A Practical Approach to Learning Dynamics for Rough Terrain Navigation

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Mechanical Engineering

Sean J. Wang

B.S., M.S., Mechanical Engineering, University of California, Santa Barbara

Carnegie Mellon University

Pittsburgh, PA

February, 2024

Acknowledgement

I would like to acknowledge the funding that made this work possible, provided by Chevron Technology Center and a Tata Consultancy Services Presidential Fellowship.

I would like to thank my advisor and committee chair, Prof. Aaron Johnson, for his endless support and guidance. Since the beginning of my journey at CMU, he has encouraged me to explore new ideas, all the while supporting me throughout the process.

In addition, I would like to thank other members of my thesis committee, Prof. Sebastian Scherer, Prof. Srikanth Saripalli, and Prof. Ding Zhao, for all their insightful feedback that helped guide this dissertation. To all my collaborators, Sam Triest, Wenshan Wang, Matthew Sivaprakasam, Honghao Zhu, and Ankit Bhatia - thank you for all the intellectual discussions and help with running experiments on the robots.

I would also like to thank all past and present members of the Robomechanics Lab for contributing to such a positive environment. The time spent together, both inside and outside of the lab, has truly been a pleasure. Specifically to Joe Norby, Catherine, Joe Payne, Nathan, Paul, James, Ardalan, Justin, Vivek, and Aja - it has been a pleasure to share so many different offices with you. To Ardalan, thank you for joining me on all the adventures Piximo has led us on. I am excited to see all the future developments from everyone.

I would also like to thank my Golden Retriever, Mija, for all her emotional support, despite her endless demand for attention, chaos wreaked on my belongings, and financial burden. Also, thank you Joe, Stutt, Suji, Fernanda, Ian, and Jon for helping to babysit.

To all my friends in and out of Pittsburgh, thank you for encouraging me to take time off when I most needed it. I could not ask for a better group of friends with whom to explore all that Pittsburgh and other visited cities have to offer.

To my parents, Rebecca, and Christine - thank you for always encouraging me to pursue my dreams, and supporting me on the difficult journey there. Last, but not least, to my partner, Jamie - thank you for believing in me and supporting me. I cannot wait to start our next journey together.

Abstract

Unmanned ground vehicles offer great potential, but struggle in rough terrain environments due to their inability to reason about complex dynamics over longer horizons. Thus, driving strategies are often short-sighted and underutilize the robot's dynamic capabilities. This thesis addresses the challenges of long-horizon, dynamics-aware decision making in rough terrain, arising from the inability to model complex system dynamics. The findings reveal two key insights: 1) leveraging low-quality simulation data can reduce training requirements of real-world dynamics models; and 2) long-horizon decision making can still utilize imprecise dynamics models through careful handling of prediction uncertainty.

We first discuss our model-based reinforcement learning approach for rough terrain navigation. This approach trains a dynamics model to predict the robot's trajectory over uneven terrain and capture prediction uncertainty. Decision making uses this model along with a closed-loop divergence constraint to aid in longer-horizon trajectory prediction and prevent exploitation of potentially problematic modeling errors. We show that this approach leads to robust, non-myopic driving strategies that take full advantage of the robot's capabilities.

Next, we explore leveraging simulation to reduce real-world training data requirements. This culminates in an approach that uses a large variety of simulated systems to train a dynamics model that quickly and probabilistically adapts to any new, including real-world, target system using any available target system data. Using this model within an uncertainty-aware decision making framework results in safe, albeit low performance, driving upon initialization. As more target system observations are collected, the adaptive dynamics model becomes more tailored to the target system resulting in increased driving performance.

Finally, we combine these concepts to form a non-myopic rough terrain navigation framework that can quickly and robustly adapt to new target systems. We show that upon initialization, this framework chooses conservative routes that avoids obstacles. However, after just one demonstration of driving over obstacles, the framework chooses more aggressive routes over obstacles that match the system's capabilities.

Contents

List of Figures

Chapter 1

Introduction

Unmanned ground vehicles (UGVs) have demonstrated remarkable utility in a broad spectrum of applications, ranging from industrial automation and logistics [1–4] to planetary exploration and hazardous environment characterization [5–7]. These robots, capable of navigating various environments autonomously, have significantly reduced human exposure to hazardous conditions and streamlined numerous repetitive tasks. Historically, however, the operational domain of these robots have been confined to structured terrains, typically characterized by flat, uniform, high-friction surfaces. This limitation has predominantly restricted their applications to indoor settings or well-paved environments.

Recent advancements in the field of robotics have begun to address the challenges of off-road autonomy [8–11]. Notable research efforts have focused on enhancing robots' ability to identify and circumvent problematic terrain features, e.g. challenging soil types or steep slopes [12, 13]. Alongside avoiding problematic areas, other research streams have concentrated on quick decision-making algorithms designed to recover from wheel slip and other disturbances stemming from unexpected environmental interactions, allowing UGVs to travel at high speed on loose soil [14, 15].

Despite significant advancements in the field of autonomous UGV navigation, numerous environments still remain challenging for these autonomous systems. Figure 1.1 showcases

Figure 1.1: Wheeled robot in challenging rough terrain environments.

several examples of particularly challenging rough terrain environments characterized by abrupt elevation changes and diverse ranges of surface types such as rock, dirt, or concrete. While these environments are challenging, human teleoperators are capable of driving the robot through such environments by leveraging their intuitive understanding of vehicle dynamics and capabilities in different types of terrain. However, current autonomous navigation controllers lack this level of sophisticated reasoning.

Navigating such challenging environments presents a unique challenge for autonomous UGVs as it requires long horizon decision making under careful consideration of the vehicle's dynamics and complex robot-terrain interactions. These environments are often littered with obstacles that block the robot's path to its destination, so the robot must carefully reason about which obstacles are traversable and which are are not. However, distinguishing the traversability of different obstacles is not simple. For example, the angle at which the robot approaches an obstacle may significantly influence its traction and stability. This variation leads to differing levels of slippage, which in turn directly affects the robot's capability to traverse the obstacle. The traversability of different obstacles may also depend on the vehicle's momentum. For example, certain obstacles may require the robot to increase speed to build up momentum. However, too much momentum may lead to loss of control or inability to stop in time. Furthermore, the robot must also consider how its dynamics and capabilities change depending on what kind of surface it is driving on. For example, a vehicle

may be capable of ascending an incline at low speeds if it is driving on high-friction concrete. However, the same incline covered in loose dirt might require the vehicle to increase speed to overcome reduced traction by leveraging momentum.

The robot must not only consider obstacles in its immediate vicinity but also consider potential obstacles further along its path. Adopting a short-sighted, greedy approach of heading straight towards the goal can lead the robot into dead ends. For instance, while initial obstacles on a direct route might be manageable, subsequent ones could be impassable leaving the robot stuck in a dead end. Long-horizon decision making is especially important for environments with complex obstacle layouts that resemble intricate mazes. Considering future obstacles and planning well in advance enables the robot to opt for less direct but more reliable paths and ensure that it can traverse all obstacles encountered.

1.1 Current Approaches to Off-Road Navigation

Current rough terrain navigation frameworks predominantly fall into two categories: Hierarchical Planning and Controls, and Model Predictive Controls. Hierarchical Planning and controls involve a multi-layered strategy, where higher layers handles coarse decision-making, such as route planning, while lower layers focus on real-time corrections and fine-tuning of the vehicle's movements. Model Predictive Controls, on the other hand, present a unified and integrated method, operating continuously within a feedback loop. This approach is characterized by its rapid and iterative decision-making process, which constantly adjusts the vehicle's trajectory and actions in response to real-time sensor feedback. It's important to note that these frameworks are often used in tandem. For instance, Model Predictive Controls is commonly used as the lower layer within a hierarchical scheme, executing the plans chosen by the higher layer [16, 17].

Additionally, Reinforcement learning has recently emerged as another promising approach to controlling complex robotic systems [18–20]. This approach adopts a data-driven strat-

egy, wherein the robot iteratively refines its decision-making policy through repeated trials. Again, the boundaries between these categories are not always well defined with Reinforcement learning methods often adopting a hierarchical structure [21] or model predictive control structure [15]. Despite its potential, reinforcement learning faces practical challenges for the task of rough terrain navigating of real-world robots. We discuss these limitations below and aim to address them throughout this dissertation.

Hierarchical Planning and Controls

Hierarchical approaches towards navigation have been predominant in early off-road navigation frameworks and continue to be heavily utilized in current systems [9, 22–24]. In these frameworks, the navigation task is primarily divided into two layers: a high-level path planning layer and a low-level controls layer, each handling the problem at distinct levels of abstractness. At the high level, the focus is on coarse, strategic planning. This involves making non-myopic decisions to identify safe routes through the environment that avoid potential hazards. On the other hand, the low level focuses on responding in real-time to any environmental disturbances to ensure that the vehicle adheres to the route chosen by the high level path planner. This includes adjusting to terrain irregularities, unexpected slip, or any other factors that cause discrepancies between the robot's true and planned trajectory.

During high-level path planning, the complexity of off-road navigation is simplified by abstracting away finer details. The robot's movements are commonly approximated using basic kinematic models such as the unicycle [25] or kinematic bicycle model [26]. These kinematic models ignore the complex effects of vehicle dynamics and robot-terrain interactions to provide a coarse understanding of what paths are feasible assuming minimal slip and relatively flat terrain.

In many cases, modeling errors arising from these simplifying assumptions at the high level are non-problematic as the low-level controller can compensate for such inaccuracies. However, in more challenging environments, such assumptions may pose significant diffi-

culties to the low-level controller. For example, paths through deep mud or steep slippery slopes might be deemed feasible by the high-level planner's simplified models, but in reality are beyond the physical limits of the robot.

Traversability mapping approaches aim to bridge the gap between high-level route selection and low-level control capabilities [13]. These approaches identify potentially problematic regions within the environment and generate traversability cost maps to bias high level planning towards less problematic regions. Various methods exist to generate these maps. One common approach is to generate these maps using geometric features of the terrain such as slope or roughness [27, 28]. Another popular technique includes leveraging visual sensors to categorize terrain types and assess their traversability [12, 29, 30].

However, a notable shortcoming of many of these approaches is the lack of consideration for vehicle dynamics in the traversability assessment. For rough terrain navigation, factors such as the vehicle's momentum, center of mass, traction, and other dynamic effects are important to consider. The planner's choice of speed and angle of approach can significantly alter these dynamic factors, thereby changing the traversability of certain obstacles. By excluding vehicle dynamics from the traversability assessment, these methods may have to adopt an overly conservative stance, evaluating terrain navigability based on worst-case driving scenarios. This conservative assessment could lead to underutilization of the vehicle's true capabilities, thereby limiting the efficiency and effectiveness of the navigation strategy, and in extreme cases prevent the robot from reaching its destination.

Model Predictive Controls

Another approach to off-road navigation is Model Predictive Control (MPC), where the robot continuously optimizes its trajectory in real-time using a predictive model [31–33]. This approach has two notable benefits over hierarchical planning and controls that are particularly advantageous for off-road navigation. First, by leveraging a more sophisticated predictive model that captures effects of momentum, traction and other dynamic effects,

trajectory optimization can generate dynamically feasible trajectories that utilize the full capabilities of the robot.

Second, the iterative nature of the trajectory optimization process in MPC makes it inherently robust against inaccuracies in the predictive model. Given the complexity of both the robot and the rough terrain environments, discrepancies between the robot's actual and predicted trajectory are inevitable. MPC addresses this issue by continuously reoptimizing the robot's trajectory based on real-time sensor feedback, allowing MPC to quickly accommodate for any variances between predicted and actual outcomes.

While this iterative optimization process increases robustness towards modeling errors, it also leads to one notable disadvantage: heavier computational demands. Due to computational constraints, optimization horizons for MPC are often quite short leading to more myopic strategies. Previously, numerous methods have been used to solve the nonlinear optimization problem within an MPC framework including nonlinear programming [34], differential dynamic programming [35], interior point optimization [36], and sequential quadratic programming [37]. More recently, Model Predictive Path Integral (MPPI) [14] has gained prominence due to its parallelizable sampling-based methodology, which is particularly well-suited for Graphic Processing Units (GPUs). MPPI's compatibility with GPUs allows for more efficient computation and makes it a promising solution for both extending the optimization horizon and control rate of MPC. In [15], this more computationally efficient MPC approach is used to control a robot at 60 Hz optimizing over a 2.5 second horizon using a fairly simple neural network dynamics model with 2 hidden layers of 32 neurons each.

While this horizon is still quite short, dynamics model are often quite inaccurate towards the end of the prediction horizon anyways due to the compounding nature of prediction error along trajectories. Thus MPC implementations often opt to for a shorter optimization horizon to allow for quick reoptimization and real-time correction for modeling errors.

Reinforcement Learning

Reinforcement learning (RL) has recently gained popularity as a data-driven approach to controlling highly complex robotic systems. Within the large body of work in RL, there are predominantly two categories of approaches: model-based and model-free method, each with their own merits.

In the model-based reinforcement learning paradigm, training data collected on the real-world system is used to train a model to approximate the system's dynamics [38, 39]. This dynamics model is then used for decision making using a variety of approaches, including the hierarchical planning and controls framework and model predictive control framework described earlier. In contrast, model-free reinforcement learning aims to directly optimize a policy that maps robot states or observations to actions by observing results of policy trials on the system [40, 41]. Both model-based and model-free approaches have their advantages and disadvantages.

The model-based reinforcement learning approach is often more practical for real-world systems where safety is a critical concern. Firstly, model-based approaches are known for their higher sample efficiency, which is especially invaluable for real-world systems operating in unstructured environments. Collecting training data through potentially risky trials can be time consuming and dangerous to hardware and bystanders. Secondly, model-based frameworks tend to be more interpretable and predictable compared to their model-free counterparts. The use of an explicit model of the system's dynamics allows for a better understanding of how and why certain decisions are made. This transparency is crucial in safety-critical applications, as it facilitates easier validation and verification of the system's behavior, ensuring that it adheres to safety constraints. Lastly, model-based methods are generally more adaptable to new tasks and environments. Since the learned model aims to represent the underlying system dynamics, it can more easily be applied to different scenarios.

In contrast, model-free methods offer significant advantages in scenarios where creating

an accurate model of the environment is exceptionally difficult or infeasible. For example, when dealing with high-dimensional sensory inputs such as RGB images or LiDAR point clouds [42, 43]. Furthermore, their approach of directly mapping observations to actions allow for rapid, reflex like responses, as opposed to model-based methods that require an additional decision-making layer that leverages the model.

