Rough Terrain Navigation Using Divergence Constrained Model Based Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract: Autonomous navigation of wheeled robots in rough terrain environments has been a long standing challenge. In these environments, predicting the robot's trajectory can be challenging due to the complexity of terrain interactions, as well as the divergent dynamics that cause model uncertainty to compound and propagate poorly. This inhibits the robot's long horizon decision making capabilities and often lead to shortsighted navigation strategies. We propose a model-based reinforcement learning algorithm for rough terrain traversal that trains a probabilistic dynamics model to consider the propagating effects of uncertainty. During trajectory predictions, a trajectory tracking controller is considered to predict closed-loop trajectories. Our method further increases prediction accuracy and precision by using constrained optimization to find trajectories with low divergence. Using this method, wheeled robots can find non-myopic control strategies to reach destinations with higher probability of success. We show results on simulated and real world robots navigating through rough terrain environments.

Keywords: Rough Terrain Navigation, Model-Based Reinforcement Learning, Model Uncertainty

1 Introduction

Autonomous wheeled robots are useful for a variety of applications such as environmental monitoring, package delivery, and warehouse operation. Although these robots are highly capable of autonomously navigating through flat terrain, rough terrain environments still pose a challenge. Rough terrain navigation requires non-myopic decision making to avoid getting trapped in cul-de-sacs. This is challenging as the interactions between the robot and the unstructured terrain are complex and difficult to model [1]. Furthermore, the highly divergent dynamics can cause modeling errors to compound and propagate poorly along trajectories.

Methods that rely on accurate handcrafted dynamics models for decision making, e.g. [2, 3, 4], often perform poorly when predictions contradict reality. Some methods aim to combat this issue by formulating control policies that are robust against model uncertainty [5, 6, 7, 8]. Others abstract the robot's dynamics by using traversability maps instead to provide heuristics for how difficult different regions are to traverse [9, 10, 11, 12, 13, 14]. However, these approaches are limited to coarse approximations of traversability and don't consider dynamic interactions.

However, MBRL methods often have poorer performance as their decision making processes may exploit model inaccuracies. Previous MBRL algorithms have addressed this issue by considering prediction uncertainty during decision making [15, 16], though characterizing prediction uncertainty for rough terrain navigation is challenging since the divergent dynamics cause model uncertainty to propagate poorly over long prediction horizons.

We propose a MBRL algorithm for rough terrain navigation, summarized in Sec. 3 and Fig. 1. Like previous methods [15, 16], our method uses a probabilistic dynamics model to characterize prediction uncertainty. However, our approach considers and handles uncertainty differently based on three observations. First, we observe that dynamics models often propagate uncertainty poorly when predicting long horizon trajectories. We propose a training loss that considers how model uncertainty

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.

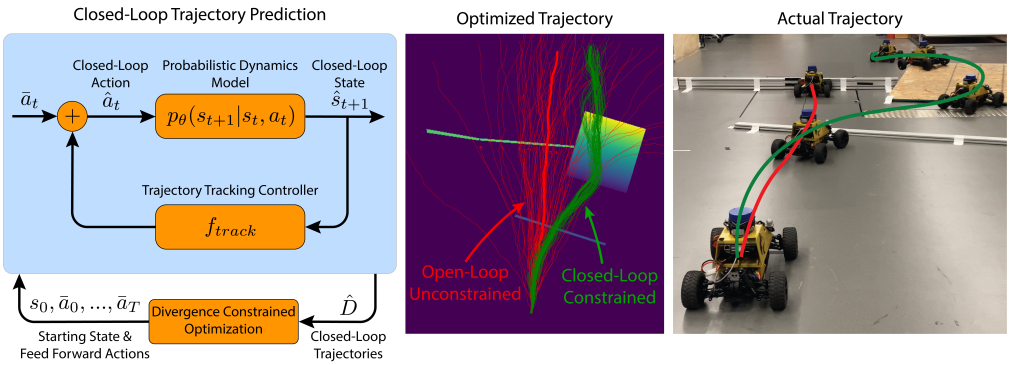


Figure 1: Left: An overview of the method. The probabilistic dynamics model is trained using a multistep loss that considers how uncertainty propagates. The trajectory tracking controller is used to predict a distribution of closed-loop trajectories. The divergence constrained optimizer is used to find a closed-loop trajectory with low divergence. Center: Example trajectory distribution found using our method (green) and a trajectory distribution found without using closed-loop predictions or constrained optimization (red). Right: The actual results of running the two optimized trajectories.

propagates along trajectories (Sec. 4). Second, since trajectory tracking controllers are commonly used to reliably execute planned motions, we consider the capabilities of these lower level controllers during trajectory prediction (Sec. 5). Third, we use constrained optimization to avoid trajectories that result in particularly high divergence (Sec. 6), similar to Convergent Planning [8]. Through simulation and hardware experiments, we show that our method can find non-myopic trajectories with high prediction accuracy and precision, increasing the likelihood of successful navigation (Sec. 7).

