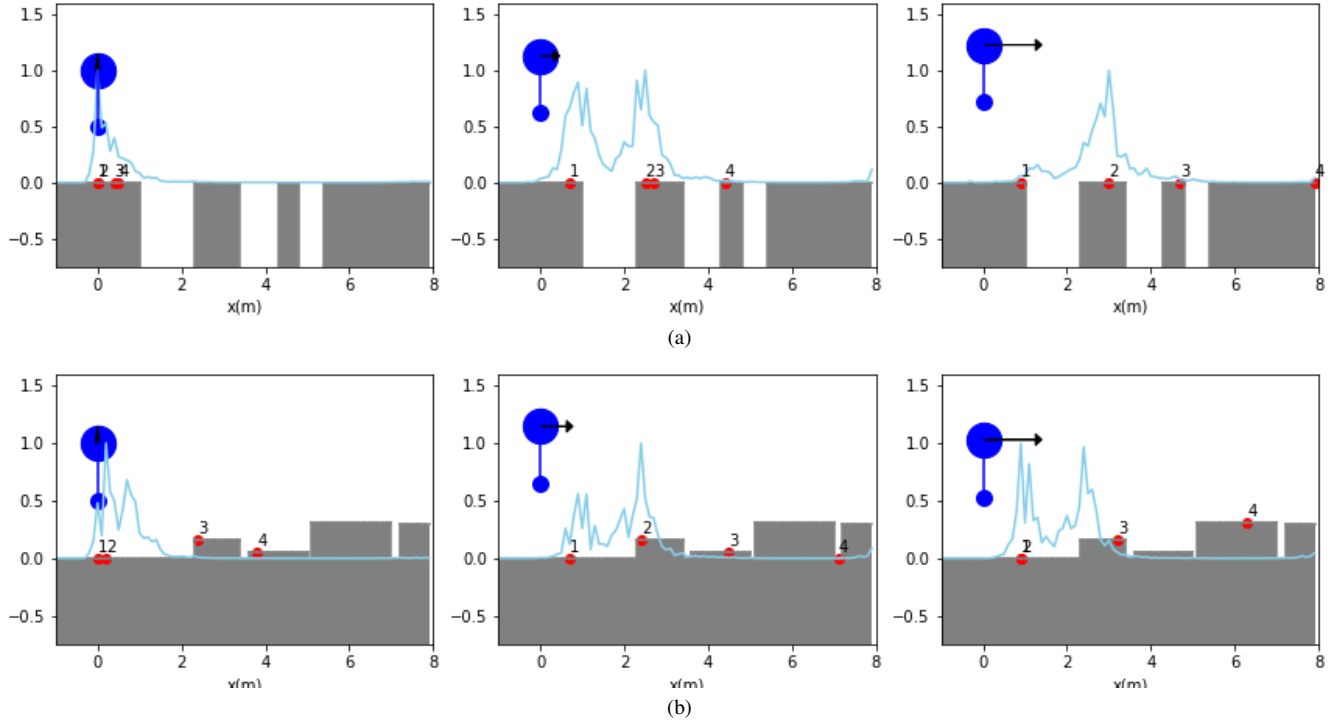
Fig. 4: Probability distributions over possible **second** step locations predicted by the LSTM. Increasingly long arrows indicate higher initial energy. Distributions are plotted such that the mode has height 1.0.

steps. However, the high performance of the LSTM-Guided sampler shows that the LSTM still suggests a high-quality probability distribution over possible locations. The heuristic planner performs well on the steps terrain, but performs poorly on the ditch scenarios. Its performance is highly sensitive to its parameters: the scaling factor with the initial velocity, and how far a step is moved if it lands in a no-go area.

The distributions conditioned on terrain and state allow the LSTM-Guided planner to explore fewer nodes. Tables II and III show that the LSTM-Guided planner achieves extremely high success rates while making fewer than 1/6 the number of ODE calls than the Step-space planner, and fewer than 1/4 of the Angle-space planner.