In this dissertation, we mainly follow the model-based reinforcement learning paradigm, although we discuss in Chapter 6 how a model-free policy could be used in conjunction with our navigation framework.

1.2 Standing Challenges

While there have been significant advancements in the field of autonomous off-road navigation, current autonomous control systems still struggle at long-horizon dynamics-aware decision making in unstructured rough terrain environments. A large component of this standing challenge stems from the inability to perfectly model the system's dynamics.

Modeling the dynamics of off-road vehicles is particularly challenging due to the unpredictable nature of such environments coupled with complexity of robotic hardware. Traditional analytical methods for vehicle dynamics modeling are often labor-intensive, require extensive domain expertise, and are prone to errors. Accurate modeling of complex interactions between robots and terrain, such as soil flow or sinkage, necessitates the use of sophisticated terramechanics models [44–46]. Simplifications are often made to make dynamics modeling more tractable [47], with one extreme but common simplified model being the kinematic bicycle model [26]. These simplified models, while easier to create and compute, fail to capture the full intricacies of off-road vehicle dynamics. Regardless of the approach, these analytical models require careful calibration of model parameters to align model predictions to the actual behavior of the system.

In contrast to analytical modeling approaches, data-driven approaches have also been

explored for learning vehicle dynamics [15, 48, 49]. These methods involve collecting data from the real-world system to train highly parameterized models, such as neural networks. Compared to analytical modeling approaches, data-driven methods offer a more automated approach to modeling system dynamics that are less reliant on domain expertise.

However, the efficacy of data-driven modeling approaches is closely tied to the volume and quality of training data available. Models trained on limited or non-diverse datasets may encounter difficulties with out-of-distribution predictions, leading to reduced accuracy in scenarios not adequately represented in the training data. Collecting the comprehensive dataset necessary to train accurate dynamics models is often impractical, especially for real-world systems operating in unstructured environments. This data collection process is not only time consuming, but also poses safety risks, as implementing untested policies during data collection can lead to the robot becoming stuck or damaged. This necessitates constant supervision, with a readiness to intervene, reset, or repair the robot in case of failures. While risk mitigation and automation of the training are feasible in controlled environments, such as indoor environments or simulations [50], doing so in unstructured environments, such as rough terrain environments, is difficult.

This dissertation aims to improve long-horizon decision making for rough terrain navigation by answering two key questions:

Can leveraging low quality simulation data provide insights into real-world vehicle dynamics and reduce the amount of real-world data needed?

Given the practical constraints of collecting training data on real-world systems, simulation often emerges as appealing alternative. However, substituting real-world data with simulation data comes with inherent limitation. Simulated system dynamics inevitably differ from real-world dynamics, a problem known as the "reality gap" [51]. Consequently a model trained on simulation data may accurately approximate the simulated system, but fail to accurately model real-world dynamics.

Despite this inevitable reality gap, simulation data can still provides valuable insights into the general behavior of the real world system. Could the coarse insights drawn from simulation lead to sufficient, albiet possibly suboptimal, control of the robot? Furthermore, could insights drawn from simulation be refined online using real-world data to increase performance?

Besides leveraging simulation, another promising approach towards better rough terrain navigation is to make decision making robust towards modeling inaccuracies, leading to our second key question:

Can quantifying uncertainty in an imperfect dynamics model improve long-horizon decision making?

Dynamics models are inevitably imperfect. Besides the previously mentioned challenges of analytical and data-driven modeling approaches, robot dynamics may be inherently stochastic due to partial observability and unpredictable environmental factors. Furthermore, their dynamics may change stemming from variations in terrain types encountered or changes to the robot's hardware. Consequently, unless dynamics models are frequently recalibrated or retrained, they may become inaccurate over time. This inherent imperfection in dynamic models results in predictions that inevitably contain some degree of error.

Over confidence in incorrect model predictions can result in driving strategies that, despite having high predicted performance, fail in the real world. Ideally, decision making should recognize potential modeling inaccuracies and choose driving strategies that are robust towards potential prediction uncertainty. However, can uncertainty in model predictions be quantified?

Furthermore, handling prediction uncertainty can be particularly challenging when dealing with long-horizon decision making. This is especially true under divergent dynamics, as is common for rough terrain traversal, as small amounts of uncertainty can amplify quickly over longer horizons. For example, a small deviation in the robot's state when navigat-

ing a steep edge can result in drastically different outcomes. Can decision making choose non-myopic driving strategies that are robust towards uncertainty despite their divergent nature?

1.3 Dissertation Outline

The methodology throughout this dissertation centers on a data-driven approach to modeling, planning, and control. Specifically, we focus on the model-based reinforcement learning (MBRL) paradigm [38], where a predictive dynamics model is trained using real-world data, then used for decision making.

In **Chapter 2**, we focus on long-horizon decision making under uncertain learned dynamics, addressing the second question posed above: can long-horizon decision making leverage imprecise dynamics models? We provide an in-depth discussion of our MBRL framework for non-myopic rough terrain navigation, first presented in [52]. In this framework, we train a terrain-aware dynamics model to capture the vehicle's dynamics as it drives over uneven terrain. This model is trained to capture both aleatoric uncertainty, which refers to the inherent stochasticity in the system, and epistemic uncertainty, which stems from having insufficient training data. Capturing epistemic uncertainty can be especially useful in situations where training data is limited or poorly distributed, as in our experiments where training data was collected offline through short manual teleoperation trials. We also introduce the closed-loop divergence constraint that facilitates long horizon trajectory predictions while providing insight into when modelling errors are potentially problematic for the low-level trajectory tracker, i.e. predictions are not trustworthy. We show that constraining optimization solutions to have low closed-loop divergence prevents the optimizer from exploiting modeling errors and results in solution trajectories that the robot can track effectively. We demonstrate this framework's non-myopic and robust rough terrain navigation capabilities in simulation and the real world.

Chapter 3 serves as our initial exploratory work addressing the first question: can simulation data reduce the need for real-world data? In this chapter, we delve into a framework for accelerating dynamics model training for a target system by leveraging large amounts of training data collected on a proxy system with similar, but different, dynamics. This framework was first presented in [53]. In our experiments, we show that this approach results in more accurate target system dynamics models in low-data regimes when compared to dynamics models trained using solely data from the target system, but not from the proxy system. This result suggests that data collected in simulation may help reduce the requirement for real-world training data even though simulated dynamics may not perfectly mirror real-world dynamics.

In **Chapter 4**, we explore another approach to leveraging cheap simulation data which we believe to be superior and recommend as the focal point for readers. This approach builds upon the idea of leveraging simulation and also the idea of capturing modeling uncertainty. This approach, first presented in [54], leverages simulation data collected on a large set of simulated systems, each with the same morphology but different dynamics, to train a dynamics model. This dynamics model can quickly adapt to any new target system of the given morphology by extracting information from any observations available from the target system. This model also captures prediction uncertainty stemming from inherent system stochasticity as well as from insufficient target system observations. By assuming knowledge of the target system's morphology, this framework transforms epistemic uncertainty from uncertainty in neural network parameters to uncertainty about the target system's specifics within the given morphology. Using this model within an uncertainty-aware decision making process allows the driving policy to align with its current understanding of the vehicle's dynamics and capabilities. In situations where real-world data is extremely limited, the robot can still be safely controlled, albeit at a sacrifice of performance. As more real-world data becomes available, the dynamics model and resulting policy becomes more tailored towards the target system, resulting in higher performance. We demonstrate this approach

on different real-world systems for the simpler task of driving along a flat off-road track.

In **Chapter 5**, we discuss a robust and adaptive rough terrain navigation framework that serves as the culmination of this dissertation and builds on previously discussed concepts. This framework leverages a large range of simulated systems to train a terrain-aware dynamics model that can quickly adapt to any new system while capturing uncertainty in predictions. We adapt the previously discussed closed-loop divergence constraint to be used within an RRT planning framework [55]. We show in a couple of toy simulated scenarios the capabilities of this framework. Most notably, the navigation framework results in safer and more conservative strategies when no data is available from the target system, but quickly adapts the driving strategy to the particular system's capabilities after giving the model adaptation data from just one demonstration.

Finally, in **Chapter 6**, we summarize the key findings from this dissertation that answer the two key questions posed above. Furthermore, we discuss how concepts from this dissertation are applicable to other planning and controls frameworks, and discuss potential future extensions.

Beyond this dissertation on rough terrain navigation, I have also pursued other projects during my doctorate, such as developing a method for localizing unexpected contact for compliant robots (demonstrated on a legged robot and a manipulation system) [56], and designing an environmental sampling robot for surveying large contaminated areas [57].

Chapter 2

Robust Non-Myopic Decision Making with Learned Dynamics

2.1 Introduction

Autonomous navigation in rough terrain still presents a significant challenge to wheeled robots as it requires non-myopic decision making under complex, difficult-to-model dynamics, stemming from unpredictable robot-terrain interactions [58]. Furthermore, the discrete nature of contact between the robot and obstacles in the terrain results in highly divergent dynamics that causes any modeling errors to compound and propagate poorly along trajectories.

Methods that rely on accurate handcrafted dynamics models for decision making, e.g. [59–61], often perform poorly when predictions contradict reality. Some methods aim to combat this issue by formulating control policies that are robust against model uncertainty [62–65]. Others abstract the robot's dynamics by using traversability maps to provide heuristics for how difficult different regions are to traverse [13, 66–70]. However, these approaches are limited to coarse approximations of traversability and don't consider dynamic interactions.

Alternatively, reinforcement learning can be used for rough terrain traversal. Some meth-

ods use model-free reinforcement learning (MFRL) [71–73], where control policies are directly optimized. Others use model-based reinforcement learning (MBRL) [74], where predictive dynamics models are learned and then used for decision making. Due to their higher sample efficiency, MBRL methods are generally more practical for real world systems where training data is limited and expensive to collect. 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 [75, 76], though characterizing prediction uncertainty for rough terrain navigation is challenging since the divergent dynamics cause model uncertainty to propagate poorly over long prediction horizons.

In this chapter, we delve into our Model-Based Reinforcement Learning (MBRL) framework, first presented in [52]. This framework, summarized in Section 2.3 and Figure 2.1, aims to address the second question posed in Chapter 1, "Can quantifying uncertainty in an imperfect dynamics model improve long-horizon decision making?"

Our algorithm's unique method of handling uncertainty allows the robot to more accurately optimize longer trajectories. Like previous methods [75, 76], our method performs trajectory optimization using learned probabilistic dynamics models to predict trajectories and characterize prediction uncertainty. However, we train the probabilistic dynamics model using a multistep loss that considers how model uncertainty propagates along trajectories (Section 2.4). We perform closed-loop trajectory prediction, where the capability of the robot's trajectory tracking controller is considered during trajectory prediction (Section 2.5). The use of a multi-step loss and closed-loop trajectory prediction decreases long horizon trajectory prediction uncertainty and allows the robot to be more selective during optimization. Our algorithm uses constrained optimization to avoid trajectories that result in high divergence (Section 2.6), similar to Convergent Planning [65]. Through simulation and hardware experiments, we show that these improvements allow the robot to find non-myopic trajectories with high prediction accuracy and precision, increasing the likelihood of

successful navigation (Section 2.7).

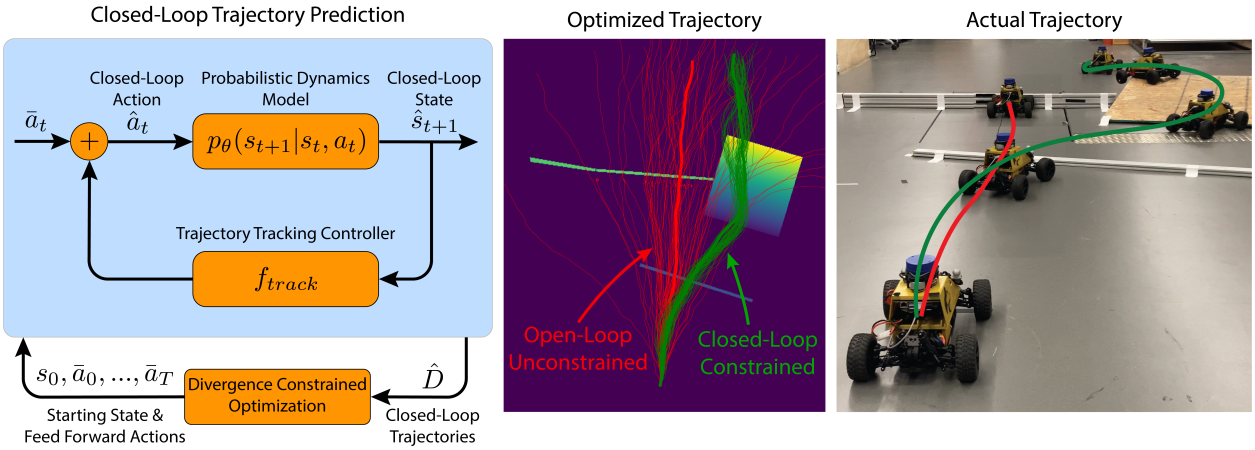


Figure 2.1: Left: An overview of the method. The probabilistic dynamics model is trained using a multistep loss that considers how uncertainty propagates. The trajectory tracking controller is used to predict a distribution of closed-loop trajectories. The divergence constrained optimizer is used to 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.

2.2 Related Works

Probabilistic Ensembles with Trajectory Sampling (PETS) [75] aims to make MBRL more robust to modeling errors by characterizing uncertainty with probabilistic ensemble models and propogating uncertainty with trajectory sampling. PETS optimizes trajectories based on expected rewards and performs well on benchmark tasks [77]. However, using PETS on rough terrain is challenging due to the high variance in predictions, which makes it difficult to estimate expected reward. Furthermore, high reward expectation with high variance does not mean the robot will actually perform well.

Kahn et al. [76] use uncertainty in a model-based fashion for collision avoidance in navigation. They train an ensemble of models (via variational dropout [78]) 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 uncer-

tainty 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. [79] 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.

