Navigation Using Divergence Constrained Model-Based Reinforcement Learning

Aditya Singh Fazil Khan Sukriti Gupta

I. Introduction

Wheeled robots struggle to navigate rough, unstructured terrain reliably. On flat ground, dynamics are predictable. Rough terrain, however, introduces high variance in geometry, interaction forces, and wheel slip. These effects make the system's dynamics highly divergent, meaning that even small modeling errors amplify rapidly when propagated over time. As a result, long-horizon prediction becomes unreliable, forcing robots to rely on short-sighted or conservative actions. This limits strategic decision-making and safe traversability. Improving reliability in these settings is crucial for applications such as environmental monitoring, search-and-rescue, planetary exploration, and autonomous off-road navigation.

In this work, we address the problem of long-horizon trajectory planning on rough terrain under significant model uncertainty. We aim to generate dynamically feasible, non-myopic trajectories robust to accumulated prediction error. We focus on the setting where the robot has access to onboard terrain perception, but the true dynamics remain difficult to model due to complex robot–terrain interactions. Our goal is not merely to fit a predictive model, but to ensure that the model can be safely and effectively used for planning in environments where prediction errors naturally grow with horizon length.

Existing approaches each face critical limitations in this context. Handcrafted or physics-based models often break down when their assumptions fail on irregular terrain, leading to poor long-term predictions. Model-free reinforcement learning can learn robust policies, but typically requires large amounts of real-world data and often fails to generalize to unseen terrain. Model-based reinforcement learning offers a promising compromise leveraging learned dynamics for sample efficiency, yet most MBRL methods suffer from a key failure mode: during optimization, they exploit inaccuracies in the learned model. This leads to trajectories that appear optimal under the model but are infeasible or unsafe when executed, a problem that becomes especially severe in divergent rough-terrain dynamics.

To address this challenge, we build on the divergence-constrained MBRL framework introduced by Wang et al. and adapt its core ideas to our robot and environment. Our approach combines 1) uncertainty-aware probabilistic dynamics learning, 2) closed-loop trajectory prediction that accounts for the stabilizing action of a tracking controller, and 3) a divergence-constrained trajectory optimizer that prevents the planner from selecting trajectories the model cannot reliably predict. By constraining the allowable deviation between nominal and closed-loop predictions, the planner avoids exploiting model errors and instead selects long-horizon maneuvers that are both strategically advantageous and physically executable.

Our contribution is a practical and robust planning framework for rough-terrain navigation that integrates uncertainty-aware dynamics learning, closed-loop prediction, and divergence-constrained optimization into a unified MBRL pipeline. Through simulation experiments, we show that this approach improves prediction stability, reduces model-exploitation failures, and enables more reliable non-myopic navigation in challenging unstructured environments.

II. Related Works

Model-based reinforcement learning (MBRL) has emerged as a data-efficient alternative to model-free methods for robotic control. Probabilistic Ensembles with Trajectory Sampling [1] demonstrates that probabilistic ensembles and trajectory sampling can capture epistemic uncertainty for long-horizon planning. Subsequent work [2] expanded uncertainty-aware MBRL through risk-sensitive objectives and uncertainty penalties, enabling safer behavior in navigation and offline RL settings. Yet, these approaches are prone to model exploitation: the optimizer eventually discovers and abuses inaccuracies in the learned dynamics.

Rough-terrain navigation is traditionally addressed using predictive models and geometric reasoning. [3] used kinematic and dynamic mobility models to reason about terrain, while later traversability methods [4] built costmaps based on local slope, roughness, and semantic cues. Although effective for short-horizon planning, these methods lack the ability to account for the divergent dynamics and long-horizon uncertainty present in unstructured environments.

A key challenge in learned dynamics models is multi-step error accumulation. [5] shows that single-step likelihood training induces covariate shift and destabilizes long rollouts. This motivates multistep training objectives and techniques that explicitly model uncertainty propagation. Variational dropout [6] and Bayesian approximations have further provided practical tools for capturing epistemic uncertainty in neural dynamics models.