B. Qualitative Analysis

4 shows a few distributions and plans generated by our learned planner. 4-step plans and the probability distributions for the second step generated by the LSTM over 3 different initial apex states. As we increase both the apex height and initial velocity, the LSTM alters its plan accordingly to traverse the terrain. On the ditch terrain, the peaks of the distribution for the second step move further away as the initial energy increases. On the step terrain, when starting with too high an initial velocity it shortens its first two steps due to the tradeoff mentioned in Section IV-B. In the 3rd subfigure in Figure 4b, with too high an initial velocity, it will not be able to generate the necessary height to clear the obstacles, so it takes a short step to slow down.

The LSTM planner does have a number of shortcomings. As with many learned planners, there is no way to tune it to change the output behavior without retraining the model. Although it takes noticeably shorter steps when the hopper has lower energy, it still tends to propose more aggressive plans than the Angle-space planner.

In addition, many of the pathological behaviors present in the training data also appear in the learned sequences, including stepping very close to edges, as seen in Figure 4b. In Figure 4a there are spikes in the probability distribution around the edges of the islands, corresponding to training sequences where the Angle-space planner selects the node that minimizes the distance to the goal.

IX. CONCLUSION AND FUTURE WORK

In this work, we demonstrated how we can use an LSTM model to learn a distribution for likely footsteps conditioned both on terrain geometry and robot dynamics, and that this model can be successfully used to quickly generate footstep plans and to speed up a sampling-based planner. These promising results also set up a great deal of future work. We have explored adapting our approach to a 3D SLIP model with 2D heightmaps, and preliminary results show that with a few adjustments, the LSTM can match the Angle-space planner in full 3D. Further work includes exploring generalizing to more complicated robots, generating maximally informative terrains and apexes to minimize the amount of training data, and implementing our framework on real hardware.

References

[1] 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," *The International Journal of Robotics Research*, vol. 30, no. 2, pp. 175–191, 2011.

[2] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-Based Locomotion Planning, Estimation, and Control Design for the Atlas Humanoid Robot," *Auton. Robots*, vol. 40, no. 3, p. 429–455, Mar. 2016.

[3] M. Kalakrishnan, J. Buchli, P. Pastor, and S. Schaal, "Learning locomotion over rough terrain using terrain templates," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 167–172.

[4] G. Zeglin and B. Brown, "Control of a bow leg hopping robot," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation*, vol. 1, May 1998, pp. 793–798 vol.1.

[5] M. A. Arain, I. Havoutis, C. Semini, J. Buchli, and D. G. Caldwell, "A comparison of search-based planners for a legged robot," in *9th International Workshop on Robot Motion and Control*, Jul. 2013, pp. 104–109.

[6] A. W. Winkler, C. Mastalli, I. Havoutis, M. Focchi, D. G. Caldwell, and C. Semini, "Planning and execution of dynamic whole-body locomotion for a hydraulic quadruped on challenging terrain," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 5148–5154.

[7] M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal, "Fast, robust quadruped locomotion over challenging terrain," in *2010 IEEE International Conference on Robotics and Automation*, May 2010, pp. 2665–2670.

[8] M. Wermelinger, P. Fankhauser, R. Diethelm, P. Krüsi, R. Siegwart, and M. Hutter, "Navigation planning for legged robots in challenging terrain," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2016, pp. 1184–1189.

[9] A. Hornung, A. Dornbush, M. Likhachev, and M. Bennewitz, "Anytime search-based footstep planning with suboptimality bounds," in *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, 2012, pp. 674–679.

[10] P. Vernaza, M. Likhachev, S. Bhattacharya, S. Chitta, A. Kushleyev, and D. D. Lee, "Search-based planning for a legged robot over rough terrain," in *2009 IEEE International Conference on Robotics and Automation*, May 2009, pp. 2380–2387, iSSN: 1050-4729.

[11] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and Trajectory Optimization for Legged Systems Through Phase-Based End-Effector Parameterization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1560–1567, Jul. 2018.

[12] R. Deits and R. Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization," in *2014 IEEE-RAS International Conference on Humanoid Robots*, 2014, pp. 279–286.

[13] B. Ponton, A. Herzog, S. Schaal, and L. Righetti, "A convex model of humanoid momentum dynamics for multi-contact motion generation," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 2016, pp. 842–849.