2.3 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 true, but unknown, 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 training data collected offline. We denote the training dataset as $\mathcal{D} = \{d_0, ..., d_M\}$, where $d_i = (s_0, .., s_T, a_0, ..., a_{T-1})$ is a state-action trajectory. This neural network model is terrain-aware, as it's predictions are conditioned on a robot-centric terrain height map and captures the effects that uneven terrain has on the robot. We train the model to capture both aleatoric and epistemic uncertainty using a multistep training loss that considers how model uncertainty propagates along trajectories (Section. 2.4). The trained probabilistic model is used to predict a distribution of the robot's trajectory under the influence of a tracking controller (Section 2.5). This allows for reduction in prediction variance when predicting long horizon trajectories. Finally, a low cost trajectory to the goal is generated using constrained optimization. The solution is constrained based

on the probabilistic model to avoid steps where the vehicle's dynamics under the trajectory tracking controller diverges significantly (Section 2.6). This closed-loop divergence constraint ensures that potential prediction inaccuracies of the solution trajectory are non-problematic for the low level tracker, thus preventing the optimizer from exploiting modeling errors. A visual summary of our approach is shown in Figure 2.1.

2.4 Probabilistic Terrain-Aware Dynamics Models

We propose a method for training a probabilistic terrain-aware dynamics model to predict long-horizon robot trajectories when driving over uneven terrain. Similar to [75], 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.

2.4.1 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 [80] 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 [75] 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, [81, 82] 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 [81, 82] 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 con-

ditioned 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:

$$\begin{aligned} p_\theta(s_1|A) &= p_\theta(s_1|s_0, a_0) \\ p_\theta(s_2|A) &= \mathbb{E}_{\hat{s}_1 \sim p_\theta(s_1|A)}\big[p_\theta(s_2|\hat{s}_1, a_0)\big] \\ &\vdots \\ p_\theta(s_t|A) &= \mathbb{E}_{\hat{s}_{t-1} \sim p_\theta(s_{t-1}|A)}\big[p_\theta(s_t|\hat{s}_{t-1}, a_{t-1})\big] \end{aligned} \tag{2.1}$$

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 $N \in \mathbb{N}$,

$$\mathcal{L}^N_{ms}(s_t, A, \theta) := -\log \frac{1}{N} \sum_{i=1}^{N} p_\theta(s_t|\hat{s}^i_{t-1}, a_{t-1}) \tag{2.2}$$

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:

$$\mathbb{E}\big[\mathcal{L}^1_{ms}(s_t, A, \theta)\big] \geq \mathbb{E}\big[\mathcal{L}^2_{ms}(s_t, A, \theta)\big] \geq ... \geq \mathcal{L}_{ms}(s_t, A, \theta) \tag{2.3}$$

As such, minimizing $\mathcal{L}^N_{ms}$ using any stochastic optimization methods such as Stochastic Gradient Descent (SGD) [83] 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.

2.4.2 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 [75], while other methods such as Bayesian neural networks [84] or hypernetworks [85, 86] parameterize models via uncertainty in the weights of the neural network.

In our work, we use variational dropout to characterize epistemic uncertainty [78, 87]. 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 Section 2.4.1. We then update Θ to minimize training loss. The complete model training procedure is summarized in Algorithm 1.

2.4.3 Terrain-Aware Neural Network Architecture

When navigating over uneven terrain, the shape of the terrain can have a drastic impact on the robot's state-transitions. For example, a robot attempting to drive forward over an elevated step may not make forward progress. In contrast, a robot can easily drive forward to descend steps. We inform neural network state-transition predictions of the terrain by assuming that the robot's state s_t includes a $2.5D$ heightmap of the terrain generated using existing terrain mapping techniques such as [88].

In our implementation, the neural network dynamics model (p_θ), consisted of a Convolutional Neural Network (CNN) [89] and a Long-Short Term Memory (LSTM) [90] network. The CNN was chosen due to the image-like form factor of the terrain height maps, and the LSTM was chosen for the time sequential nature of state-transition predictions along trajectories. For sample efficiency, we used a robot-centric form for all neural network inputs and outputs, i.e. a form that is independent of the robot's $x - y$ position and heading angle. This choice was motivated by the fact that the robot's dynamics should not depend on these

variables. Eliminating these variables from the inputs and outputs eliminates the need to collect training data from different positions and orientations in the world.

The robot-centric heightmap input, denoted as m_t^l, was cropped from the world terrain heightmap such that the cropping was centered around the robot's $x-y$ position and aligned with the robot's heading. This robot-centric heightmap was processed using the CNN. The output of the CNN was concatenated with a robot-centric form of the robot's state, denoted as s_t^l and fed into the LSTM. The output of the LSTM was a multivariate Gaussian distribution of the robot's state-transition. Note that both m_t^l and s_t^l can be generated directly from s_t. During prediction, the robot's state s_t is used to generate the robot-centric neural network inputs m_t^l and s_t^l. Given m_t^l, s_t^l, and a_t, the neural network predicts the robot's state-transition distribution. A state-transition can be sampled from this distribution to transition the robot's state s_t to s_{t+1}. This prediction process can be chained for trajectory prediction.

For our simulation experiments, robot state s_t consisted of a $SE(3)$ pose and R^6 twist velocity vector. The robot-centric state s_t^l consisted of R^3 tilt vector denoting the direction of gravity relative and the twist velocity vector, both relative to the robot's body frame. State-transition predictions s_t' included the $SE(3)$ pose of the robot at time $t+1$ relative to the robot's body frame at time t. It also included the robot's new twist velocity vector. Since the $SE(3)$ pose included the height of the robot, we adjusted the height values of the robot-centric heightmap m_t^l to be relative to the height of the robot's body frame.

For our real-world experiments, we chose to represent the robot's state as a $SE(2)$ pose and prior state-transition vector, which was the robot's new $SE(2)$ pose relative to the prior. As such, the robot-centric state s_t^l only consisted of the prior state-transition vector. This choice was motivated by two factors. First, the odometry method used resulted in accurate positional estimates, but noisy velocity estimates. Transitioning from a velocity vector to a prior state-transition vector input eliminated the reliance on the noisy velocity measurement signals, while still providing context about the robot's velocity. Second, moving

from $SE(3)$ to $SE(2)$ reduced the amount of training data needed due to the lower capacity state description.

In theory, this lower capacity state description reduces the model's ability to precisely capture the robot's dynamics. In practice however, the effects of moving from a $SE(3)$ to $SE(2)$ prediction framework was negligible. Furthermore, this simplified representation can be viewed as a partially observable representation which results in probabilistic system dynamics with higher aleatoric uncertainty. This is non-problematic for the uncertainty aware decision framework in this chapter as it is meant to handle aleatoric uncertainty.

Table 2.2 shows the neural network sizes used for simulation experiments and Table 2.1 shows the neural network sizes for the real-world experiments.

Table 2.1: The network architecture for real world

Layer	Input	Input Dim	Activation	Output	Output Dim
$CNN_{1,8}$	Heightmap	$1 \times 16 \times 16$	ReLU	Hmap feat1	$8 \times 8 \times 8$
Reshape	Hmap feat1	$8 \times 8 \times 8$	None	Hmap feat2	512
$FC_{517,2048}$	Hmap feat2 + Rob-Cen State + Action	517	ReLU	feat1	2048
$LSTM_{2048,256}$	feat1	2048		feat2	256
$FC_{256,256}$	feat2	256	ReLU	feat3	256
$FC_{256,6}$	feat3	256	None	pred. μ, log-σ	6

Table 2.2: The network architecture for simulation

Layer	Input	Input Dim	Activation	Output	Output Dim
$CNN_{1,8}$	Heightmap	$1 \times 64 \times 64$	ReLU	Hmap feat1	$8 \times 32 \times 32$
$CNN_{8,4}$	Hmap feat1	$8 \times 32 \times 32$	ReLU	Hmap feat2	$4 \times 16 \times 16$
Reshape	Hmap feat2	$4 \times 16 \times 16$	None	Hmap feat3	1024
$FC_{1035,2048}$	Hmap feat3 + Rob-Cen State + Action	1035	ReLU	feat1	2048
$LSTM_{2048,256}$	feat1	2048		feat2	256
$FC_{256,256}$	feat2	256	ReLU	feat3	256
$FC_{256,13}$	feat3	256	None	pred. μ, log-σ	13

2.5 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 [75] 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:

$$a_t = \bar{a}_t + f_{track}(s_0, ..., s_t, \bar{d}), \tag{2.4}$$

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.

Our method first predicts a nominal reference trajectory given a starting state and feed forward actions,

$$\bar{s}_{t+1} = \mathbb{E}\big[s_{t+1} \sim p_\Theta(s_{t+1}|\bar{s}_t, \bar{a}_t)\big]. \tag{2.5}$$

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 2.4. 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 $\hat{d}^i = (\hat{s}_0^i, ..., \hat{s}_T^i, \hat{a}_0^i, ..., \hat{a}_{T-1}^i)$ is a particle's state-action trajectory. The procedure for predicting closed-loop trajectories is summarized in Algorithm 2.

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**
    $\hat{s}_t^i \sim p_{\theta^i}(s_t|\hat{s}_{t-1}^i, \hat{a}_{t-1}^i)]$
    $\hat{a}_t^i \leftarrow \bar{a}_t + f_{track}(\hat{s}_0^i, ..., \hat{s}_t^i, \bar{d})$

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 [79] prevented the optimizer from exploiting modeling errors by penalizing trajectories with high prediction uncertainty. This approach may be problematic as the penalty opposes 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 diverge too much from the reference. Values for U are dependant on the trajectory tracker used. In our experiments, we selected U based on the tracker's observed capabilities.

Given a nominal trajectory $\bar{d}$ and predicted closed-loop trajectory distribution $\hat{D}$, we define divergence as follows:

$$u(\bar{d}, \hat{D}) = \max_t \frac{1}{I} \sum_{i=1}^{I} \|\hat{s}_t^i - \bar{s}_t\|_2^2. \tag{2.6}$$

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

$$\min_{\bar{a}_0,\ldots,\bar{a}_{T-1}} \quad C(\bar{d}) \qquad \text{s.t.} \quad u(\bar{d}, \hat{D}) < U \tag{2.7}$$

We use an Augmented Lagrangian optimizer to solve (2.7), 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. While we chose to use an Augmented Lagrangian optimizer here, this choice of optimizer is arbitrary and any constrained optimization framework could be used. In

Chapter 5, we even reuse the divergence constraint within a Rapidly-exploring Random Trees (RRT) [55] planning framework.

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

The divergence metric defined in (2.6) serves as an indication of when potential modeling errors may be problematic for the low level trajectory tracker. While we chose a metric based on mean squared error between the predicted trajectory distribution and the nominal trajectory, this choice is non-unique. Other distance measures between the predicted and nominal trajectories could also serve the same purpose. One possible choice is the dynamic time warping metric [91], which deforms time between sequences to handle temporal shifts. However, one important advantage of using mean squared error (MSE) over dynamic time warping (DTW) that should be taken into consideration is that MSE inherently increases as the variance in the predictions increases. This characteristic of MSE makes it particularly useful for indicating when trajectory predictions become unreliable. In some situations, using DTW to compare particle trajectory predictions against the nominal trajectory may result in low distance measure although the predicted distribution has high variance and should not be trusted. An example of such a situation can be seen in Figure 2.6a. In this example, the robot is attempting to drive over a nontraversable obstacle. The nominal prediction and a few of the particles predictions incorrectly predict that the robot can easily drive over the obstacle. Other particles in the distribution predict that the robot initially gets stuck on the obstacle but then eventually makes it over the obstacle after some time. While the high variance in the predicted distribution results in high divergence, the DTW metric remains

low.

2.7 Experiments

We compare our navigation method against others in simulation [1] and in the real world. We used PyBullet [92] for our simulation environment and a MuSHR robot [93] for our real world experiment.

2.7.1 Model Training and Prediction

For both the simulation and real world system, we trained a probabilistic terrain-aware dynamics model using training data collected offline, using either random actions or manual controls. We trained using our proposed multistep loss, $\mathcal{L}_{ms}^{128}$, with Algorithm 1 using 5 step prediction horizons.

For the simulation experiments, training data was collected by randomly driving the robot using Ornstein–Uhlenbeck noise in random environments. The terrain in each environment was randomly generated, and had both discrete block-like features and continuously varying noise features. For the block-like features, we first segmented the environments into cell regions using Voronoi diagrams with random sites. The height of each cell was randomized using Perlin noise, where the cell height corresponded to the Perlin noise height at the site point. We applied Gaussian blur to smooth the edges of each cell. Finally, we added continuous variations to the terrain using Perlin Noise. A few examples of the randomly generated environments can be seen in Figure 2.4. We used ground truth pose, velocity, and terrain height map measurements for both model training and online decision making.

For the real world experiments, training data was collected using human selected actions with added Gaussian noise in one environment. The training environment consisted of the same obstacles as the test environment, shown in Figure 2.1, but with different arrangements.

[1] Simulation code can be found at https://github.com/robomechanics/dco_rough_terrain_nav.

As this work is not focused on state estimation or terrain mapping, we chose to use ground truth vehicle odometry and world terrain height maps, measured using a motion capture system, for both training and online decision making. For terrain mapping, we measured the locations of and shape of all obstacles using the motion capture system, and manually recreated a height map of the environment.

Trajectory Tracker

In our implementation, we chose a simple Proportional-Derivative (PD) controller as the trajectory tracker for the Ackermann steered robot. This choice was mainly motivated by the simplistic form of $f_{track}(s_0, ..., s_t, \bar{d})$ takes that makes for easier computation during optimization. In Chapter 5, we describe a modification to the closed-loop prediction framework that allows for more sophisticated trajectory tracking policies, such as linear-quadratic regulator (LQR) controller or even model-free reinforcement learning trajectory tracking policies [94–96].

Given the desired state trajectory and feed forward actions $\bar{d} = (\bar{s}_0, ..., \bar{s}_T, \bar{a}_0, ..., \bar{a}_{T-1})$, the feed back actions executed by the PD trajectory tracking controller takes the form:

$$a_t = \bar{a}_t + f_{track}(s_0, ..., s_t, \bar{d}) \tag{2.8}$$

$$f_{track}(s_0, ..., s_t, \bar{d}) = K_p e_t + K_d(e_t - e_{t-1}), \tag{2.9}$$

where K_p and K_d are gain terms, and $e_t \in \mathbb{R}^2$ is the error vector. The error vector e_t is defined as the $x - y$ position of the desired state $\bar{s}_t$ relative to the actual state s_t of the vehicle. An illustration of this vector is shown in Figure 2.2.