To prevent planners from exploiting imperfect models, several works have introduced uncertainty penalties or risk-sensitive costs. However, such penalties often trade off task performance with excessive conservatism. Convergent planning as proposed in [7] demonstrated that constraining divergence between nominal and closed-loop predictions helps avoid unrealistic trajectories while preserving long-horizon reasoning.

Finally, the role of tracking controllers in multi-step prediction has been highlighted in work on closed-loop trajectory optimization and feedback-aware rollouts. PD and LQR-based controllers in [8] helped stabilize prediction by modeling corrective actions that occur during execution and reducing prediction variance. Heightmap-conditioned models in off-road navigation further demonstrate the benefits of coupling perception with closed-loop dynamics prediction.

Our work integrates probabilistic dynamics learning, multistep uncertainty propagation, closed-loop prediction, and divergence-constrained trajectory optimization into a unified framework for reliable long-horizon planning on rough terrains.

III. Problem Formulation

We consider a robot in a continuous state space with onboard perception and a learned dynamics model. Given the robot's current state and a goal position, our objective is to compute a long-horizon action sequence that drives the system toward the goal while respecting the inherent uncertainty of rough-terrain dynamics. Unlike standard trajectory optimization, the problem must explicitly account for the rapid growth of uncertainty and the stabilizing effect of closed-loop feedback during execution.

A. State, Action, and Terrain Representation

At time t, the robot maintains a 9-dimensional internal state:

$$s_t = \begin{bmatrix} g_t \\ v_t \\ \omega_t \end{bmatrix} \in \mathbb{R}^9,$$

where g_t is the body-frame gravity vector, v_t the linear velocity, and ω_t the angular velocity.

The robot's pose is:

$$x_t = (X_t, Y_t, Z_t, \psi_t),$$

where ψ_t is the yaw.

Terrain is encoded by a robot-centric heightmap patch:

$$h_t \in \mathbb{R}^{64\times 64},$$

extracted from the global elevation map.

The control input at each step consists of throttle and steering angle and is represented as:

$$a_t = (v_t^{\mathrm{cmd}}, \omega_t^{\mathrm{cmd}}) \in \mathbb{R}^2.$$

The system evolves according to an unknown stochastic transition function:

$$(s_{t+1}, x_{t+1}) \sim P(s_{t+1}, x_{t+1} \mid s_t, x_t, a_t, h_t). \tag{1}$$

B. Learned Probabilistic Neural Dynamics Model

We approximate the transition dynamics using a neural network parameterized by θ. Given the current heightmap m_t, state s_t, and action a_t, the model predicts a Gaussian distribution over the next–state change:

The supervised learning target is the 13-dimensional state transition vector:

$$\Delta\xi_t = \begin{bmatrix} \Delta X_t \\ \Delta Y_t \\ \Delta Z_t \\ \Delta\psi_t \\ s_{t+1} \end{bmatrix} \in \mathbb{R}^{13}.$$

We need to model how uncertainty propagates over time, So instead of predicting a single next step, we predict a K-step trajectory into the future. Instead of predicting just one future path, we need to model Epistemic Uncertainty, which is how unsure the model itself is about the physics of the world. To do this, we roll out N different particles. For each particle, we sample a different dropout mask using Variational Dropout.

$$\hat{s}_{t+1}^{(i)} \sim p_\theta\Big(s_{t+1} \mid \hat{s}_t^{(i)}, a_t\Big).$$

Next, we calculate the loss for each particle using the Gaussian Negative Log-Likelihood. This models the Aleatoric Uncertainty, the inherent noise in the environment (like wheel slippage).

$$\mathcal{L} = \frac{1}{K}\sum_{t=1}^{K}\left[\frac{1}{2}\left(\frac{(\Delta\xi_t - \mu_t)^2}{\sigma_t^2} + \log\sigma_t^2\right)\right], \tag{3}$$

where (μ_t, σ_t^2) are produced by the CNN–LSTM model.