2 Related Works

Probabilistic Ensembles with Trajectory Sampling (PETS) [15] aims to make MBRL more robust against modeling errors by characterizing aleatoric and epistemic uncertainty with probabilistic ensemble models. Trajectory sampling is used to propagate prediction uncertainty along the trajectory. PETS optimizes trajectories based on expected rewards and has performed well on benchmark tasks [17]. However, using PETS on rough terrain is challenging due to the higher variance in predictions, which makes it more difficult to estimate expected reward using trajectory sampling. Furthermore, high reward expectation with high variance does not mean the robot will actually perform well.

Kahn et al. [16] use uncertainty in a model-based fashion for collision avoidance in navigation. They train an ensemble of models (via variational dropout [18]) to predict the probability of a collision. They then use this uncertainty to augment their task-specific cost function via a risk-averse prediction of the collision probability, which factors in the uncertainty of the collision predictor ensemble. This collision probability is used in their reward function, which is solved using MPC. This approach is shown to be effective with real-world experiments.

Recent work by Yu et al. [19] takes a more general approach to uncertainty-aware MBRL by learning an ensemble of dynamics models and augmenting arbitrary cost functions with a scaled penalty based on the uncertainty of the ensemble. The authors show that such an approach allows them to perform offline reinforcement learning, or reinforcement learning without access to the environment.

3 Method Overview

We formulate the rough terrain navigation problem as a Markov decision process (MDP). Our problem is defined as $(\mathcal{S}, \mathcal{A}, \mathcal{P}, C, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}(s'|s,a)$ is the non-deterministic discrete time transition dynamics, $C(s,a)$ is the cost function, and γ is the discount factor. We assume that the state includes terrain features (e.g. terrain height map).

Our approach falls under the MBRL paradigm. We train a probabilistic neural network to model the unknown transition dynamics $\mathcal{P}$ using previously collected data $\mathcal{D} = \{d_0, ..., d_M\}$, where $d_i = (s_0, .., s_T, a_0, ..., a_{T-1})$ is a state-action trajectory. We train the model to capture both aleatoric

and epistemic uncertainty using a multistep training loss that considers how model uncertainty propagates along trajectories (Sec. 4). Given an action sequence and starting state, the trained probabilistic model is used to predict a distribution of the robot's trajectory under the influence of a low level feedback controller (Sec. 5). Finally, a navigation controller is generated using constrained optimization to find a low cost trajectory to the goal. The solution is constrained based on the probabilistic model to avoid steps where the dynamics significantly diverges to ensure that the low level tracking controller can actually achieve the trajectory (Sec. 6). A method overview is shown in Fig. 1.

4 Probabilistic Dynamics Models

We propose a method for training a probabilistic dynamics model to predict long horizon trajectories. Similar to [15], we train the model to capture both epistemic and aleatoric uncertainty, however here we propose a multistep loss that considers how prediction uncertainty propagates along a trajectory.

4.1 Aleatoric Uncertainty

In the learned model we aim to capture aleatoric uncertainty by modeling the true transition dynamics $\mathcal{P}$ with a probabilistic neural network [20] parameterized by θ. Given a state s_t and action a_t at time t, the model predicts a probability distribution of possible next states. For our implementation, the network predicts the mean and log-variance of an uncorrelated gaussian distribution. The estimated likelihood of a next state s_{t+1} conditioned on s_t and a_t is denoted as $p_\theta(s_{t+1}|s_t, a_t)$. The training method in [15] uses a single step likelihood loss. However, this loss fails to consider how prediction uncertainty propagates when multiple single step predictions are chained for trajectory prediction. Furthermore, [21, 22] describe how states generated from earlier predictions may fall out of the distribution of training data, leading to poor predictions for later steps in a trajectory. We adapt [21, 22] to non-deterministic environments by proposing a probabilistic multistep loss that considers the propagation of model uncertainty instead of model error.

