Hierarchical Reinforcement Learning with Trajectory Optimization for Unitree Go2 on Extreme Terrain

Anand Majmudar Faraz Rahman Ian Pedroza

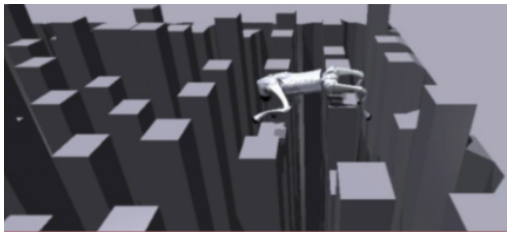

Fig. 1. Extreme Column Terrain

***Abstract*—We use a trajectory optimizer as a high-level planner for footstep positions to achieve a given end goal position. We train a reinforcement learning model with Proximal Policy Optimization as a low-level actor to control quadruped Unitree Go2's actuators to achieve those target footstep positions. We then train and test this hierarchical system on extreme terrain including separated pillars for the quadruped to cross.**

I. Introduction

A. Setting

Our goal is to run a mobile policy on the Unitree Go2 to allow it to pass through extreme terrain, especially this spaced multi-column environment (see Fig 1.) which necessitates well-planned foot placement with little room for error. The Unitree Go2 is a 12-actuator (3 per leg) affordable quadruped robot which also has access to lidar and is often used for agile locomotion tasks.

B. Motivation

The baseline, Proximal Policy Optimization incentivized to have positive forward velocity, maintain vertical position, and obey torque and actuator positional constrains, fails to traverse the terrain and most importantly has an unnatural gait. We aim to instead use a traditional Trajectory Optimizer as a higher-level planner for footsteps and a lower-level reinforcement learning policy to achieve those desired positions with a defined steady gait.

II. Related Work

Much of the inspiration for our project comes from the Deep Tracking Control paper [1] from RSL at ETH Zurich. The paper builds off of previous work in online quadrupedal Trajectory Optimization. Specifically, the TO model is based on the paper Terrain Aware Optimization for Legged Systems (TAMOLS) [2], another project out of RSL.

TAMOLS [2] is a perceptive motion planning algorithm that uses a local height map to better plan foot steps to achieve a desired reference trajectory. In a sim2real implementation, this height map may be generated from a LiDAR or learned from live-cameras.

We also use Extreme Parkour [3] to bootstrap our Reinforcement Learning environment setup, infrastructure, and reward shaping on extreme terrain.

III. Methods

A. Trajectory Optimizer

We built our trajectory optimization algorithm PyDrake using the SNOPT solver as closely as possible to the TAMOLS reference design. In order to meet solving time requirements we relaxed the problem setup by only requiring that the Go2 move forward (i.e. not turn). Fig 2 is a visualization of the motion plan generated based on the height map of stairs terrain.

1) TAMOLS overview: Our implementation of TAMOLS solves for the four next optimal footholds (one for each leg) as well as a polynomial spline that parametrizes a trajectory for the quadruped's center of mass the xyz-coordinates and ZYX-Euler angles.

The foot holds are given by

$$\mathbf{p}_i, \quad i \in \{0, 1, 2, 3\}$$

The base pose is given by

$$\mathbf{\Pi}_B = [\mathbf{p}_B \ \phi_B]$$

where $\mathbf{p}_b$ represents the XYZ coordinates and ϕ_B represents the ZYX Euler angles. The splines are split into k segments called phases. For the kth phase, the lth dimension of the base pose trajectory is parametrized as:

$$\mathbf{\Pi}_{B,kl}(t) = a_{0kl} + a_{1kl}t + \cdots + a_{4kl}t^4$$

where $t \in (0, \tau)$ and τ is the length of each spline, $k \in [0, N_s - 1]$ indexes the spline segment and N_s is the total number of phases, and $a_{0kl}, \ldots, a_{4kl}$ are the coefficients of the polynomial. This notation is consistent with the original paper and will give us language to discuss the modifications we made to help our implementation. However more in depth discussion on the derivation of the planner can be found in the original paper [2]. We will limit our discussion to modifications we made to the algorithm to simplify implementation.

2) Our modifications to TAMOLS: The original paper uses a sequence of quadratic program (SQP) to solve the constrained non-linear program (NLP). Each iteration of the SQP used the Guass-Newton method to approximate the objective with a convex surrogate. They then use a custom solver to efficiently find local optima. Without the flexibility of custom tooling we choose to modify to the original problem to ease the optimization in SNOPT and PyDrake. Specifically, we

- Switch the gait from a trot (i.e. diagonal feet move in sync) to a walk (i.e. each leg moves one at a time)
- Assume reference trajectory moves forward at some constant velocity
- Use a manual warm-start of initial foot positions instead of using a smoothed height map to solve a more relaxed problem as a warm start to the full optimization.