[14] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," 2013.

[15] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, "Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning," 2020.

[16] X. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Trans. Graph.*, vol. 36, no. 4, Jul. 2017. [Online]. Available: https://doi.org/10.1145/3072959.3073602

[17] S. Gangapurwala, M. Geisert, R. Orsolino, M. Fallon, and I. Havoutis, "Rloc: Terrain-aware legged locomotion using reinforcement learning and optimal control," 2020.

[18] A. Meduri, M. Khadiv, and L. Righetti, "Deepq stepper: A framework for reactive dynamic walking on uneven terrain," 2020.

[19] X. Peng, G. Berseth, and M. van de Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," *ACM Trans. Graph.*, vol. 35, no. 4, Jul. 2016. [Online]. Available: https://doi.org/10.1145/2897824.2925881

[20] O. Melon, R. Orsolino, D. Surovik, M. Geisert, I. Havoutis, and M. Fallon, "Receding-horizon perceptive trajectory optimization for dynamic legged locomotion with learned initialization," 2021.

[21] R. Deits, T. Koolen, and R. Tedrake, "Lvis: Learning from value function intervals for contact-aware robot controllers," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 7762–7768.

[22] B. Yang, L. Wellhausen, T. Miki, M. Liu, and M. Hutter, "Real-time optimal navigation planning using learned motion costs," 2021.

[23] Y. Lin, B. Ponton, L. Righetti, and D. Berenson, "Efficient humanoid contact planning using learned centroidal dynamics prediction," 2019.

[24] O. A. V. Magaña, 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," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2140–2147, Apr. 2019.

[25] D. Kim, D. Carballo, J. D. Carlo, B. Katz, G. Bledt, B. Lim, and S. Kim, "Vision Aided Dynamic Exploration of Unstructured Terrain with a Small-Scale Quadruped Robot," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 2464–2470.

[26] O. Melon, M. Geisert, D. Surovik, I. Havoutis, and M. Fallon, "Reliable trajectories for dynamic quadrupeds using analytical costs and learned initializations," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 1410–1416.

[27] J. Siekmann, K. Green, J. Warila, A. Fern, and J. Hurst, "Blind Bipedal Stair Traversal via Sim-to-Real Reinforcement Learning," in *Proceedings of Robotics: Science and Systems*, Virtual, July 2021.

[28] Y. Kuo, A. Barbu, and B. Katz, "Deep sequential models for sampling-based planning," *arXiv:1810.00804 [cs]*, Oct. 2018.

[29] M. J. Bency, A. H. Qureshi, and M. C. Yip, "Neural Path Planning: Fixed Time, Near-Optimal Path Generation via Oracle Imitation," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov. 2019, pp. 3965–3972.

[30] W. J. Schwind, "Spring loaded inverted pendulum running: A plant model."

[31] R. J. Full and D. E. Koditschek, "Templates and anchors: neuromechanical hypotheses of legged locomotion on land," *Journal of Experimental Biology*, vol. 202, no. 23, pp. 3325–3332, Dec. 1999, publisher: The Company of Biologists Ltd Section: Journal Articles.

[32] T. Apgar, P. Clary, K. Green, A. Fern, and J. Hurst, "Fast online trajectory optimization for the bipedal robot cassie," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[33] S. G. Carver, N. J. Cowan, and J. M. Guckenheimer, "Lateral stability of the spring-mass hopper suggests a two-step control strategy for running," *Chaos (Woodbury, N.Y.)*, vol. 19, no. 2, Jun. 2009.

[34] G. Piovan and K. Byl, "Enforced symmetry of the stance phase for the Spring-Loaded Inverted Pendulum," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 1908–1914.

[35] J. K. Yim, "Precision Jumping Limits from Flight-phase Control in Salto-1P." pp. 2229–2236, 2018.

[36] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[37] A. H. Qureshi and M. C. Yip, "Deeply Informed Neural Sampling for Robot Motion Planning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 6582–6588.

[38] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On Calibration of Modern Neural Networks," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, 2017, p. 1321–1330.