The multistep loss we use aims to minimize the negative log likelihood of states conditioned on all prior actions and the starting state of the trajectory, i.e. $\mathcal{L}_{ms}(s_t, A, \theta) := -\log p_\theta(s_t|A)$, where $A = \{s_0, a_0, ..., a_{t-1}\}$. The likelihood of states $s_1, ..., s_t$ conditioned on A can be found as follows:

$$\begin{aligned} p_\theta(s_1|A) &= p_\theta(s_1|s_0, a_0) \\ p_\theta(s_2|A) &= \mathbb{E}_{\hat{s}_1 \sim p_\theta(s_1|A)}\big[p_\theta(s_2|\hat{s}_1, a_0)\big] \\ &\vdots \\ p_\theta(s_t|A) &= \mathbb{E}_{\hat{s}_{t-1} \sim p_\theta(s_{t-1}|A)}\big[p_\theta(s_t|\hat{s}_{t-1}, a_{t-1})\big] \end{aligned} \tag{1}$$

Calculating $\mathcal{L}_{ms}$ exactly is intractable due to the expectations required. Instead we define upper bounds that can be used during training instead, denoted as $\mathcal{L}^N_{ms}$ where $N \in \mathbb{N}$,

$$\mathcal{L}^N_{ms}(s_t, A, \theta) := -\log \frac{1}{N} \sum_{i=1}^{N} p_\theta(s_t|\hat{s}^i_{t-1}, a_{t-1}) \tag{2}$$

where $\hat{s}^i_j$ is sampled recursively from $p_\theta(s_j|\hat{s}^i_{j-1}, a_{j-1})$ for $j < t$. As N approaches infinity, $\mathcal{L}^N_{ms}$ approaches the true multistep loss, $\mathcal{L}_{ms}$. Furthermore, due to Jensen's inequality:

$$\mathbb{E}\big[\mathcal{L}^1_{ms}(s_t, A, \theta)\big] \geq \mathbb{E}\big[\mathcal{L}^2_{ms}(s_t, A, \theta)\big] \geq ... \geq \mathcal{L}_{ms}(s_t, A, \theta) \tag{3}$$

As such, minimizing $\mathcal{L}^N_{ms}$ using any stochastic optimization methods such as Stochastic Gradient Descent (SGD) [23] will effectively also minimize the true loss. Any value could be used for N, however larger values give more realistic estimates at the expense of increased computation.

4.2 Epistemic Uncertainty

Epistemic uncertainty is defined as model uncertainty due to a poorly distributed or lack of training data. In the context of deep learning, it is typically quantified using a prediction disagreement metric via an ensemble of neural network models. Some popular methods use explicit ensembles of

models [15], while other methods such as Bayesian neural networks [24] or hypernetworks [25, 26] parameterize models via uncertainty in the weights of the neural network.

In our work, we use variational dropout to characterize epistemic uncertainty [27, 18]. With this approach, inputs to the neural network's fully connected layers are randomly dropped (set to zero) with some probability. We define Θ as the parameters of the neural network dynamics model with no dropout applied. Using dropout on this model will effectively result in a distribution of models. We denote the probability of sampling a model, parameterized by θ, as $q_\Theta(\theta)$.

During training, we randomly sample parameters θ from $q_\Theta(\theta)$ and calculate the multistep training loss described in Sec. 4.1. We then update Θ to minimize training loss. The complete model training procedure is summarized in Algorithm 1.

5 Closed-Loop Trajectory Prediction

Imprecise trajectory predictions are problematic for two reasons. First, predicted trajectory distributions with high variance provides little insight into the robot's true outcome. Second, the standard error of Monte Carlo predictions scales linearly with prediction variance and inversely with the square root of number of samples. This means that using Trajectory Sampling [15] may require an intractable number of samples when prediction variance is high. We observe that it is common to use a low-level trajectory tracking controllers to correct for deviations of the true trajectory from the desired (predicted) trajectory. Our method aims to predict robot trajectories under the influence of a low-level trajectory tracker controllers. Doing so allows prediction of less divergent robot trajectory distributions and better captures the capabilities of the closed-loop robot control system.