The walking gait allowed three legs make contact with the ground at all times. This assumption allowed unify the dynamics constraints into (17a) and (17b) from [2], and let us ignore (17c, d). Originally the dynamics constraints would alternate between (17b) and (17c, d) depending on whether 3 or more legs were in contact with the ground for the respective phase.

$$\mu \mathbf{e}_z^T \cdot \mathbf{a}_B \geq \|(\mathbf{I}_{3\times 3} - \mathbf{e}_z \mathbf{e}_z^T) \cdot \mathbf{a}_B\| \quad N > 0 \qquad (17a)$$

$$m \det(\mathbf{p}_{ij}, \mathbf{p}_B - \mathbf{p}_i, \mathbf{a}_B) \leq \mathbf{p}_{ij}^T \cdot \dot{\mathbf{L}}_B \quad N \geq 3 \qquad (17b)$$

$$m \det(\mathbf{p}_{ij}, \mathbf{p}_B - \mathbf{p}_i, \mathbf{a}_B) = \mathbf{p}_{ij}^T \cdot \dot{\mathbf{L}}_B \quad N = 2 \qquad (17c)$$

$$\det(\mathbf{e}_z, \mathbf{p}_{ij}, \mathbf{M}_i) \geq 0 \quad N = 2 \qquad (17d)$$

Here N is the number of legs in contact with the ground, μ is a friction coefficient, $\mathbf{p}_{ij} = \mathbf{p}_j - \mathbf{p}_i$, $\mathbf{M}_i$ is the moment about foothold i (derived in the [2]) and $\mathbf{a}_B$ is an acceleration vector. Note that removing (17c) eliminates the only equality constraint in the dynamics model. More discussion on how this may have benefited the optimization will be in the following section.

The assumption of a forward velocity reference trajectory means that the angular momentum will remain near zero along the motion plan. Once again we see this simplify our dynamics since it zeroes out the RHS of (17b).

$$m \det(\mathbf{p}_{ij}, \mathbf{p}_B - \mathbf{p}_i, \mathbf{a}_B) \leq 0 \quad N \geq 3 \qquad (17b)$$

The forward assumption also allows us to manually warm-start the optimization based on the given reference forward velocity. The original formulation used a smoothed height-map to do this. The smoothed height map provided a smoother optimization landscape allowing to the solver to find a neighborhood near the global optimal before solving further. An example objective which used this graduated technique was the following objective.

$$\min \nabla h_{s1}(\mathbf{p}_i)^T \nabla h_{s1}(\mathbf{p}_i).$$

Which encouraged footholds to be on flatter terrain. Note that $h(\cdot)$ is a function from xy-coordinates to a height and that $h_{s1}(\cdot)$ here is a smoothed version.

3) Implementation: The optimizer was built using PyDrake and SNOPT. The source code for the optimization itself can be found at https://github.com/farazsrahman/fetch. We use Plotly to make the custom visualizations to debug the TO.

B. Actor-Critic Proximal Policy Optimization

For the actual simulation of the Go2 and terrain, we use NVIDIA Isaacgym which can parallelize and accelerate simulation rollout on NVIDIA GPUs (thus we ran this on a 2-GPU cluster).

For the RL portion, we use Actor-Critic architecture and train with Proximal Policy Optimization.

Actor-Critic architecture has a Critic network, which is a network approximating the value function (given state-action pairs, output rewards-to-go), and an Actor Network, which actually makes actuator torque decisions which impact the next state of the environment. Both are implemented as MLPs of linear layers and Exponential Linear Unit activations (a smoother and sometimes-negative ReLU).

Both share the same observation space, which for the baseline consists of the Unitree Go2's joint linear and angular velocities, actions, and the 14 by 14 heightmap of terrain close to the quadruped.

Proximal Policy Optimization uses a clipped objective function to prevent too-large updates to the actor or critic networks in a single step, leading to more stable learning over a longer training time. It also simultaneously optimizes both the actor and critic networks for sample efficiency and to ensure the most current value estimates are used for each actor update.

We use a learning rate of $1e-3$ along with the other default RSL RL PPO hyperparameters.

We reward the network for proximity to the current next set of goal footsteps (in a diagonal gait pattern: first reward one diagonal pair of legs, then when that pair reaches the targets, we switch to rewarding the other pair). We also reward it for z-stability in position and velocity (maintaining the same height), positive forward velocity, increasing forward position, using less torque, maintaining a flat orientation, staying within actuator positional limits, and reducing xy angular velocity.

The final run was 2000 episodes of training parallelized across 200 environments.

IV. Results

A. Baseline RL

The base RL policy was able to traverse a few columns but fell shortly after due to the lack of proper gait and its inability to select perceptively select footholds. These shortcomings reflected the issues mentioned in the paper in that RL is increasingly data-inefficient as rewards become sparse [1]. In this case the large gaps between columns are what made the set of valid (reward-able) footholds sparse.

B. Trajectory Optimizer

We tested the trajectory optimizer on a number of terrains and were able to consistently see reasonable solutions solutions. Figures 2 and 3 display two examples of such terrains.