At each time step, the PD tracker calculates a corrective feedback action to minimize the error between the true robot state s_t and the desired state $\bar{s}_t$. For our systems, an action $a = [1, 0]^T$ drives the robot forward in the $+x$ direction, and an action of $a = [0, 1]^T$ will steer the robot to turn counterclockwise in the $+\theta_z$ direction. An approximate action of

$a = [1, 1]^T$ is illustrated in Figure 2.2.

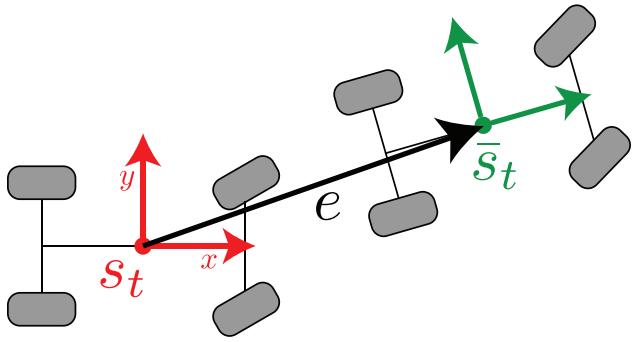


Figure 2.2: Definition of PD tracker's error vector. The actual state s_t is shown in red. The reference state $\bar{s}_t$ is shown in green. The error vector is shown in black and defined relative to frame s_t.

Trajectory Prediction Accuracy

In simulation, we evaluated the effects of 1) using a multistep loss during training, and 2) making closed-loop predictions on trajectory prediction accuracy. We trained one model using a single-step log-likelihood loss, and another model using our proposed multistep loss, $\mathcal{L}_{ms}^{128}$, with Algorithm 1. Using each model, we predicted 32 time-step long trajectories using a naive open-loop method and the proposed closed-loop (tracked) method. For the open-loop method, a sequence random actions were selected and executed on the robot with no low-level tracker. The trained models were used to naively predict the trajectory of the robot executing the actions with no feedback controller. We repeated this process for the closed-loop method, performing trajectory predictions using Algorithm 2. However, when executing the action sequences on the robot, we included the feedback action term from the low-level trajectory tracker following the nominal trajectory, $\bar{s}_0, ..., \bar{s}_T$, calculated during the closed-loop trajectory prediction process.

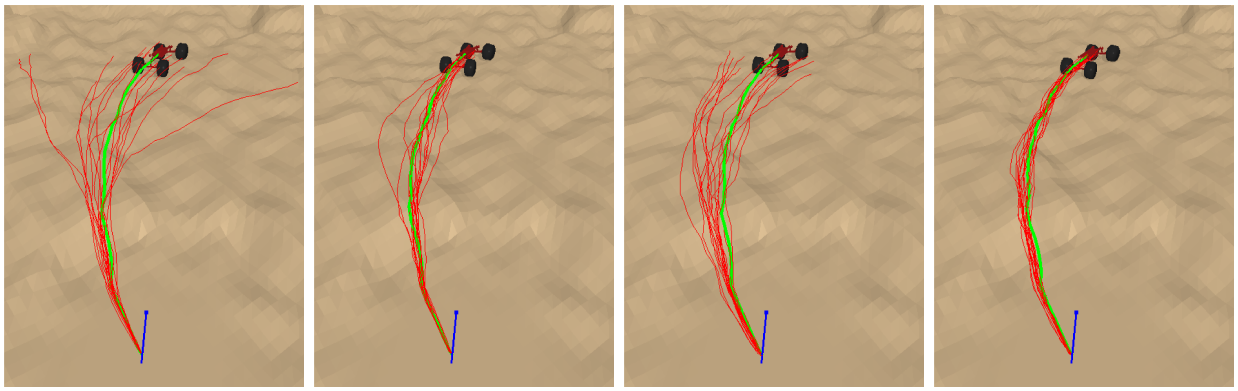
For all methods, we predicted the probability distribution of the robot's final state and reported the log-likelihood of the true final state. We predicted the final probability distribution as a Gaussian mixture model by propagating 512 particles using the trained models (each uniformly weighted mixture of the Gaussian mixture model corresponds to the final

Gaussian distribution prediction of a particle).

In total, we ran 100 trials. The mean and variance of the log-likelihood metric for all methods are listed in Table 2.3. The true trajectories and some particle trajectories for one trial are shown in Figure 2.3. From these results, we find that both the multistep training loss and closed-loop prediction framework aids in predictions that more accurately capture uncertainty.

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

Table 2.3: Mean and standard deviation of prediction log-likelihood for 100 trials.



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

Figure 2.3: Predicted robot trajectories using different prediction methods The predicted trajectories are shown in red, and the true trajectories are shown in green.

2.7.2 Rough Terrain Navigation

We compared the performance of three different trajectory optimization frameworks:

1. an optimizer that only minimizes the goal cost, $C(\bar{d})$
2. an optimizer that adds a scaled divergence penalty
3. the proposed divergence constrained optimizer (Algorithm 3)

We used final squared distance to the goal as our optimization cost. We evaluate the performance of the different methods by reporting success rate, defined as reaching a certain distance from the goal, and average final squared distance to the goal. The number of optimization gradient steps were kept consistent between methods. We tracked optimized trajectories using a simple PD controller (detailed definition of f_{track} provided in Section 2.7.1). The effects of the tracker were considering during optimization using closed-loop predictions (Algorithm 2). We treated the threshold U for constrained optimization as a hyperparameter, which we adjusted based on the observed abilities of the tracker. All optimization methods used the same trained model.

Navigating Over Simulated Terrain

We compared the performance of different navigation methods for the task of driving over randomly generated terrain with large obstacles to a goal $7m$ away. Besides the three aforementioned optimization frameworks, we also included an A* path planning method with tracking using the traverssability mapping method from [97], which considered local terrain smoothness, slope, and curvature. We also evaluated the sensitivity of the optimizers to trajectory prediction accuracy by not performing closed-loop predictions. For 250 trials, Table 2.4 shows each method's success rate, defined as ending within $0.5m$ of the goal. Table 2.5 shows the actual cost and standard deviation for each method with tracking.

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

Table 2.4: Random terrain results. Success percentage over fifty random terrains, using the same terrain set for all methods. The A* planner does not provide actions, so tracking was required.

Overall, we found that the constrained optimization has the highest success rate and lowest final cost. Furthermore, seeding the constrained optimization with the planner resulted in an even higher success rate (88%), suggesting that future work should address issues with

	A* Planner	No Uncertainty	Penalty	Constraint
Cost (Squared distance to goal)	2.0892	1.8792	0.9561	**0.9109**
Divergence	0.2039	0.2350	**0.1220**	0.1529

Table 2.5: Results of random terrain testing. Despite having higher divergence than the penalty method, the constrained method has lower final cost.

local minima. As expected, the penalty method has the lowest divergence since it directly minimizes it, however this did not translate to higher success rate. We also found that prediction accuracy was especially important to uncertainty aware optimization methods. Not including tracking during prediction significantly decreased the success rate of both the penalty and constrained optimization methods.

Figure 2.4 shows some example trials. When divergence is not considered, the optimizer tends to exploit prediction error and finds risky solutions with high divergence. When executed, the robot tends to collide with obstacles and rolls over. Conversely, the penalty method tends to find overly conservative trajectories since the optimizer minimizes divergence at the expense of task cost. The trajectories have low divergence, but does not reach the goal. The divergence constrained method strikes a middle ground finding safe low-divergence trajectories to the goal. This behavior was exhibited throughout most of our experiments.

Figure 2.5 shows another illustrative example in a simple ramp environment. When divergence is not considered, the optimizer exploits prediction errors and finds a short, but highly divergent trajectory that is infeasible. During execution, the robot slides off the ramp due to insufficient friction. Again, penalizing divergence leads to an overly conservative trajectory since the optimizer minimizes divergence at the expense of task cost. The divergence constrained optimizer finds a trajectory with low divergence and task cost, allowing the robot to consistently reach the goal.

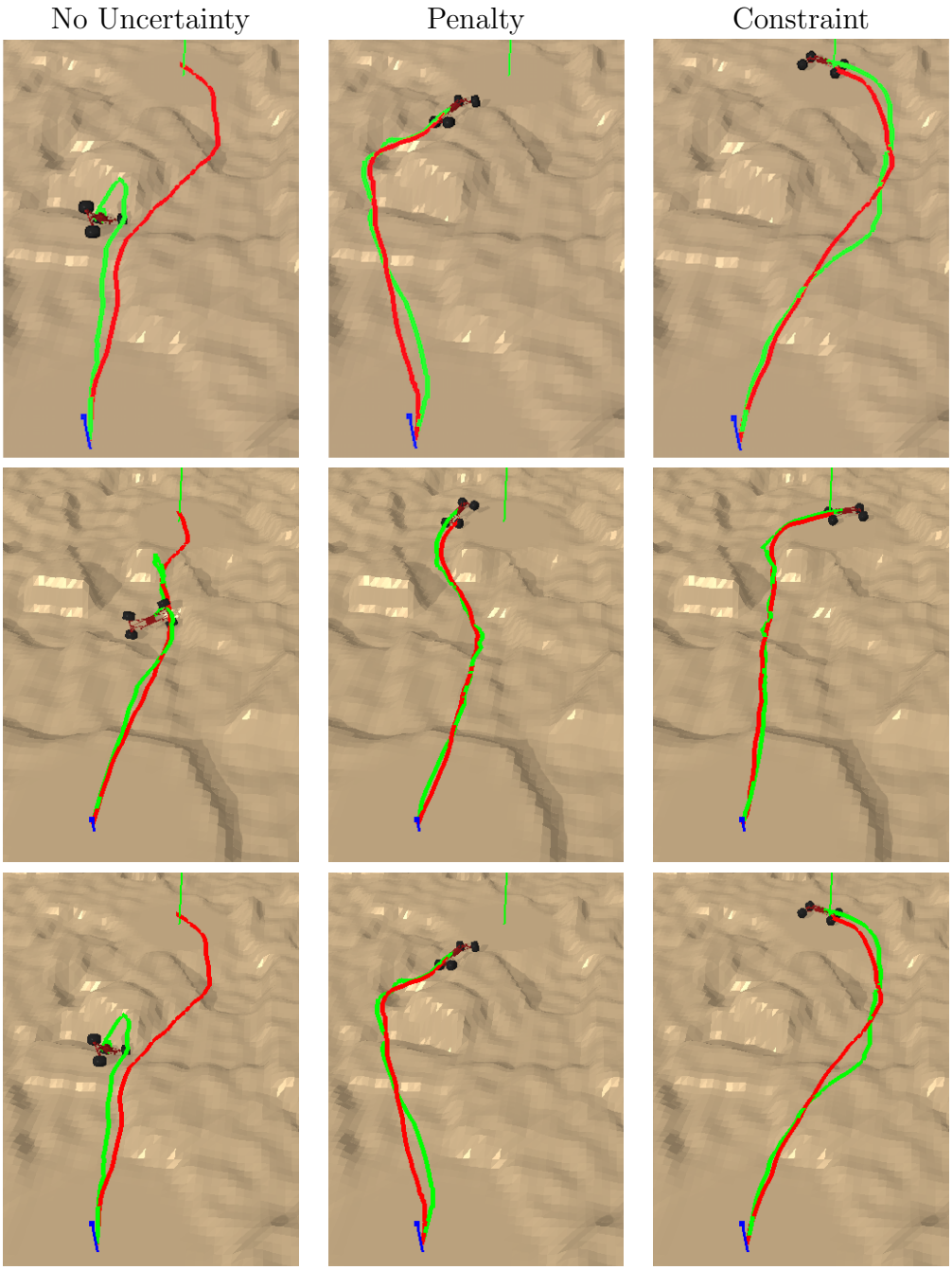
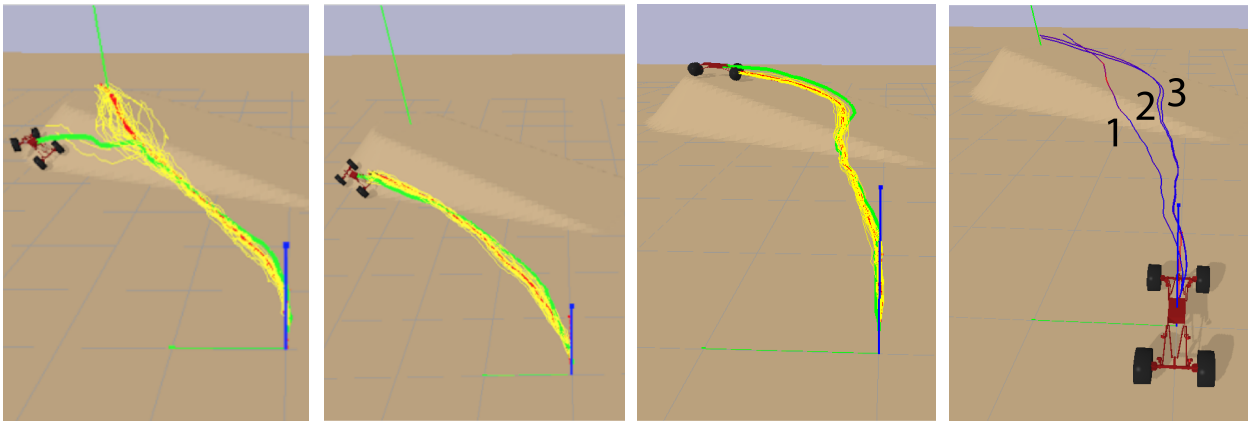


Figure 2.4: Three random simulation trials. Left: optimization without uncertainty. Middle: optimization using closed-loop divergence as a penalty. Right: optimization using closed-loop divergence as a constraint (our method). Predicted nominal solution trajectories are shown in red, and true trajectories are shown in green. The goals are marked with vertical green lines.



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

Figure 2.5: Ramp climbing trial. The optimized, predicted closed-loop, and actual trajectories are shown in red, yellow, and green respectively. The optimization process for (c) is shown in (d), where the divergence constraint is progressively satisfied, with red indicating higher divergence.