We assume that the controllers take the form:

$$a_t = \bar{a}_t + f_{track}(s_0, ..., s_t, \bar{d}), \tag{4}$$

where $s_0, ..., s_t$ are the robot's actual states, and $\bar{d} = (\bar{s}_0, ..., \bar{s}_T, \bar{a}_0, ..., \bar{a}_{T-1})$ is a reference trajectory containing reference states and feed forward actions. a_t is the combination of the planned, feed-forward action $\bar{a}_t$ and the corrective action applied to the robot to track the reference trajectory.

Training a dynamics model to directly predict the robot's closed loop behavior is difficult as the controller's inputs are high dimensional and unbounded. Instead, our method first predicts a nominal reference trajectory given a starting state and feed forward actions,

$$\bar{s}_{t+1} = \mathbb{E}\big[s_{t+1} \sim p_\Theta(s_{t+1}|\bar{s}_t, \bar{a}_t)\big]. \tag{5}$$

Our model outputs the predicted distribution's mean which is the expected value above.

A distribution of closed-loop trajectories is then predicted by propagating a set of particles forward in time using the probabilistic dynamics model from Section 4. At each time step, corrective actions for each particle is calculated using f_{track} based on the predicted nominal reference trajectory and the feed forward actions. We denote the predicted closed-loop trajectory distribution as $\hat{D} = \{\hat{d}^0, ..., \hat{d}^I\}$, where $\hat{d}^i = (\hat{s}^i_0, ..., \hat{s}^i_T, \hat{a}^i_0, ..., \hat{a}^i_{T-1})$ is a particle's state-action trajectory. The procedure for predicting closed-loop trajectories is summarized in Algorithm 2.

6 Divergence Constrained Optimization

In MBRL, low cost trajectories are found via trajectory optimization through the learned model. However, naively minimizing cost allows the optimizer to exploit modeling errors, resulting in a trajectory with low predicted cost, but high cost under the true dynamics. Previous work [19] prevented the optimizer from exploiting modeling errors by penalizing trajectories with high prediction uncertainty. This approach may be problematic as the penalty will oppose the cost objective. For example, if a robot remains stationary, its trajectory has low prediction uncertainty, but high task cost.

Instead of penalizing uncertainty, our method uses constrained optimization to find low cost trajectories with divergence bounded by some U, as trajectory trackers are usually effective as long as the true trajectory does not divergence too much from the reference. Values for U must be selected based on the capabilities of the trajectory tracker. Given a nominal trajectory $\bar{d}$ and predicted closed-loop

Algorithm 1: Probabilistic Model Training

for *number of epochs* **do**
  $\mathcal{L} \leftarrow 0$
  for *training batch size(b)* **do**
    $\theta \sim p_\Theta(\theta)$
    sample $A = \{s_0, a_0, ..., a_{T-1}\}$
    $\hat{s}_0^1, ...\hat{s}_0^N \leftarrow s_0$
    for $t \in [1, T]$ **do**
      for $i \in [1, N]$ **do**
        $\hat{s}_t^i \sim p_\theta(s_t | \hat{s}_{t-1}^i, a_{t-1})$
      $p \leftarrow \frac{1}{N} \sum_{i=1}^{N} p_\theta(s_t^i | \hat{s}_{t-1}^i, a_{t-1})$
      $\mathcal{L} \leftarrow \mathcal{L} - \frac{1}{T} \log p$
  $\Theta \leftarrow$ update using gradient $\nabla_\Theta \frac{\mathcal{L}}{b}$

Algorithm 2: Closed-Loop Prediction

Input: Starting State s_0,
  Feed-Forward Actions $\bar{a}_0, ..., \bar{a}_{T-1}$
`// Predict Nominal Trajectory`
$\bar{s}_0 \leftarrow s_0$
for $t \in [1, T]$ **do**
  $\bar{s}_t \leftarrow \mathbb{E}[s_t \sim p_\Theta(s_t | \bar{s}_{t-1}, \bar{a}_{t-1})]$
`// Closed-Loop Prediction`
for $i \in [1, I]$ **do**
  $\theta^i \sim p_\Theta(\theta)$
  $\hat{s}_0^i, \hat{a}_0^i \leftarrow s_0, \bar{a}_0$
  for $t \in [1, T]$ **do**
    $\hat{s}_t^i \sim p_{\theta^i}(s_t | \hat{s}_{t-1}^i, \hat{a}_{t-1}^i)]$
    $\hat{a}_t^i \leftarrow \bar{a}_t + f_{track}(\hat{s}_0^i, ..., \hat{s}_t^i, \bar{d})$