Finally, we combine the losses from all N particles using the Log-Sum-Exp trick, which provides numerical stability and properly aggregates the probabilities.

$$L_{\mathrm{ms}} = L_{\min} - \log\left(\frac{1}{N}\sum_{i=1}^{N}\exp\Big(L_{\min} - L^{(i)}\Big)\right),$$

C. Nominal Trajectory Prediction

Given an initial state s_0 and candidate open-loop action sequence $\bar{a}_{0:T-1}$, the nominal trajectory

$$\bar{d} = \{\bar{s}_0, \bar{s}_1, \ldots, \bar{s}_T\}$$

is generated by rolling out the mean model:

$$\bar{\xi}_{t+1} = \mu_\theta(\bar{s}_t, \bar{a}_t, h_t). \tag{4}$$

Pose integration:

$$\begin{aligned} \bar{X}_{t+1} &= \bar{X}_t + \Delta X_t, \\ \bar{Y}_{t+1} &= \bar{Y}_t + \Delta Y_t, \\ \bar{Z}_{t+1} &= \bar{Z}_t + \Delta Z_t, \\ \bar{\psi}_{t+1} &= \bar{\psi}_t + \Delta \psi_t, \end{aligned} \tag{5}$$

D. Cost Function

The planning objective is to minimize terminal position error:

$$C(\bar{d}) = \left\| \begin{bmatrix} \bar{X}_T \\ \bar{Y}_T \end{bmatrix} - \begin{bmatrix} X_{\text{goal}} \\ Y_{\text{goal}} \end{bmatrix} \right\|^2. \tag{8}$$

E. Divergence Measure Between Nominal and Closed-Loop Predictions

To prevent selecting trajectories the model cannot reliably predict, we measure mismatch:

$$u(\bar{d}, \hat{D}) = \max_{t \in \{1,\ldots,T\}} \frac{1}{I} \sum_{i=1}^{I} \left\| \hat{x}_t^{(i)} - \bar{x}_t \right\|^2.$$

We use the max divergence across steps.

F. Divergence-Constrained Trajectory Optimization

The planning problem is:

$$\min_{\bar{a}_{0:T-1}} \quad C(\bar{d}) \qquad \text{s.t.} \qquad u(\bar{d}, \hat{D}) \le U. \tag{10}$$

We solve it using an augmented Lagrangian:

$$L = C(\bar{d}) + \lambda \, \max\big(0, \; u(\bar{d}, \hat{D}) - U\big),$$

with multiplier λ increased iteratively.

IV. Simulation Setup

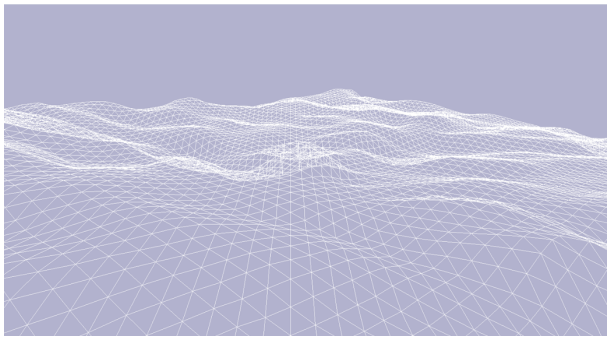

Fig. 1. Heightmap

To evaluate our approach under realistic rough-terrain conditions, we built a custom simulation environment using the Jezero Crater Digital Terrain Model (DTM) from NASA/UA HiRISE. The GeoTIFF elevation map was converted into a 128×128 heightmap using rasterio, providing a 1meter-per-pixel resolution that preserves the natural undulations and slopes in the planetary landscape. This heightmap was then loaded into PyBullet as a 3D terrain, scaled, and textured using a Mars-inspired surface texture for visual realism. To ensure accurate physical interaction, we applied height normalization, optional vertical exaggeration for low-variance terrain, and height-dependent Z-offset correction before generating the collision shape. This produced a high-fidelity rough-terrain surface suitable for long-horizon navigation testing.