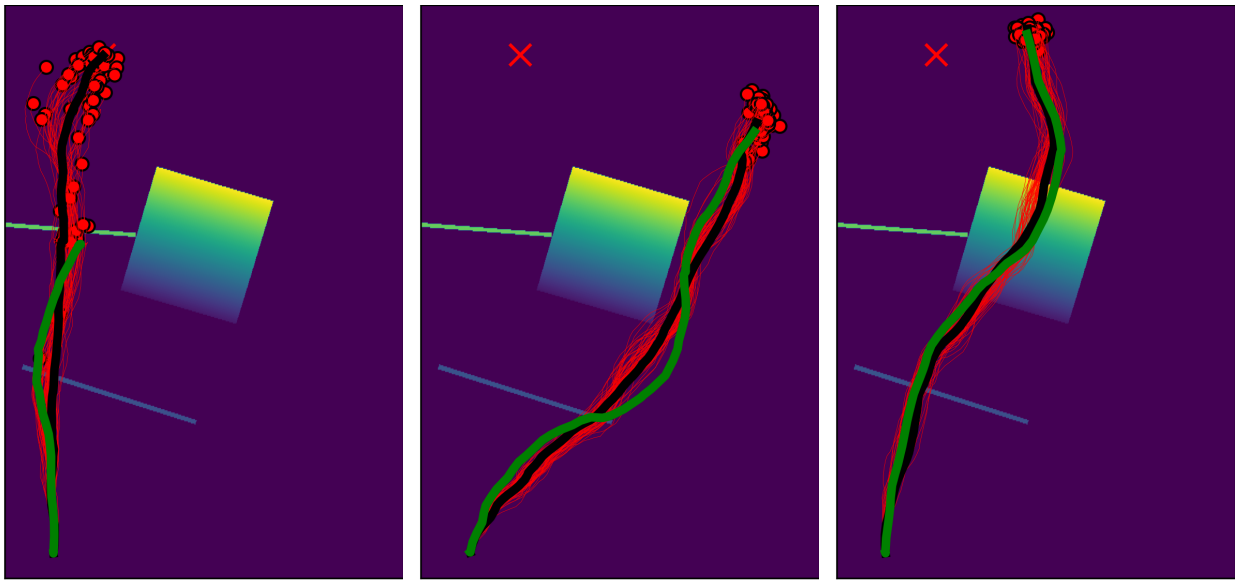
Real World Navigation

We also compared the three optimization frameworks in the real world, Figure 2.6. The robot was tasked with navigating across an obstacle field to a goal $5m$ away. The field consisted of a traversable short obstacle, a non-traversable tall obstacle, and a ramp whose traversability depended on the robot's maneuver. Table 2.6 shows the actual cost and success rate for 10 trials of each of the three optimization frameworks. As in the simulated examples, our cost is squared distance to goal.

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 one 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 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 (Squared dist. to goal)	3.57±1.30	5.11±4.45	**1.68±1.42**
Success Rate	10%	40%	**70%**

Table 2.6: 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.



(a) No Uncertainty (b) Penalty (c) Constraint

Figure 2.6: 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.

2.8 Discussion

Here, we note some advantages of our closed-loop divergence constrained optimization approach observed in our experimental results. We also note some potential future extension that could help make this approach applicable to more systems, some of which are addressed in future chapters of this dissertation.

Strengths of Method

First, the adoption of data-driven terrain-aware dynamics models provides a more streamlined approach to generating higher fidelity models when compared to traditional handcraft-

ing methods. This approach eliminates the need for domain expertise as models are directly generated from data collected on the system. As a resulting benefit, this makes resulting models less prone to modeling errors that may arise from incorrect system assumptions or inadequate system identification. Additionally, the statistical approach to generating these data-driven models naturally captures inherent uncertainty in the system, which is crucial for robust and reliable navigation in unpredictable environments.

Second, the use of a closed-loop prediction framework significantly enhances long horizon decision making capabilities in systems characterized by uncertainty. Figure 2.1 Right, contrasts the predicted trajectory distribution made with an open-loop prediction framework and with a closed-loop prediction framework. When not using a closed-loop prediction framework, the predicted distribution has much higher variance, especially towards the end of the prediction horizon. For any trajectory optimization framework that considers uncertainty, including ours, this high variance in predictions can be problematic. In contrast, the closed-loop framework, by continuously integrating real-time feedback, maintains tighter control over the trajectory predictions, enabling decision making to precisely plan further into the future.

Another key strength of our framework is its ability to prevent trajectory optimization from exploiting modeling inaccuracies, a common challenge in model-based control frameworks. This aspect is particularly crucial in unpredictable environments, such as scenarios involving navigating over unstructured obstacles, where perfect predictions of the robot's dynamics is infeasible. In our framework, the divergence constraint avoids trajectories that are characterized by high prediction uncertainty, stemming either from aleatoric or epistemic uncertainty. In our experiments, we found that our method had a higher success rate compared to the baseline optimization framework that ignored uncertainty and the divergence metric. This was as a result of the baseline method tending to return optimized solutions who's predicted nominal trajectory reached the goal, but could not be faithfully executed on the actual system. In contrast, when the divergence metric was used as a penalty instead of

a constraint, the solutions tended to be faithfully executable by the robot, but were overly conservative and did not reach the goal. Our divergence constraint approach resulted in optimized solutions that reached the goal and could be faithfully executed.

As an added benefit of being robust against prediction errors, our method lends itself well to the offline learning setting, where the agent is unable to interact with the environment during the learning process. In both the real-world experiments and the simulation experiments, training data was collected a prior either through random actions or manual controls. In spite of this, the resulting control policy was still able to navigate through new environments. We tribute this to the epistemic uncertainty quantification by the learned model as well as closed-loop divergence constraint used during trajectory optimization. The divergence constrained optimization naturally avoids actions that result in high epistemic uncertainty. As a result, the robot's policy is biased towards maneuvers that are well supported by the training dataset.

Future Extensions

One notable shortcoming of the approach described in this chapter is the tendency for a gradient descent based optimizer to fall into a local minimum. In our experiments, this was the biggest source of failure for our method, where a lower cost solution that satisfied the constraint existed, but was not found by the optimizer. In our implementation, we alleviated this issue by performing multiple gradient descent optimization descent in parallel, each with different random seeds, and choosing the best final solution. The intuition behind this approach is that at least one of the local minima found will likely be near optimal. However, it does not guarantee that the globally optimal solution will be found. Thus, it is important to acknowledge that this approach does not guarantee that a constraint satisfying trajectory to the goal will be found if one exists.

This local minima issue is not unique to our approach, but is inherent to all optimization based approaches. On the other hand, planning based approach discretize the search space

in order to guarantee that a solution will be found if one exists. In our own simulation experiments, we experimented with seeding divergence constrained optimization with the solution from an A* search algorithm using a traversability mapping cost. This combination showed an improvement over using divergence constrained optimization or planning alone. However, we note that the seed used in this approach can only serve as a heuristic, as the traversability mapping cost only provides a coarse approximation of the dynamic capabilities of the robot. In Chapter 5, we revisit the divergence constraint but adapt it for use within an RRT planning framework. This adaptation suggests that the divergence constraint could also be relevant to other decision making frameworks.

In this chapter, we assumed that the feedback controller used for trajectory tracking takes the form shown in equation 2.4. However, we note that not all feedback controllers take this form. For example, the low-level trajectory tracker could be conditioned on additional observations not captured in the system's state, e.g. RGB images. Alternatively, if the tracker does take the form shown in equation 2.4, there many not be a simple closed form solution mapping states to feedback action. For example, a model-predictive trajectory tracker could be used, which requires optimization to calculate the feedback action term. In such scenarios, determining the feedback action terms iteratively within the closed-loop divergence constrained optimization framework may not be computationally feasible. In Chapter 5, we describe an alternative modeling approach to directly learn the closed-loop behavior of the system given candidate reference trajectories.

Finally, although we assume full state knowledge, we believe that the approach could be extendable to Partially Observable Markov Decision Processes (POMDP). Recent work [98, 99] have formed predictive dynamics models for POMDPs by learning an encoder that maps observations to a latent vector, and learning a latent space dynamics model that predicts how the latent state vector evolves over time. We believe that this could potentially extend our method to situations where the robot does not have complete world terrain height map knowledge, such as when the robot only has observations from a camera.

2.9 Conclusion

In this chapter, we delve into our model-based reinforcement learning framework for rough terrain navigation. This framework addresses the second key question posed in Chapter 1, "Can quantifying uncertainty in an imperfect dynamics model improve long-horizon decision making?"

In this framework, training data is collect on the system offline to train a dynamics model to approximate the robot's complex dynamics when driving over uneven terrain. Given the understanding that dynamics model are inevitably imperfect, this framework also trains the model to capture aleatoric and epsitemic uncertainty in its predictions. We introduce the divergence constraint to prevent decision making from exploiting potential modeling errors. During trajectory optimization, solution trajectories are constrained to have low divergence ensuring that they can be faithfully executed by the low-level trajectory tracking controller. Addressing the issue of poor propagation of prediction uncertainty along longer-horizon trajectories, we perform closed-loop trajectory prediction, where the capabilities of the low-level trajectory tracker is considered during trajectory prediction. This reduces prediction variance along longer-horizon trajectories and facilitates better long-horizon decision making.

We show in our experiments that this enables effective non-myopic navigation despite using an imperfect dynamics model trained with limited training data collected offline. This framework is robust against modeling errors, and can take full advantage of the robot's dynamics and capabilities. As a result, decision making does not simply avoid obstacles, but reason about how the robot should drive over them.

Chapter 3

Accelerating Dynamics Model Training with Cheap Simulation Data

3.1 Introduction

In the previous chapter, we discussed our model-based reinforcement learning framework to driving wheeled robots over uneven terrain. In that framework, machine learning was used to model the robot's dynamics, offering a more streamlined and higher fidelity alternative to traditional modeling approaches.

However, this data-driven approach to modeling dynamics is highly dependant on the quality and quantity of training data available. In situations where training data is insufficient, or has a distribution shift, the learned dynamics model will have inaccurate predictions. When paired with uncertainty-aware decision making processes, like the closed-loop divergence constrained optimization process used in Chapter 2, this may result in a dead-lock due to high uncertainty in predictions. Even worse, uncertainty-unaware decision making processes may control the robot dangerously.

In this chapter, we discuss our work that aims to aims to make data-driven dynamics modeling, and thus model-based reinforcement learning, more practical for a target system

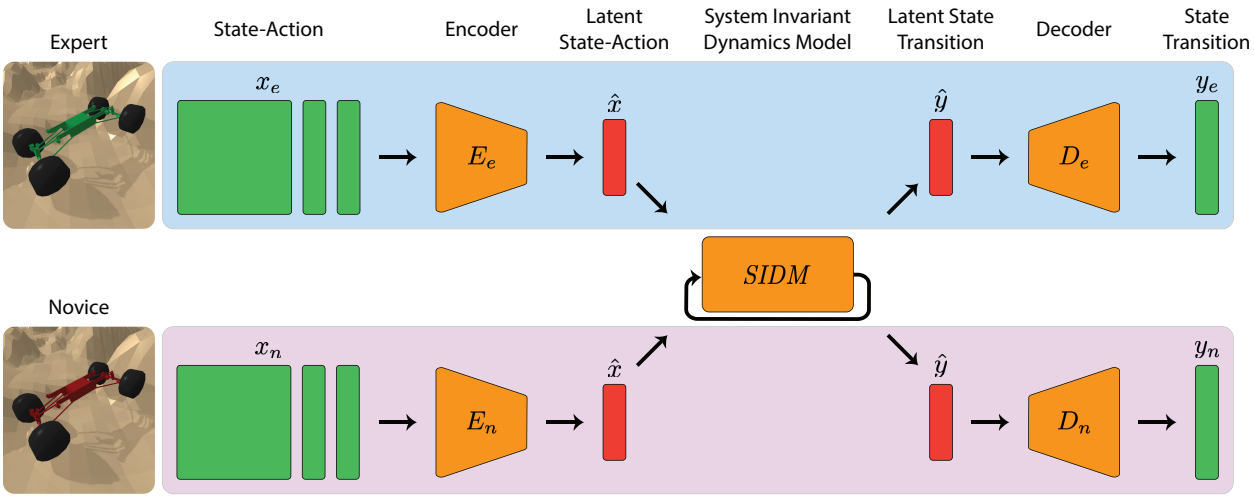


Figure 3.1: The prediction pathway for the expert is shown on the top and the prediction pathway for the novice is shown on the bottom. The expert and novice share a system invariant dynamics model (SIDM) which models the underlying dynamics shared by the systems. Each system has an encoder and decoder to model their relationship to the SIDM.

(novice), by leveraging a similar proxy systems (expert) on which data can be collected cheaply. This approach was first presented in [53] and serves as preliminary work answering the first key question posed in Chapter 1, "Can leveraging low quality simulation data provide insights into real-world vehicle dynamics and reduce the amount of real-world data needed?" While in this chapter we focus on leveraging data from one simulated system to accelerate model training for a different simulated system, this approach could be used in theory to accelerate model training on a real-world system as well.

Our domain adaptation approach, summarized in Figure 3.1, allows model-based reinforcement learning algorithms to use the large expert dataset to learn more accurate novice dynamics models when novice data is limited. This approach trains a System Invariant Dynamnics Model (SIDM) that models underlying dynamics shared between the expert and novice. We provide an algorithm to train the SIDM as well as learn the relationship each particular system has to it. We show through experimentation that using novice and expert data to learn a SIDM and the novice's relation to it results in more accurate predictions for the novice compared to if a dynamics model was trained using solely novice data.

3.2 Related Work

Model Based Reinforcement Learning In model-based reinforcement learning [100, 101], a predictive model of the robot's dynamics is learned. This dynamics model is then used to generate control policies using methods such as the cross-entropy method (CEM) of optimization [102]. Model-based reinforcement learning methods are appealing due to their sample efficiency as well as ability to generalize to new tasks due to the reward independent nature of the dynamics model. However, model-based reinforcement learning methods are sensitive to the prediction accuracy of the dynamics model due to the compounding nature of their error. In this chapter, we focus on using expert training data to improve model prediction accuracy for a novice system in cases where novice training data is limited.

Domain Adaptation In domain adaptation, learning is performed on a expert robot and knowledge is transferred to a novice robot. One common approach for domain adaptation is in Sim2Real, where the goal is to transfer knowledge learned in simulation to the real robot. The simplest approach to the Sim2Real problem is system identification, where parameters of the simulation are fine tuned to closely match the real robot's dynamics. Another Sim2Real approach is domain randomization [103,104], where training is performed on multiple simulations with randomized parameters. Domain randomization allows for robust policies that perform well in a wide range of domains, hopefully including the desired real world domain. Alternatively, training could be performed on a single expert domain, and the relationship of the novice domain to the expert domain could be learned. [105] restylizes images from a novice domain to make them more similar to images from the expert domain. [106] trains a network to transform predictions for the expert to true predictions for the novice. Since these methods train the base policy or dynamics model to be specific to the expert, the novice can only vary slightly from the expert dynamics.