Algorithm 3: Divergence Constrained Optimization

Input: Starting State s_0
$\bar{a}_0, ..., \bar{a}_{T-1} \leftarrow$ 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
    $u \leftarrow \max_t \sum_i \|\hat{s}_t^i - \bar{s}_t\|_2^2$
    $\mathcal{L} \leftarrow c(\bar{d}) + \lambda \, max(u - U, 0)$
    $a_0, ..., a_{T-1} \leftarrow$ update using gradient $\nabla_{a_0,...,a_{T-1}} \mathcal{L}$
  Increase λ

trajectory distribution $\hat{D}$, we define divergence as follows:

$$u(\bar{d}, \hat{D}) = \max_t \frac{1}{I} \sum_i^I \|\hat{s}_t^i - \bar{s}_t\|_2^2. \tag{6}$$

The constrained optimization problem we aim to solve is define as follows:

$$\min_{\bar{a}_0,...,\bar{a}_{T-1}} \quad C(\bar{d}) \qquad \text{s.t.} \quad u(\bar{d}, \hat{D}) < U \tag{7}$$

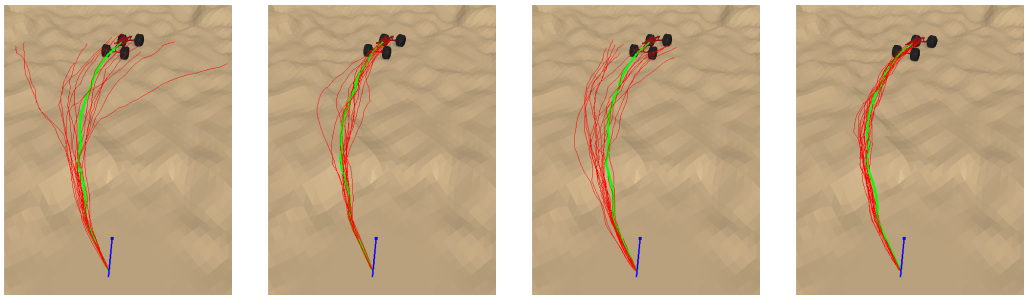
We use an Augmented Lagrangian optimizer to solve (7), as shown in Algorithm 3. During optimization, the estimated divergence is thresholded, $max[u - U, 0]$, scaled by a penalty factor λ, and added to the nominal trajectory cost. λ starts small, but is progressively increased.

7 Experiments

We compare our navigation method against others in simulation and in the real world. We used PyBullet [28] for our simulation environment and a MuSHR robot [29] for our real world experiment.

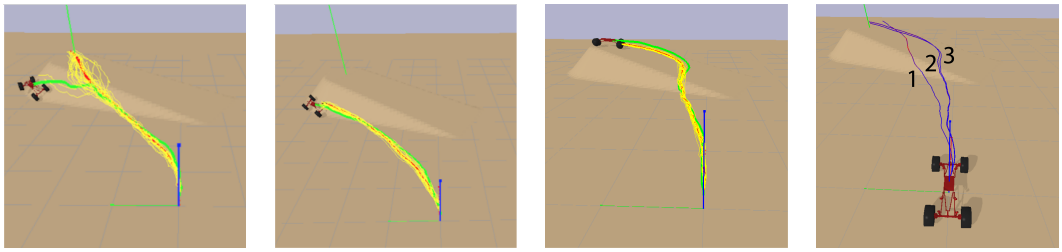
Training data consisted of terrain height maps and state-action trajectories, which were split into 5 step trajectories during training. Simulation training data was collected in randomly generated environments (similar to Figure 4) using random actions. Real world training data was collected in one environment using manual controls with added noise. This environment had same obstacles as the test environment shown in Figure 5a, but arranged differently. Motion capture was used to estimate the real robot's pose, and terrain height maps were manually created.

We trained the dynamics models to predict state transition probability, given an action and an observation. For simulation, observations included the robot's tilt ($\mathbb{R}^3$ gravity vector) and body velocity ($\mathbb{R}^6$ twist vector). State transitions consisted of the robot's change in pose (in $SE(3)$), and new



(a) Single Step (b) Single Step Tracked (c) Multistep (d) Multistep Tracked

Figure 2: Predicted robot trajectories using different prediction methods The predicted trajectories are shown in red, and the true trajectories are shown in green.