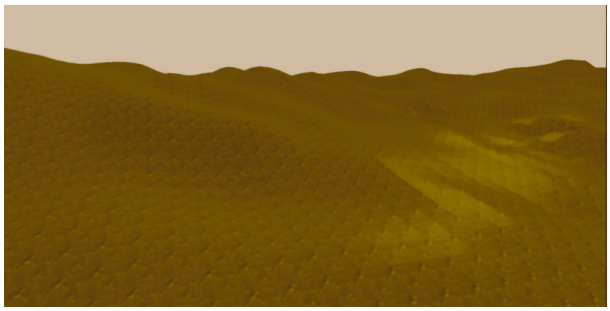

Fig. 2. Simulation Environment

On top of the terrain, we deployed a simulated Husky ground robot. The robot is spawned at randomized positions near the center of the map and is allowed to physically settle before any control is applied, ensuring stable contact initialization. Wheel joints are identified at runtime, assigned realistic friction coefficients, and controlled through a simple skid-steer velocity model. The environment provides full 6-DoF base state information at each timestep, including body-frame gravity, linear and angular velocities, and local terrain geometry. For terrain perception, we extract a robot-centric heightmap patch of configurable size centered on the robot's current position, enabling the learned dynamics model to incorporate local slope information while remaining agnostic to global coordinates.

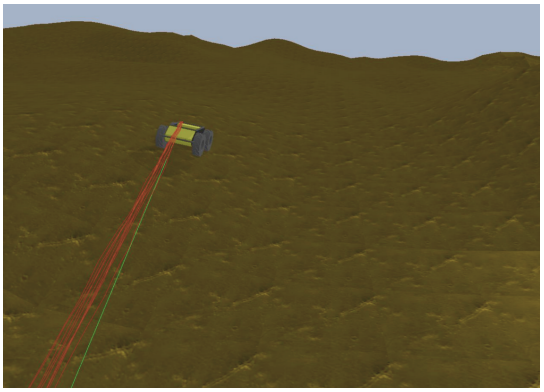

Fig. 3. Trajectory Planning

The environment supports step-based simulation and implements termination conditions based on height discontinuities, flipping, maximum horizon, or reaching the goal. The environment also exposes utilities for dataset collection, logging actions, rewards, and observations for offline training.

V. Approach

A. Probabilistic Model Training

Single-step learning on rough terrain quickly breaks down because prediction errors accumulate, causing the model to drift off the data manifold. To address this, we adopted a multistep training procedure in which the network is trained on short K-step trajectory segments sampled from real episodes. To model epistemic uncertainty which is how unsure the model is about its own predictions we roll out N different trajectories using variational dropout. By aggregating the negative log-likelihood across these N trajectories using a Log-Sum-Exp estimator, the model is forced to account for aleatoric as well as epistemic uncertainty.

Our dynamics model combines a convolutional encoder for the terrain heightmap with an LSTM that captures short-horizon temporal structure. The CNN extracts local geometric features that influence traction and slippage, while the LSTM propagates those features through time and models how robot motion evolves under successive control inputs. Together, they form a compact architecture capable of learning both spatial and temporal dependencies in the dynamics.

This structure forces the network to learn not just instantaneous motion but how prediction errors propagate over several steps, resulting in a more stable and realistic dynamics representation. Dropout is enabled throughout training, giving an approximation of epistemic uncertainty. This uncertainty-aware design is essential for downstream planning, which relies on the model to express both its predicted state change and its confidence.