Another approach is to use autoencoders to form system invariant latent representations of expert and novice states. [107] focus on transferring new policies from the expert to

the novice. A new novice policy is trained to output similar states as the known expert policy, using the latent state representation for state similarity comparisons. [108] focus on corresponding expert and novice policies by learning the forward dynamics of the policies in the latent state space, with the hope that this correspondence knowledge can be used to accelerate policy training on the novice. One challenge to learning system invariant representations of states is ensuring that latent state representations from different systems are similar. [107] uses Dynamic Time Warping [91] to find corresponding states from novice policies and expert policies trained for proxy tasks. The encoder is trained to maximize similarity of the corresponding states' latent representation. Unfortunately, this method requires a priori knowledge of similar proxy task policies on the novice and the expert. [108] use Iterative Closet Point (ICP) [109] to correspond the shape of the latent state space for each batch given similar novice and expert policies. Using ICP to match the latent space requires large batch sizes during training, making computation difficult especially in memory limited cases. [108] used a batch size of 4096 during their experiments.

The method proposed in this chapter is similar to [107] and [108]. We use system invariant latent spaces to correspond systems. However, instead of transferring policies, we focus on transferring models of the system's dynamics. We use a system invariant latent space to represent not only states, but also actions. We also use another system invariant latent space to represent state transitions. We ensure that latent state-actions from the expert and novice are similar by using an adversarial network for similarity comparison. A system invariant dynamics model is learned to predict latent state transitions given latent state-actions. This method allows model-based reinforcement learning algorithms to leverage large expert datasets to learn more accurate novice dynamics models when only small novice datasets are available.

3.3 Method

Our method assumes that similar systems share some underlying dynamics, as has been observed for many classes of dynamical tasks such as running and climbing [110]. We model the underlying dynamics of the systems with a System Invariant Dynamics Model (SIDM). The inputs of the SIDM are system invariant representations of the robot state-action tuples. The output of the SIDM are predicted state transitions represented in a system invariant form. For each particular system, an encoder is used to transform the system specific state-action tuples to inputs for the SIDM. Similarly, each system has a decoder that transforms predictions from the SIDM to system specific state transition predictions. Our method trains an encoder and decoder specific to each system, and an SIDM shared between all systems. This prediction structure is shown in Figure 3.1. We use an adversarial network to force SIDM inputs and outputs to be similar between systems, and thus system invariant.

The encoder and decoder of each system can be modeled with relatively small networks, so they are less susceptible to overfitting. Although the SIDM network is required to be larger for precise predictions, it is trained using a larger dataset consisting of data from all systems. As a result, the novice prediction network graph, which consists of the novice encoder, novice decoder, and SIDM, can be trained for high accuracy before it becomes overfit.

3.3.1 Notation

We denote the expert and novice state-action spaces, which contain state-actions pairs for the expert or novice system, respectively, as X_e and X_n. We denote the state-transition spaces, which contains state transition vectors for the expert or novice system, as Y_e and Y_n. We also define two latent spaces. The latent state-action space, denoted as $\hat{X}$, contains system invariant representations of state-action pairs. The latent state-transition space, denoted as $\hat{Y}$, contains system invariant representations of state transitions.

We define an encoder that maps the state-action space to the latent state-action space of the expert or novice as $E_e : X_e \to \hat{X}$ and $E_n : X_n \to \hat{X}$. Similarly, we define a decoder that maps the latent state-transition space to the state-transition space for the expert or novice as $D_e : \hat{Y} \to Y_e$ and $D_n : \hat{Y} \to Y_n$. The system invariant dynamics model predicts resulting latent state transitions given latent state-actions and is denoted as $SIDM : \hat{X} \to \hat{Y}$.

We denote $\mathcal{D}_e = \{(x_e^{(i)}, y_e^{(i)})\}_{i=1}^{N}$ as the expert dataset containing N samples, where $x_e^{(i)} \in X_e$ is a state-action pair resulting in the state transition $y_e^{(i)} \in Y_e$. Likewise, we denote $\mathcal{D}_n = \{(x_n^{(i)}, y_n^{(i)})\}_{i=1}^{M}, x_n \in X_n, y_n \in Y_n$, as the novice dataset contains M samples. We assume that $M << N$. For simplicity of notation, we drop the e or n subscript denoting which system an element or function belongs to when it is clear from context.

3.3.2 Training Procedure

The training approach of our method follows closely to that of Adversarial Autoencoders (AAE) [111]. For AAE, an autoencoder is trained with the two objectives of minimizing reconstruction loss and matching the aggregated posterior distribution of latent representations to an arbitrary prior distribution. For the second training objective, an adversarial training criterion is used to match generated samples (latent representations from the encoder) to the true samples (sampled from the prior distribution). As our application focus is on state transition prediction instead of reconstruction, our approach differs from AAE in two ways: (1) Instead of reconstructing the input and minimizing reconstruction error, our whole network predicts state transitions of the robot and we minimize prediction error. (2) We divide the single latent representation from AAE into two latent representations: a latent state-action representation and a latent state transition representation. We add a SIDM network to predict latent state transitions given latent state-actions.

The training procedure for our method alternates between two phases – the prediction phase, and the regularization phase. The prediction phase aims to increase prediction accuracy for both the expert and novice system. The regularization phase aims to match

the distribution of latent state-actions over novice state-actions to the distribution of latent state-actions over expert state-actions. We only match the distribution of latent state-actions, since the SIDM network should map similarly distributed latent state-actions to similarly distributed latent state transitions. Both the prediction training and regularization phase can be executed on minibatches using stochastic optimization methods such as SGD or Adagrad [112].

During the prediction training phase, samples are randomly chosen from either the expert dataset with probability p_e or the novice dataset with probability $1 - p_e$. For each sample $(x^{(i)}, y^{(i)})$, the appropriate encoder, either expert or novice, is used to map $x^{(i)}$ to a latent state-action $\hat{x}^{(i)}$. Given $\hat{x}^{(i)}$, the SIDM predicts a latent state-transition $\hat{y}^{(i)}$. Finally, the appropriate decoder is used to map $\hat{y}^{(i)}$ to a predicted state transitions $y'^{(i)}$. For the prediction phase, the loss function compares the predicted state transitions to true state transitions using the squared second norm difference,

$$\mathcal{L}_{pred}(y', y) = \|y' - y\|^2 \tag{3.1}$$

The procedure for the prediction training phase is summarized in Algorithm 4.

Algorithm 4: Prediction Training

for *number of prediction training iterations* **do**
  $g \leftarrow 0$
  for *prediction batch size (b_{pred})* **do**
    if *randomly chosen to update expert with probability p_e* **then**
      $(x, y) \leftarrow$ random sample from expert dataset $\mathcal{D}_e$
      $y' \leftarrow D_e \circ SIDM \circ E_e(x)$
    else
      $(x, y) \leftarrow$ random sample from novice dataset $\mathcal{D}_n$
      $y' \leftarrow D_n \circ SIDM \circ E_n(x)$
    $g \leftarrow g + \frac{1}{b_{pred}} \nabla_{E_e, E_n, SIDM, D_e, D_n} \mathcal{L}_{pred}(y', y)$
  $E_e, E_n, SIDM, D_e, D_n \leftarrow$ update parameters using gradient g

The regularization phase follows the Generative Adversarial Networks (GAN) framework [113]. An adversarial network (discriminator) is used to distinguish between latent state-

actions from the expert encoder and latent state-actions from the novice encoder. The sigmoid output of the discriminator, $A(\hat{x})$, predicts the probability that a latent state-action came from the expert instead of the novice. At the start of the regularization phase, the adversarial network is trained to minimize expected negative log-likelihood,

$$\min_{A} \mathbb{E}_{x_e \sim \mathcal{D}_e, x_n \sim \mathcal{D}_n}[\mathcal{L}_{disc}(E_e(x_e), E_n(x_n), A)] \tag{3.2}$$

$$\mathcal{L}_{disc}(\hat{x}_e, \hat{x}_n, A) = -\log A(\hat{x}_e) - \log(1 - A(\hat{x}_n)) \tag{3.3}$$

After updating the weights of the discriminator, the novice encoder (generator) is trained to minimize the negative log-likelihood of latent state-action of the novice coming from the expert.

$$\min_{E_n} \mathbb{E}_{x_n \sim \mathcal{D}_n}[\mathcal{L}_{gen}(E_n(x_n), A)] \tag{3.4}$$

$$\mathcal{L}_{gen}(\hat{x}_n, A) = -\log(A(\hat{x}_n)) \tag{3.5}$$

The procedure for the regularization training phase is summarized in Algorithm 5.

In this formulation, all mappings are deterministic. Alternatively, an approach similar to that taken by [114] could be used, where mappings are nondeterministic. In this alternate approach, the encoders, decoders, and SIDM would estimate posterior distributions for the latent state-actions, latent state transitions, and system specific state transitions. Latent state-actions and latent state transitions would be randomly sampled from the estimated distributions, and negative log-likelihood of the true state transition given estimated posterior distribution from the decoder would be used as the prediction phase loss function.

Algorithm 5: Regularization Training

for *number of discriminator training iterations* **do**
  $g \leftarrow 0$
  for *discriminator batch size (b_d)* **do**
    $x_e, x_n \leftarrow$ random sample from expert and novice datasets $\mathcal{D}_n, \mathcal{D}_e$ respectively
    $\hat{x}_e, \hat{x}_n \leftarrow E_e(x_e), E_n(x_n)$ respectively
    $g \leftarrow g + \frac{1}{b_d} \nabla_A \mathcal{L}_{disc}(\hat{x}_e, \hat{x}_n, A)$
  $A \leftarrow$ update parameters using gradient g
for *number of distribution matching iterations* **do**
  $g \leftarrow 0$
  for *distribution matching batch size (b_g)* **do**
    $x_n \leftarrow$ random sample from novice datasets $\mathcal{D}_n$
    $\hat{x}_n \leftarrow E_n(x_n)$
    $g \leftarrow g + \nabla_{E_n} \mathcal{L}_{gen}(\hat{x}_n, A)$
  $E_n \leftarrow$ update parameters using gradient g

3.4 Experiment Setup

We demonstrate the value of using SIDM by training dynamics models for novice systems with and without knowledge transfer from expert systems using SIDM. The improvement in motion prediction accuracy for the novice is compared over varying amounts of novice training data.

3.4.1 Simulated Systems

The expert and novice systems used were both wheeled robots driving over randomly generated rough terrain simulated with PyBullet [92], as shown in Figure 3.2. The robot actions corresponded to desired wheel velocity and steering angle applied using PD controllers. The expert and novice had different scaling of actions, contact friction, link masses, link inertia, suspension heights, suspension limits, spring constants, and damping constants.

The state-action input (x) to the dynamics models included the robot's tilt (represented as the gravity vector relative to the robot frame), body velocity (represented as a twist), terrain height map centered about the robot, and action taken. The robot centered terrain height maps were generated from global terrain height maps, as shown in Figure 3.2. We

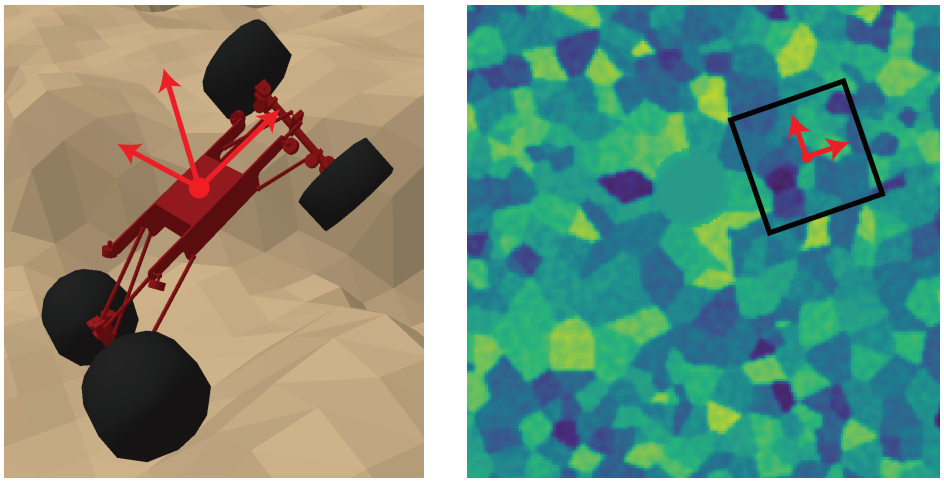


Figure 3.2: Left shows the simulation environment used for experiments. Right shows the world terrain height map with the robot centered map outlined in black. The robot frame is shown in red. State-Actions for prediction included the robot body velocity, gravity vector relative to the robot frame, robot centered terrain map, and action (throttle, steering). Predicted state transitions included the new robot frame relative to the previous, and the new robot body velocity.

find it realistic to assume the robot has knowledge of the global terrain height map due to existing methods for generating these maps from raw sensor data [88, 115]. Alternatively, SIDM could also be used in other approaches where state transitions are predicted directly from raw depth sensor data [74]. The dynamics model's predicted state transition (y) consists of the relative movement of the robot's body (represented as a translation and quaternion orientation vector) and the new body velocity of the robot. In total, the state-action tuple consisted of a 11 dimension vector and a 300 by 300 matrix (terrain map) while the state transition vector had a size of 13.

3.4.2 Model Training

Motion data was collected using random actions and PyTorch [116] was used to train dynamics models. During training, 75 percent of the novice data was used for training and 25 percent of the data was used for testing. We stopped training once the test prediction loss started increasing indicating overfitting of the dynamics model. We then validated the prediction loss of each trained model using a separate novice validation dataset.

When training without SIDM, the prediction phase only sampled data from the novice dataset ($p_e = 0$) and the regularization phase was ignored. To keep the comparison consistent, the same novice dynamics model network structure was used when training without SIDM, and the expert encoder and decoder were simply removed. Anecdotally, we found that changing the network structure when training without SIDM led to minimal change in prediction loss. Adam [117] with weight decay [118] was used to optimize the neural network parameters.