(a) No uncertainty (b) Penalty (c) Constraint (d) AL Optimization

Figure 3: Ramp climbing trial. The optimized, predicted closed-loop, and actual trajectories are shown in red, yellow, and green respectively. (d) shows the Augmented-Lagrangian based optimizer progressively satisfying the divergence constraint, with red indicating higher divergence.

body velocity. For the real robot, observations included the robot's previous state transition. State transitions included the robot's change in pose (in $SE(2)$). Observations also included a robot centric terrain height map cropped from the world height map. During execution, simple PD trajectory tracking controllers were used. The task cost was final squared distance to the goal.

Model Accuracy

Using the simulation dataset, we trained one dynamics model using a single-step negative log-likelihood loss, and another dynamics model using our proposed multistep loss, $\mathcal{L}_{ms}^{128}$, with Algorithm 1. We used these two models to estimate the probability distribution of the robot's state at the end of a 32 step trajectory by propagating 512 particles using the trained models and averaging the predicted distribution from all particles. We simulated the robot's true trajectory and recorded the log-likelihood of the true state given the estimated distributions. We repeated this process with the closed-loop trajectory predictions using Algorithm 2. The average log-likelihoods over 100 trials are listed in Table 1, which shows an improvement from both the multistep loss and the tracking controller. The true trajectories and some example particle trajectories are shown in Fig. 2.

Simulated Ramp Climbing

We compare the performance of different trajectory optimization methods for the task of climbing up a simulated ramp. The three trajectory optimization frameworks used were: 1) an optimizer that only minimizes the goal cost, 2) an optimizer that adds a scaled divergence penalty, and 3) the proposed divergence constrained optimizer. The test environment and results are shown in Figure 3.

	Single-Step	Single-Step Tracked	Multistep	Multistep Tracked
Log-Likelihood	-692.7 $\pm$ 470.0	-242.5 $\pm$ 364.6	-110.0 $\pm$ 166.6	**-31.1** $\pm$ **48.5**

Table 1: Mean and standard deviation of prediction log-likelihood for 100 trials.

	Planner	No Uncertainty	Penalty	Constraint
Tracking	72 %	66 %	60 %	**82 %**
No Tracking	-	62 %	32 %	64 %

Table 2: Random terrain results. Success percentage over fifty random terrains, using the same terrain set for all methods. The planner does not provide actions, so tracking was required.

	Planner	No Uncertainty	Penalty	Constraint
Cost	2.0892	1.8792	0.9561	**0.9109**
Divergence	0.2039	0.2350	**0.1220**	0.1529

Table 3: Results of random terrain testing. Despite having higher divergence than the penalty method, the constrained method has lower final cost.

When divergence is not considered, the optimizer exploits errors in the nominal prediction. The optimized trajectory is short, but has high divergence. During execution, the robot slides off the ramp due to insufficient friction. Conversely, using divergence as a penalty leads to overly conservative trajectories, even after tuning penalty scaling. The optimizer minimizes divergence, but loses sight of the actual cost. With the divergence constrained optimizer, optimized trajectories had low cost and low divergence, allowing the robot to consistently reach the goal.

Navigating Over Random Simulated Terrain

We compared different navigation methods in randomly generated rough terrain environments. The robot was tasked with driving over large terrain obstacles to a goal $7m$ away. In this environment, we evaluated the importance of including tracking during prediction (Algorithm 2) and the importance of using divergence as an optimization constraint (Algorithm 3). We used the same three optimization frameworks from the ramp test (no uncertainty considered, divergence penalty, divergence constrained) and added an A* path planning method using the terrain cost from [30], which considered local terrain smoothness, slope, and curvature. Table 2 shows each method's success rate, defined as ending within $0.5m$ of the goal. We also report the final cost and standard deviation across 250 rollouts for each method with tracking (Table 3). All methods were allowed the same number of gradient steps and used the same underlying tracker for both optimization and tracking.