At each step, the neural network outputs a 26-dimensional vector (13-D mean and 13-D log-variance) over the 13-dimensional state-transition vector

Algorithm 1: Probabilistic Model Training

for *number of epochs* **do**
- $\mathcal{L} \leftarrow 0$
- **for** *training batch size* (b) **do**
  - Sample parameters $\theta \sim p_\Theta(\theta)$
  - Sample K-step trajectory segments $\{(s_t, a_t, ..., a_{k-1})\}$ from random episodes
  - Normalize: $\tilde{s}_t, \tilde{a}_t$
  - Initialize particles: $\hat{s}_0^1, \ldots, \hat{s}_0^N \leftarrow s_0$
  - **for** $t \in [1, k]$ **do**
    - **for** $i \in [1, N]$ **do**
      - Sample particle transition: $\hat{s}_t^i \sim p_\theta(s_t \mid \hat{s}_{t-1}^i, a_{t-1})$
    - Estimate NLL: p;
    - Accumulate multistep loss: $\mathcal{L} \leftarrow \mathcal{L} - \frac{1}{k} \log p$
  - Update parameters: $\Theta \leftarrow \Theta - \alpha \nabla_\Theta \frac{\mathcal{L}}{b}$

B. Closed-Loop Prediction

Even an accurate probabilistic model is insufficient for planning if predictions assume open-loop execution. In rough terrain, deviations from the nominal path accumulate quickly unless corrected by feedback. Ignoring this effect would lead the planner to overestimate its ability to traverse difficult regions.

To address this, Algorithm 2 incorporates the robot's tracking controller directly into the prediction process. The planner first constructs a nominal trajectory by rolling out the model's mean predictions under the candidate action sequence. Then, multiple particles are propagated forward, each subjected to: 1) stochastic model samples (via dropout), and 2) corrective actions from the same PD controller used during real execution.

This produces a distribution of closed-loop trajectories that reflects how the robot would actually behave under feedback not how it would behave if it executed actions blindly. Controller-in-the-loop prediction stabilizes uncertainty growth, enabling meaningful comparison of candidate actions.

Algorithm 2: Closed-Loop Prediction

Input: Starting state s_0, feed-forward actions $\bar{a}_0, \ldots, \bar{a}_{T-1}$

$\bar{s}_0 \leftarrow s_0$

for $t \in [1, T]$ **do**

  $\bar{s}_t \leftarrow \mathbb{E}\big[s_t \sim p_\Theta(s_t \mid \bar{s}_{t-1}, \bar{a}_{t-1})\big]$

for $i \in [1, I]$ **do**

  $\theta^i \sim p_\Theta(\theta)$

  $\hat{s}^i_0, \hat{a}^i_0 \leftarrow s_0, \bar{a}_0$

  for $t \in [1, T]$ **do**

    $\hat{s}^i_t \sim p_{\theta^i}(s_t \mid \hat{s}^i_{t-1}, \hat{a}^i_{t-1})$

    $\hat{a}^i_t \leftarrow \bar{a}_t + f_{\text{track}}(\hat{s}^i_0, \ldots, \hat{s}^i_t, \bar{d})$

C. Divergence Constrained Optimization

Given these predictions, we search for trajectories that are both cost-effective and executable. Pure cost minimization tends to exploit model inaccuracies, producing plans that may appear optimal under the learned model but fail in the real environment. Penalizing uncertainty can mitigate this but introduces a competing objective that often discourages meaningful progress toward the goal.

Instead, we adopt a divergence-constrained formulation: the planner may select any trajectory as long as the predicted divergence between closed-loop and nominal behavior remains below a user-specified threshold. This threshold represents the maximum deviation the tracking controller can reasonably correct.

Algorithm 3 solves this constrained problem using an augmented Lagrangian approach. The optimizer: 1) proposes a sequence of control inputs, 2) evaluates the nominal and closed-loop trajectories via Algorithm 2, 3) measures their divergence, 4) increases a penalty multiplier if the divergence exceeds the allowed limit, and 5) iteratively updates the actions through gradient descent.