3.4.3 Network Architecture

Each encoder network consisted of a two layer Convolutional Neural Network (CNN) [89] to process the terrain height map, consisting of an 8 channel layer and a 4 channel layer, both with a kernel size of 4. All CNN layers used no padding, and a stride of 1. The output of the CNN was reshaped into a vector, concatenated with the rest of the state and action input, and fed into a single hidden fully connected (FC) layer and an output FC layer, both of size 128 (latent state-action vector, $\hat{x}$). Each SIDM network consisted of four Long-Short Term Memory (LSTM) [90] layers, each with a hidden state size of 1024. The output of the LSTM layers was fed into a FC output layer of size 64 (latent state transition vector, $\hat{y}$). Each decoder network consisted of a single hidden FC layer of size 64, and an output FC layer of size 13. All CNN and FC layers used leaky ReLu except for the output FC layer of the decoder which did not have an activation function.

It is important to use an appropriate value for p_e, otherwise the SIDM network will be biased towards a specific system and not be system invariant. Low p_e values cause training of the SIDM network to be biased towards the novice and not take advantage of the large expert dataset. High p_e values cause the SIDM network to be biased towards the expert and lead to poor performance on the novice. In cases where the novice dataset is much smaller than the expert dataset, choosing a p_e value proportional to the dataset sizes will cause p_e to be too high. In our experiments, we use a p_e value of 0.5 when the dynamics of the novice

and expert differed greatly, and a p_e value of 0.75 when the novice and expert were more similar. For the same reason, the SIDM network cannot be pretrained using just expert data, although we believe the SIDM could be pretrained using multiple expert systems.

It is also important not to train in any one phase for too many iterations. Since the prediction and regularization training phases update the weights of the novice encoder for different loss functions, training in one phase for too long led to high losses for the other phase and unstable training. In our experiments, we set the number of prediction iterations to 100, and the number of distribution matching iterations to 20.

3.5 Results

To show the accuracy improvements using SIDM, we ran training experiments with varying amount of novice data on two different expert and novice systems. In the first system the expert dynamics were more similar to the novice dynamics, while in the second system the expert dynamics greatly varied from the novice dynamics. The difference in the systems was most pronounced in the suspension parameters. In the first system, the suspension of the novice was 5 times stiffer, had 2 times more damping, but had the same amount of travel compared to that of the expert. In the second system, the suspension of the novice was 667 times stiffer, had 100 times more damping, and about 1.5 times less travel compared to that of the expert, leading to an almost completely rigid suspension. For both, we collected about 14 simulation hours worth of motion data for the expert system. For the first system we varied the novice dataset size between 1 minute and 2 hours worth of motion data. For the second system, we increased this range so that the novice dataset ranged between 1 minute and 10 hours worth of motion data, as the improvements using SIDM did not converge to the non-SIDM system by 2 hours. For each system and novice dataset size, we ran 10 trials, for a total of 320 trials overall. Figure 3.3 shows the validation losses for these experiments.

In addition to the prediction validation loss, we also tested the improvement using SIDM

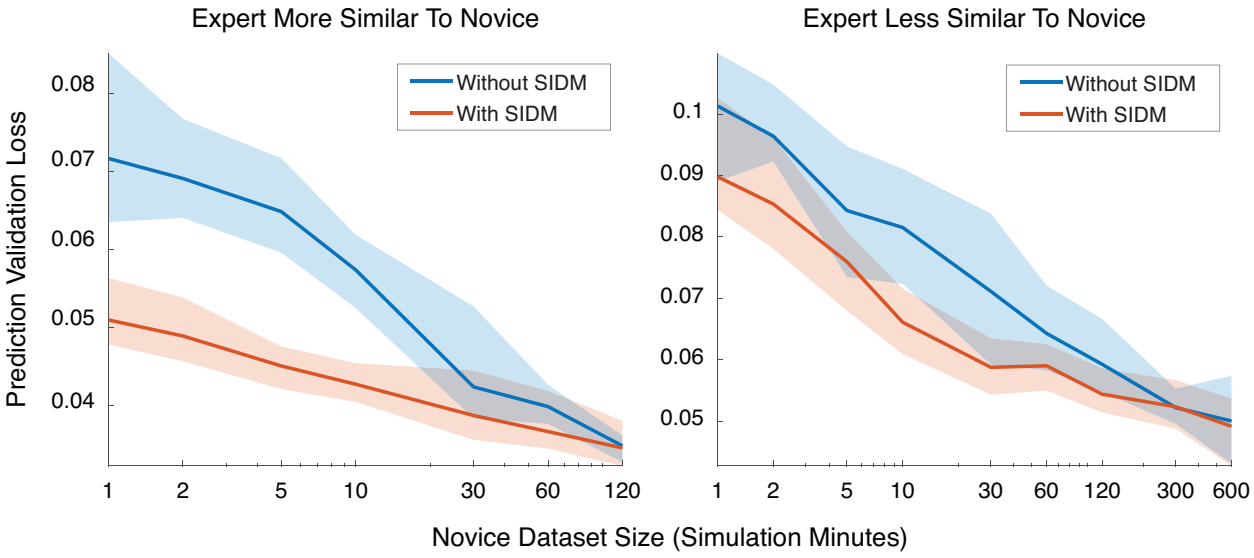


Figure 3.3: Dynamics models were trained for two different novice systems with and without knowledge transfer from expert systems using SIDM. On the left, the dynamics of the expert were more similar to the dynamics of the novice. On the right, the dynamics of the expert were less similar to the dynamics of the novice. The line shows the mean validation loss for 10 trials, while the shading shows the range between the minimum and maximum.

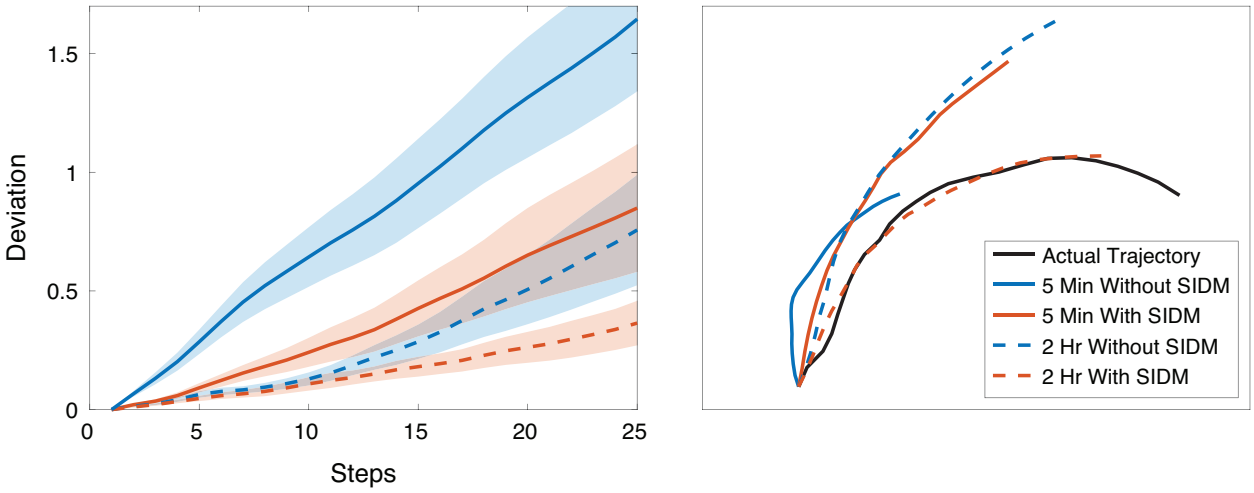


Figure 3.4: The robot's trajectory was predicted for 25 steps (2.5 seconds) using the trained dynamics models. The left figure shows the deviation of the predicted trajectory from the actual trajectory. Each line shows the mean and the shaded region shows the standard error over 10 trials. The right figure shows the actual trajectory and predicted trajectories for one example trial.

by simulating 25 step (2.5 simulation second) trajectories for the novice robot (from the first system). For each trajectory, an action was randomly selected and applied over the length of the trajectory. We use the learned novice dynamics models to predict the resulting trajectories over rough terrain given the action taken. We integrate the predicted state transitions to predicted the robot's absolute position and orientation over the course of the trajectory. At each time step, the predicted absolute position and orientation was used to generate a new robot centered terrain height map for the input of the next prediction. Figure 3.4 shows the results comparing the actual trajectory of the robot to the predicted trajectory of the robot using 5 minutes and 2 hour of novice training data both with and without SIDM.

Using the SIDM to transfer knowledge from the expert system to the novice system resulted in a significant improvement in novice predictions especially when the novice dataset was small. When the expert and novice were more similar, training using SIDM resulted in similar performance to training with up to 20 times the amount of data as training without SIDM. When the expert and novice were less similar, training using SIDM resulted in similar performance to training with about 5 times the amount of data as training without SIDM. As expected, when the novice dataset becomes larger the performance with and without SIDM become comparable (at about 2 hours and 5 hours of novice training data for the two systems, respectively). Although using SIDM does not increase overfitting as the dataset becomes larger, it does make the network harder to train. We recommend reducing expert train probability, discriminator training iterations, and distribution matching iterations as the novice dataset grows. Doing so will effectively cause training to be done only on the novice system data once the benefits of SIDM are expected to be minimal.

3.6 Conclusion

In this chapter, we explored how dynamics model training on a novice system could be accelerated by leveraging cheap training data collected on an expert system, with similar but different dynamics. More specifically, we investigated whether more accurate novice system dynamics models could be trained with less data collected on the novice system. The implication of this work is that model-based reinforcement learning could be made more practical for real-world systems, by leveraging cheap training data collected on a simulated system that imperfectly approximates the real-world system.

Our approach uses what we call a System Invariant Dynamics Model (SIDM) to model the underlying dynamics shared between the two systems. The SIDM is trained using motion data from both the expert and the novice. We provide an algorithm to train the SIDM as well as an encoder and decoder for each system that describe each system's relationship to the SIDM.

This chapter serves as preliminary work addressing the first question posed in the introduction, "Can leveraging low quality simulation data provide insights into real-world vehicle dynamics and reduce the amount of real-world data needed?" We show through experimentation that training dynamics models for the novice using SIDM and expert data leads to more accurate novice dynamics models especially in cases where motion data for the novice is limited. These promising results show that data collected in simulation could help accelerate dynamics models training for a real-world system.

While the experimental results presented in this chapter show promise, we note a few practical drawbacks, which we aimed to improve upon in our subsequent work, presented in Chapter 4. First, while using SIDM does make dynamics modeling more sample efficient for the target system, it still requires a substantial amount of data from the target system. In extreme cases where there is virtually no data from the real world system, predictions from the dynamics model will be nonsensical. Second, the resulting dynamics model using the SIDM approach is deterministic and does not capture uncertainty in predictions. Given

that the benefits are most pronounced in low-data regimes, where there will still inevitably be model inaccuracies, this may lead to poor or potentially dangerous policies resulting from decision making exploiting modeling errors. Lastly, the approach from this chapter requires a lengthy network training process for every new target system. This can be problematic in situations where the robot's dynamics are expected to frequently change. For example, when operating in environments composed of diverse terrain, or if the payload of the robot is consistently changing.

Given these limitations, we recommend the readers to refer to Chapter 4, where we discuss our subsequent approach that effectively addresses all these limitation.

Chapter 4

Simulation-Trained Dynamics Models that Robustly Adapt to New Systems

4.1 Introduction

Reinforcement learning, while powerful, faces practical challenges when applied to real-world robotic systems. One significant hurdle is the requirement for extensive training data, collected through numerous trials on the system. Not only is this process time consuming for real-world systems, but also dangerous. Inappropriate or misjudged actions can cause substantial damage to the robot, requiring expensive and time-intensive recovery and repair efforts.

Consequently, simulation has become instrumental in the development and validation of off-road driving algorithms. Beyond offering a risk-free environment for testing, simulations can operate faster than real-time, benefit from parallelization, and conduct trials autonomously. Simulation has been especially crucial in the development of model-free reinforcement learning algorithms [71, 72, 119], which aim to directly optimize a policy over many trials, as deploying an underdeveloped model during trials often leads to catastrophic failure.

However, the performance of policies trained and validated in simulation do not always transfer to the real world. This discrepancy arises from the "reality gap" – the inevitable differences between the simulated environment and the real world. Addressing these challenge requires effectively translating simulation-trained policies into the real-world, known as the "sim2real" transfer problem. While some methods aim to minimize the reality gap [120–124], accurately modeling intricate dynamics of a robot interacting with a diverse range of unstructured terrains remains challenging. Robot dynamics are not only affected by the robot properties, such as weight distribution, tire friction coefficient, and motor models, but they are also affected by the unknown terrain properties including soil cohesion, dampness, or presence of debris.

While simulated dynamics inevitably differ from real-world dynamics, we explored in Chapter 3 how training data collected in simulation could augment real-world data to help accelerate dynamics model training for model-based reinforcement learning. We demonstrated an approach that used both large amounts of data collected on an expert system (simulation) and limited amounts of data collected on a target system (ideally a real-world system) to train a dynamics model for the target system. We shown that this approach resulted in more accurate dynamics models for the target system when compared to training a dynamics model using solely target system data.

While this approach shows promise, its primary limitation lies in its deterministic nature which provides no notion into prediction uncertainty. This can be problematic for decision making in scenarios where the resulting dynamics model is unreliable, such as when target system data is particularly limited or when there is a high difference in dynamics between the expert and target system.

In this chapter, we build upon concepts introduced in Chapter 3, further addressing the first question posed in Chapter 1: Can leveraging low quality simulation data provide insights into real-world vehicle dynamics and reduce the amount of real-world data needed? Furthermore, we also build upon concepts introduced in Chapter 2, further exploring the

quantification of prediction uncertainty to make decision making robust towards inaccurate dynamics models. This is a vital component to addressing the second question: Can quantifying uncertainty in an imperfect dynamics model improve long-horizon decision making?

We propose a novel framework for sim2real transfer that balances robustness with adaptability, first published in [54]. Our method follows the model-based reinforcement learning (MBRL) paradigm, where a probabilistic predictive dynamics model is first trained then used for decision making. We train the model in simulation with varying simulation parameters to make our model robust across a variety of system dynamics. Similar to prior methods [120, 125], we train a neural network to extract a latent context vector to help adapt the policy to the target system's particular dynamics. In our approach, this neural network, called the System Identification Transformer (SIT), uses attention mechanisms to distill state-transition observations from the particular system into a context vector understanding of its particular dynamics. Unlike other approaches that condition a policy on this context vector, our approach instead conditions a dynamics model on the context vector. Given this context vector, the Adaptive Dynamics Model (ADM) probabilistically models the system's dynamics, capturing uncertainty both from the system's inherent stochasticity and from ambiguities due from insufficient state-transition observations. Online, we use a Risk-Aware Model Predictive Path Integral (RA-MPPI) controller [126] to safely control the robot under its current understanding of dynamics.