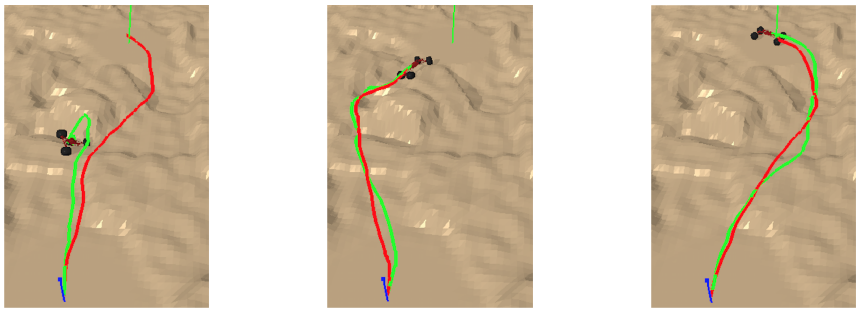
Overall, we find that the constrained optimization has the lowest final cost and highest success rate. As expected, the penalty method has the lowest divergence since it directly minimizes it, however this did not translate to higher success rate. Figure 4 shows one example trial that illustrates the benefits of our algorithm. When not considering uncertainty, the optimizer finds a solution that is too close to dangerous obstacles. The robot's true trajectory deviates from the predicted trajectory causing it to collide with the obstacle and roll over. Conversely, the penalty method finds an overly conservative trajectory that has the lowest divergence but does not reach the goal. The divergence constrained method strikes a middle ground finding a safe trajectory to the goal.

Additionally, we found that using the planner to seed the optimization gave the best overall success rate of 88%. This suggests that the optimizer still has trouble escaping local minima and that combining planning with optimization may be a successful direction for future work.

Real World Navigation

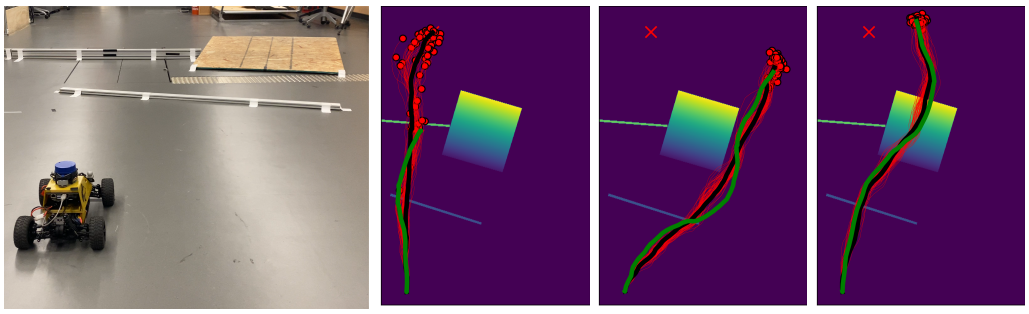
We also compared the three optimization frameworks in a real world navigation test in which the robot was tasked with navigating to a goal $5m$ away on the other side of an obstacle field consisting of a short obstacle, tall obstacle, and ramp. The robot was capable of driving over the short obstacle but not the tall one. The ramp was taller than the tall obstacle, but could be driven over from certain directions, so traversability was not a simple function of height. The test environment and one example trial is shown in Fig. 5. Table 4 shows the actual cost and success rate for 10 trials.

When no uncertainty was considered, the optimizer exploited nominal prediction errors and found impossible trajectories through the tall obstacle. For 9 out of 10 trials, the robot got stuck on the tall obstacle. For 1 trial, the true trajectory deviated from the nominal enough for the robot to catch the edge of the ramp and make it to the goal. When penalizing divergence, the optimizer avoided the



(a) No uncertainty (b) Uncertainty penalty (c) Uncertainty constraint

Figure 4: One trial in random terrain. Predicted nominal trajectories are shown in red, and the true trajectories are shown in green. The vertical green lines mark the goal.



(a) Test Environment (b) No Uncertainty (c) Penalty (d) Constraint

Figure 5: One trial of navigating a real world robot across an obstacle field. The goal (red cross), predicted closed-loop trajectories (red lines with last time step marked with dots), predicted nominal trajectory (black line), and actual robot trajectory (green line) are shown over a height map.

ramp 6 times to minimize divergence. When using divergence constrained optimization, the robot made it over the ramp 8 times, and on average got closer to the goal.

	No Uncertainty	Penalty	Constraint
Actual Trajectory Cost	3.57±1.30	5.11±4.45	**1.68±1.42**
Success Rate	10%	40%	**70%**

Table 4: Real world navigation results. Actual cost values are mean and standard deviation for 10 trials. Success is defined as getting within 1.5m of the goal.