Algorithm 3: Divergence Constrained Optimization

Input: Starting state s_0

$\bar{a}_0, \ldots, \bar{a}_{T-1} \leftarrow$ random initialization

for *number of Augmented Lagrangian iterations* **do**

  for *number of gradient descent iterations* **do**

    $\bar{d}, \hat{D} \leftarrow$ predict nominal and closed-loop trajectories using Algorithm 2

    $u \leftarrow \max_t \sum_i \|\hat{s}^i_t - \bar{s}_t\|_2^2$

    $\mathcal{L} \leftarrow c(\bar{d}) + \lambda \max(u - U, 0)$

    $\bar{a}_0, \ldots, \bar{a}_{T-1} \leftarrow$ update using gradient $\nabla_{\bar{a}_0, \ldots, \bar{a}_{T-1}} \mathcal{L}$

  increase λ

D. Planning and Execution

The dynamics model trained in Algorithm 1 provides long-horizon predictions with uncertainty. Algorithm 2 translates proposed action sequences into realistic closed-loop trajectory distributions. Algorithm 3 then searches over action sequences, shaping them into trajectories that satisfy both the task objective and the divergence constraint.

Once optimization converges, the resulting nominal trajectory is tracked in real time using the same PD controller embedded during prediction. This consistency between predicted and executed feedback behavior is critical: the model confidently predicts exactly the feedback law that will be applied on hardware, yielding robust performance on rough terrain without needing rapid replanning.

RESULTS

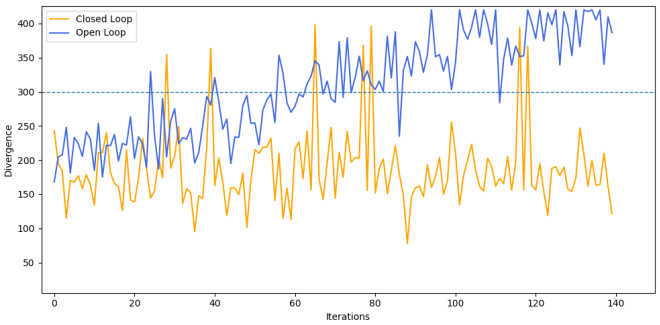

Fig. 4. Open vs Closed loop Divergence

Open-loop divergence starts near 200 (Fig. 4) and steadily increases, frequently exceeding the threshold U=300 after 60 iterations and reaching values in the 330–400 range.

In the closed-loop, divergence stays mostly within the 120–250 range and rarely exceeds the threshold. Unlike the open-loop curve, it does not exhibit sustained growth over time, and its fluctuations remain bounded throughout the horizon. On average, closed-loop divergence remains roughly 50–100 units lower than open-loop during the later roll-out stages.

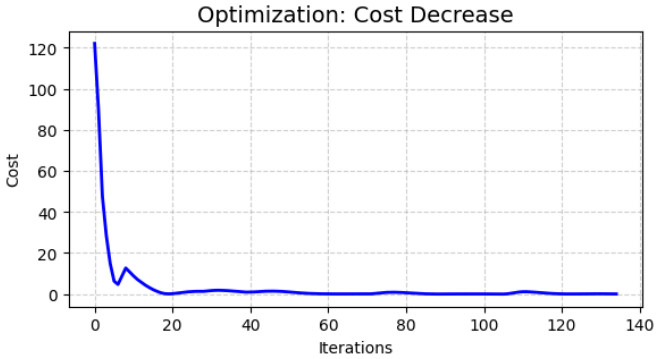

Fig. 5. Goal Distance Cost vs Iterations

The plot in Fig. 5 shows the optimization cost against optimization iterations. The cost drops sharply during the

early iterations and then slowly converges, meaning that the augmented Lagrangian optimizer successfully finds a lower-cost trajectory and then fine-tunes it until convergence.

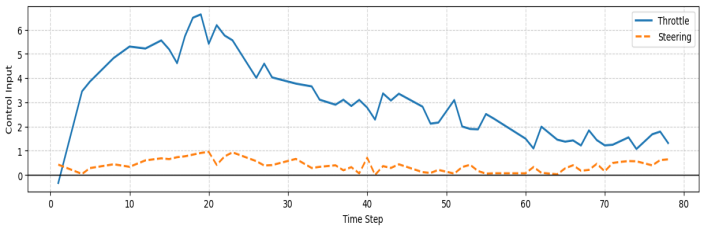

Fig. 6. Throttle vs Steering

Fig. 6 demonstrates how the control inputs (throttle, steering) change during execution. The throttle is high in the beginning and gradually decreases as the robot approaches the goal. Steering remains relatively low throughout, which indicates that the optimizer mainly selects forward motion with small corrective adjustments rather than making large turns.