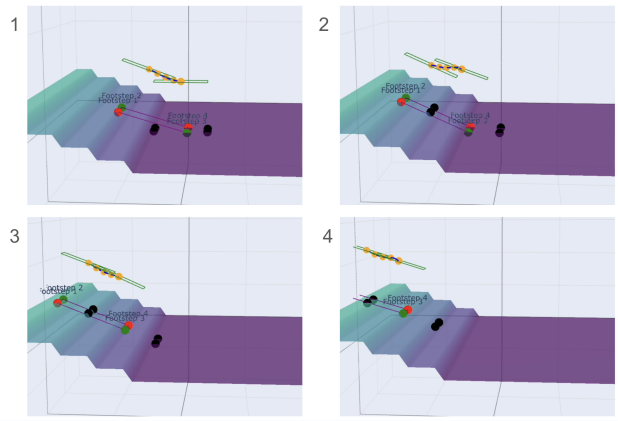

Fig. 2. Planner going up the stairs

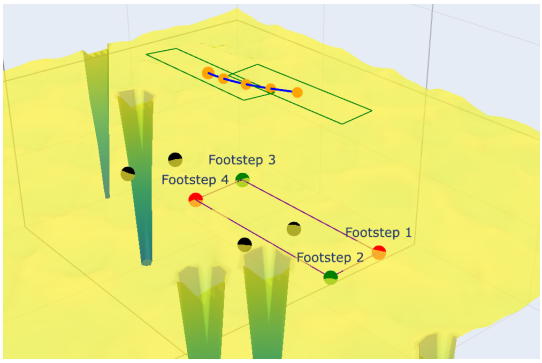

Fig. 3. Planner avoiding hole

The first displays an iterative call to the planner picking footsteps and center of mass trajectory up a flight of stairs. The second shows how the planner picks trajectories that are away from holes or sloped parts of terrains. Time to solver convergence often took 1-2 seconds, however the problem setup and auto-differentiation in PyDrake added an additional 10-15 seconds depending on which costs and constraints were enabled.

C. Hierarchical model

We were unable to recover the performance of the base-line policy on the hierarchical model, due to issues in training the the two systems end-to-end. The rest of this section will be dedicated to a reflection on what worked, what did not work and what we would do in the future to address the issue.

1) What worked: We were able to achieve one full training run end-to-end. However the resulting policy did not learn a proper gait and could not walk forward. The primary issue we identified was that the feet shot forward towards the the planned footstep. We suspect that this happened because the simple reward of smaller squared distance to the desired foothold dominated the rewards for moving forward. A solution would likely be in the form of a better engineered reward function.

2) What did not work: We categorize takeaways into the following two sections

- **Optimization speed, online learning, and the height map:** When designing the RL curriculum we assumed that the solver would be fast enough to provide solutions on line when queried by the training loop. This assumption did not hold and the solver (which often took 1-2 seconds) slowed down the training process. One item to note is that the solving time was significantly impacted by cost functions that leveraged the height map. This was not surprising because PyDrake had no symbolic expression that native-ly accepted the discrete height map data-structure. The only way to implement the height map without a custom PyDrake module was by adding a cost for every entry of the height map and every foot that was conditioned on whether the corresponding foot was located at the respective xy position.

  A solution to speeding up the optimization would likely require implementing the optimization problem in directly with PySNOPT or using the Gauss-Newton approximation directly using a vector library in a faster language such as Julia or C++.

  Another, less engineering intensive solution that would still suffice for experimental purposes would have been moving the trajectory optimization offline. A curriculum could have sampled a series of initializations for the quadruped, run the optimization algorithm over each of these initializations and then used this fix set of trajectory plans and initializations to formualte training the policy into a semi-supervised learning problem. The resulting policy would be less result to out of distribution initializations, but would serve as a good test for the rest of our setup in the absense of a faster planner.
- **Reward Function Tuning and Debugging:** As mentioned in the previous section the reward for moving the foot towards the planned location dominated the base rewards resulting in a policy that simply shot its end-effector towards the target. A simple change that would make the reward more sparse would be to only punish the distance from the desired foothold when the foot is in contact with the ground. This would also encourage lifting the foot when the foot is not near the reward.

  One item to note here is that we struggled to properly debug our model since Isaac Gym provides limited rendering support when working remotely. Without visualization of the training procedure reward function engineering can become a challenging art. We believe that newer tools such as Isaac Lab, or a non-remote server setup could have provided a better debugging experience.

  Lastly, assuming we designed better reward function, it is likely that we would have also needed to hyper-parameter tune the reward weights which we were unable to do since the training speed bottle necked the number times we could try different curricula.

V. Future Work

Future work for this project would involve reward tuning further and adjusting the RL-TO infrastructure so the quadruped is encouraged to meet the target foot positions not just by moving its legs straight to those positions, but by stepping according to the predefined natural gait. This was expanded upon in the previous section. Further down the line, it would be nice to see this policy deployed on the real Unitree Go2.

Another interesting experiment would be to drop the planner at inference time and just use the policy. The planner would still be used to train the model, but the information it provides would be distilled in to the RL policy during training to make the learning more sample-efficient.

References

[1] Jenelten et al, Deep Tracking Control: https://arxiv.org/abs/2309.15462
[2] Jenelten et al, TAMOLS https://arxiv.org/abs/2206.14049
[3] Cheng et al, Extreme Parkour https://arxiv.org/pdf/2309.14341