8 Conclusion

We propose a novel model based reinforcement learning algorithm for rough terrain traversal. The method trains dynamics models to consider uncertainty propagation, and uses the models to predict closed-loop trajectories. This allows for accurate and precise trajectory predictions. In addition, it avoid highly divergent trajectories using constrained optimization. We show in simulation that the method allows for better trajectory predictions. In addition, we show in multiple simulation and real world environments that the method allows for higher performance.

In the future, we believe that performance can be improved with more intelligent initialization of the optimization. We also believe that our control framework can be extended to tasks besides rough terrain navigation.

References

[1] K. Nagatani, G. Ishigami, and Y. Okada. Modeling and control of robots on rough terrain. In *Springer handbook of robotics*, pages 1267–1284. Springer, 2016.

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

[3] T. M. Howard and A. Kelly. Optimal rough terrain trajectory generation for wheeled mobile robots. *The International Journal of Robotics Research*, 26(2):141–166, 2007.

[4] T. Howard, M. Pivtoraiko, R. A. Knepper, and A. Kelly. Model-predictive motion planning: Several key developments for autonomous mobile robots. *IEEE Robotics & Automation Magazine*, 21(1):64–73, 2014.

[5] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1–2):99–134, 1998.

[6] A. T. Schwarm and M. Nikolaou. Chance-constrained model predictive control. *AIChE Journal*, 45(8):1743–1752, 1999.

[7] G. Kewlani, G. Ishigami, and K. Iagnemma. Stochastic mobility-based path planning in uncertain environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1183–1189, 2009.

[8] A. M. Johnson, J. E. King, and S. Srinivasa. Convergent planning. *IEEE Robotics and Automation Letters*, 1(2):1044–1051, 2016.

[9] P. Papadakis. Terrain traversability analysis methods for unmanned ground vehicles: A survey. *Engineering Applications of Artificial Intelligence*, 26(4):1373–1385, 2013.

[10] M. Agrawal, K. Konolige, and R. C. Bolles. Localization and mapping for autonomous navigation in outdoor terrains: A stereo vision approach. In *IEEE Workshop on Applications of Computer Vision*, pages 7–7, 2007.

[11] D. Maturana, P.-W. Chou, M. Uenoyama, and S. Scherer. Real-time semantic mapping for autonomous off-road navigation. In *Field and Service Robotics*, pages 335–350. Springer, 2018.

[12] N. Hirose, A. Sadeghian, M. Vázquez, P. Goebel, and S. Savarese. GONet: A semi-supervised deep learning approach for traversability estimation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3044–3051, 2018.

[13] R. O. Chavez-Garcia, J. Guzzi, L. M. Gambardella, and A. Giusti. Learning ground traversability from simulations. *IEEE Robotics and Automation letters*, 3(3):1695–1702, 2018.

[14] D. Kim, J. Sun, S. M. Oh, J. M. Rehg, and A. F. Bobick. Traversability classification using unsupervised on-line visual learning for outdoor robot navigation. In *IEEE International Conference on Robotics and Automation*, pages 518–525, 2006.

[15] 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.

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

[17] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI gym. *arXiv preprint arXiv:1606.01540*, 2016.

[18] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059. PMLR, 2016.

[19] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Zou, S. Levine, C. Finn, and T. Ma. MOPO: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

[20] D. F. Specht. Probabilistic neural networks. *Neural networks*, 3(1):109–118, 1990.

[21] E. Talvitie. Self-correcting models for model-based reinforcement learning. In *AAAI Conference on Artificial Intelligence*, volume 31(1), 2017.

[22] E. Talvitie. Model regularization for stable sample rollouts. In *UAI*, pages 780–789, 2014.

[23] S. Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4–5): 185–196, 1993.

[24] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015.

[25] D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[26] D. Krueger, C.-W. Huang, R. Islam, R. Turner, A. Lacoste, and A. Courville. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*, 2017.

[27] D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. *arXiv preprint arXiv:1506.02557*, 2015.

[28] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.

[29] S. S. Srinivasa, P. Lancaster, J. Michalove, M. Schmittle, C. Summers, M. Rockett, J. R. Smith, S. Choudhury, C. Mavrogiannis, and F. Sadeghi. MuSHR: A low-cost, open-source robotic racecar for education and research. *arXiv preprint arXiv:1908.08031*, 2019.

[30] 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*, pages 5761–5768, 2018.