Conclusion

This work presented a model-based reinforcement learning framework for rough-terrain navigation that integrates multistep probabilistic dynamics learning, closed-loop trajectory prediction, and divergence-constrained optimization. By modeling long-horizon uncertainty and incorporating feedback into prediction, the system generated stable trajectories in environments with highly divergent dynamics. In simulation, the optimizer reduced goal-distance cost by 99% within twenty iterations, kept closed-loop divergence 50–100 units below open-loop, and achieved success rates approaching 0.95 substantially outperforming cost-only and penalty-based baselines. These results show that feedback-aware predictions with divergence constraint prevents model exploitation and enables reliable long horizon planning.

The divergence constraint avoids high-uncertainty behaviors, making the method well suited for offline planning settings where interaction with the environment is limited. The particle-based closed-loop predictions can be efficiently parallelized on GPUs, allowing uncertainty propagation across long horizons at low computational cost. In our experiments, the system optimized trajectories of roughly 100 steps without requiring MPC-style replanning. The overall architecture remains modular. The dynamics model is agnostic to the tracking controller and reward structure, these components can be modified without retraining the model.

Current limitations highlight areas for future work. We found that the constrained optimizer can converge to local minima, particularly in highly irregular terrains where small initial action perturbations lead to different outcomes. Warm-starting with a path planner such as grid-based A* can noticeably improve convergence quality (as seen in [7]). Incorporating additional warm-start strategies like action sequences generated by stochastic sampling methods such as MPPI, CEM etc or LQR-guided initializations may further enhance robustness. The method also assumes access to a consistent height map, while common in practice, real maps may be sparse or noisy. Incorporating terrain-map uncertainty into prediction and divergence estimation is a promising direction for increasing real-world reliability. Our implementation uses variational dropout, however that can be replaced with Evidential Deep Learning, enabling a single forward pass to produce both predictions and epistemic uncertainty, substantially reducing the computational cost of divergence estimation.Finally, evaluating the full system on hardware remains an important next step for understanding performance under sensing noise, actuation delays, and unmodeled terrain interactions.

References

[1] K. Chua, R. Calandra, R. McAllister, and S. Levine. *Deep reinforcement learning in a handful of trials using probabilistic dynamics models*. arXiv preprint arXiv:1805.12114, 2018.

[2] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-aware reinforcement learning for collision avoidance," *arXiv preprint arXiv:1702.01182*, 2017.

[3] A. Kelly and A. Stentz, "Rough terrain autonomous mobility part 2: An active vision, predictive control approach," *Autonomous Robots*, vol. 5(2):163–198, 1998.

[4] P. Fankhauser, M. Bjelonic, C. D. Bellicoso, T. Miki, and M. Hutter, "Robust rough-terrain locomotion with a quadrupedal robot," in *IEEE International Conference on Robotics and Automation*, pp. 5761–5768, 2018.

[5] E. Talvitie, "Self-correcting models for model-based reinforcement learning," in *AAAI Conference on Artificial Intelligence*, vol. 31(1), 2017.

[6] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *International Conference on Machine Learning*, pp. 1050–1059, PMLR, 2016.

[7] S. J. Wang, S. Triest, W. Wang, S. Scherer, and A. M. Johnson, "Rough-terrain motion planning and control for wheeled robots." Dept. of Mechanical Engineering and Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA.

[8] C. Liu, J. Pan, and Y. Chang, "PID and LQR trajectory tracking control for an unmanned quadrotor helicopter," in *2016 35th Chinese Control Conference (CCC)*, pp. 10845–10850, IEEE, 2016.