The remainder of this chapter aims to validate the following hypotheses:

Hypothesis 1: Our proposed approach enables safer control in terms of the number of constraint violations, even when there is insufficient historical observation data (e.g. upon initialization).

Hypothesis 2: Leveraging the attention mechanism to extract context allows for continual improvement of the adapted policy (i.e. better lap times) as the number of state-transition observations increases.

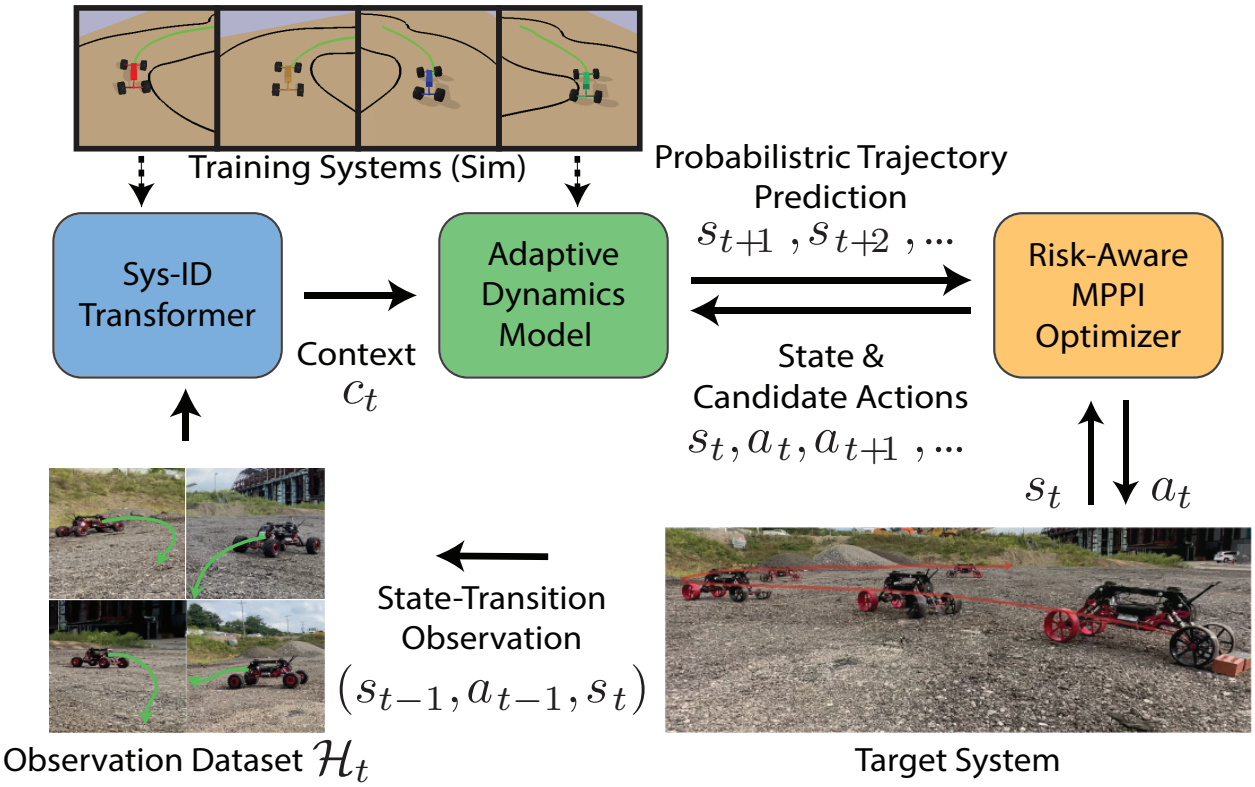


Figure 4.1: Method Overview: The System Identification Transformer (SIT) and Adaptive Dynamics Model (ADM) are trained with randomized simulation dynamics to gain a probabilistic understanding of any target system's dynamics. The SIT leverages an attention mechanism to condense state-transition observations from the target system into a compact context vector. The ADM predicts state transition distributions conditioned on robot state, action, and context vector. Online, Risk Aware MPPI chooses safe actions according to the ADM's probabilistic predictions.

Hypothesis 3: Using a risk aware MPPI controller reduces the number of constraint violations compared to a risk unaware controller with the same SIT and ADM models.

4.2 Background

One common approach to the sim2real transfer problem includes domain randomization [103, 122, 127, 128]. In this approach, a model-free reinforcement learning policy is trained in simulation while simulated dynamics are varied through randomizing simulation parameters. This approach ensures that the trained policy is robust under a diverse range of dynamics, hopefully including the dynamics of the real-world system. However, a notable trade-off of

this approach is its tendency towards conservative performance, as the policy is designed to generalize across a spectrum of potential systems rather than being optimized for a specific one.

Alternatively, some approaches train a latent vector conditioned policy that can be adapted to some particular dynamics simply by identifying a suitable latent vector. In [129–131], suitable latent vectors were found through optimization techniques such as CMA-ES [132]. Although these approaches can tailor the policy towards the particular system, they still require trial and error to refine the policy, and may be unsafe while the policy is being refined. In [120, 125], an auxiliary neural network is used to rapidly identify a suitable latent vector given a short fixed-length horizon of prior states and actions. While this allows for faster adaptation, the fixed-horizon input only utilizes recent observations for latent vector inference. Furthermore, the model-free nature of these methods prohibits any interpretability with respect to the adaptation process or the resultant policy.

4.3 Probabilistic Predictive Dynamics Model

We formulate the autonomous off-road driving problem as a distribution of Markov decision processes (MDPs), where each real world environment is represented by a single MDP. For a given environment i, the problem is defined as $(\mathcal{S}, \mathcal{A}, \mathcal{P}_i, \mathcal{C}_i)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}_i(s_{t+1}|s_t, a_t)$ is the stochastic discrete-time transition dynamics from $s_t \in \mathcal{S}$ to $s_{t+1} \in \mathcal{S}$ under action $a_t \in \mathcal{A}$, and $\mathcal{C}_i(s, a)$ is the cost function for a given state-action pair. In this formulation, the state and action space is shared between environments, but the transition dynamics and cost function are unique to each environment. The function $\mathcal{P}_i$ is a member of the function space $\mathcal{F}$ that comprises all possible stochastic transition functions. We define $\mathcal{W}$ as the distribution over of the function space $\mathcal{F}$ which encompasses all potential dynamics functions the robot might encounter in the real world.

Note that it is impossible to perfectly simulate the unknown dynamics $\mathcal{P}_i$ for any given

real world environment i, let alone the distribution $\mathcal{W}$ of all real world dynamics. We instead define a proxy distribution of dynamics in simulation, $\hat{\mathcal{W}}$, such that $\text{supp}(\mathcal{W}) \subseteq \text{supp}(\hat{\mathcal{W}})$. That is, all of the true dynamics in $\mathcal{W}$ lie within the range of dynamics functions represented in $\hat{\mathcal{W}}$. Using many cheap simulations sampled from $\hat{\mathcal{W}}$, we train a policy that can safely adapt to the particular system dynamics within $\text{supp}(\hat{\mathcal{W}})$, which includes all real world systems lying in $\mathcal{W}$.

Following the model-based reinforcement learning paradigm, we train a predictive model to approximate the probabilistic transition dynamics of any given system. The predictive model consists of two key components: the System Identification Transformer (SIT) and the Adaptive Dynamics Model (ADM). This model is then utilized for decision making, specifically using MPPI with a Conditional Value-at-Risk cost to drive the robot safely given the stochastic predictions from the predictive model.

System Identification Transformer (SIT) The SIT, denoted by $\mathcal{T}_\theta$, identifies the dynamics of a given target system by analyzing prior state-transition observations collected on that system, denoted by $\mathcal{H}$, and extracting relevant information about the target system's dynamics into a latent context vector, denoted as c,

$$c_t = \mathcal{T}_\theta(\mathcal{H}_t). \tag{4.1}$$

In this formulation, state-transition observations collected for a target system at time t are,

$$\mathcal{H}_t = \{(s_i, a_i, s_{i+1}) | i < t-1\}. \tag{4.2}$$

We use a transformer network [133] for the SIT due to several advantages it offers. Transformers can natively accommodate sequences of varying lengths by utilizing self-attention mechanisms to selectively focus on specific segments of the input sequence. These advantages are crucial for our application since state-transition observation sequences expand with

the system's run-time. Furthermore, not all state-transition observations are of equal significance (e.g., periods when the robot remains stationary may offer minimal insights), so this selective focus ensures the extracted context is most representative of the system's dynamics.

The SIT's architecture mirrors the encoder component from [133]. It comprises of a series of identical layers, each featuring a multi-head self-attention sub-layer followed by a position-wise, fully connected feed-forward network sub-layer. Each sub-layer incorporates a residual connection [134] followed by a layer normalization [135]. Unlike the original design, we opted not to use positional encoding for the input sequences. In our application, the order of (state, action, state-transition) observations is unimportant. In our trials, incorporating positional encoding seemed to negatively impact performance. Finally, we aggregated the vector outputs from the last layer by taking their mean, resulting in a single context vector. This compact representation, $c \in \mathbb{R}^{32}$ for our implementation, encapsulates the essence of all prior state-transition observations.

Adaptive Dynamics Model (ADM) The ADM provides a probabilistic understanding of the robot's dynamics based on the context vector extracted by the SIT. The ADM, denoted as $\mathcal{P}_\theta(s_{t+1}|s_t, a_t, c_t)$, is trained to predict state-transition distributions conditioned on the robot's current state, action, and context vector c_t extracted by the SIT. By predicting state-transitions as probability distributions, the ADM can capture uncertainty inherent to the non-deterministic system as well as ambiguities resulting from limited state-transition observations.

Similar to Chapter 2, the adaptive dynamics model can be used to predict a trajectory distribution for the robot by sequentially iterating through each time step of the prediction horizon and chaining samples from the predicted state-transitions distribution, as is done in Algorithm 6. We chose to use a Long Short Term Memory (LSTM) architecture [90], which inherently captures temporal dependencies across state-transition sequences. For our implementation, the LSTM is followed by a fully connected network to predict a multivariate

Gaussian state-transition distribution, parameterized by its mean μ and covariance matrix Σ. More specifically, the covariance component of the neural network output is the off-diagonal and positive diagonal terms of a lower triangular matrix L, where $\Sigma = LL^T$.

4.4 Risk-Aware Model Predictive Path Integral

In this section, we describe how controls can be made robust against the uncertainty in the probabilistic output of the SIT and ADM. This allows the robot to drive safely even when it is unsure about its dynamics while improving performance as its understanding improves with more state-transition observations.

Track Driving Problem For our application, the robot is tasked with driving down different tracks. Each track is defined by a *path* (its center line) and a fixed width w. Given *path*, we structure the task as the following constrained optimization problem,

$$\underset{a_{t_0},\ldots,a_{t_f}}{\text{maximize}} \quad L_{path}(s_{t_f+1}) \tag{4.3}$$

$$\text{subject to:} \quad s_{t+1} \sim \mathcal{P}_i(s_{t+1}|s_t, a_t) \tag{4.4}$$

$$D_{path}(s_t) \leq w \tag{4.5}$$

$$|\ddot{s}_{t,lateral}| \leq A, \tag{4.6}$$

where $\mathcal{L}_{path}(s)$ denotes the distance of state s along *path*, $D_{path}(s)$ denotes the distance of state s from *path*, and $\ddot{s}_{t,lateral}$ denotes the lateral component of the robot's acceleration (calculated through numerical differentiation). Intuitively, the robot's task is to make as much progress down the track (4.3), while staying on track (4.5), and keeping lateral acceleration under a threshold to prevent it from rolling over (4.6), subject to the stochastic dynamics (4.4).

Robust Controls While numerous methods exist for robust control of systems with probabilistic dynamics, e.g. [75, 136], we use Model Predictive Path Integral (MPPI) [137] with a Conditional Value-at-Risk (CVaR) cost to avoid risky actions, similar to [126].

MPPI is a variant of Model Predictive Control (MPC) that relies on a sampling-based approach for trajectory optimization. During each MPPI optimization iteration, candidate action sequences are sampled from a distribution centered around the previous solution. The cost associated with each candidate action sequence is evaluated by simulating the system with a predictive model. The solution is then updated by weighting the candidate actions based on their costs.

To minimize constraint violation within MPPI, we use the relaxed logarithmic barrier function introduced in [138]. This function reformulates a constraint of the form $z \geq 0$, into the following as an additional cost term:

$$\hat{B}(z) = \begin{cases} -ln(z) & z > \delta \\ \beta_e(z;\delta) & z \leq \delta \end{cases} \tag{4.7}$$

$$\beta_e(z;\delta) = \exp\left(1 - \frac{z}{\delta}\right) - 1 - \ln \delta \tag{4.8}$$

In our approach, we enhance the robustness of MPPI against uncertainties in system dynamics by incorporating a CVaR cost, Algorithm 6. The CVaR cost quantifies the expected cost in the worst α percent of scenarios. To calculate the CVaR cost for each candidate action sequence, we perform multiple trajectory simulations using our stochastic dynamics model (ADM) and average the cost of the worst-performing trajectories. This enables the optimizer to be risk-aware when choosing actions.

Algorithm 6: Calculating CVaR Cost

Input : Initial State: s_{t_0},
Context Vector: $c_{t_0} = \mathcal{T}_\theta(\mathcal{H}_{t_0})$
Candidate Actions: $a_{t_0}, a_{t_1}, \ldots, a_{t_f}$,
Number of Stochastic Evaluations: N
Confidence Level: α

Output: CVaR cost

for $j \leftarrow 1$ **to** N **do**
  $\hat{s}_{t_0} \leftarrow s_{t_0}$
  $J_j \leftarrow 0$
  for $t \leftarrow t_0$ **to** t_f **do**
    $\hat{s}_{t+1} \sim \mathcal{P}_\theta(\hat{s}_{t+1}|\hat{s}_t, a_t, c_t)$
    $J_j \leftarrow J_j + \mathcal{C}(s_t, a_t)$

return *Average of top* $\lceil \alpha \cdot N \rceil$ *values of* J

4.5 Training in Simulation

We train the SIT and ADM solely in simulation. However, instead of using one simulated system, we sample a large number of simulated systems from the distribution $\hat{\mathcal{W}}$, created by randomly varying physical parameters in simulation. By doing so, we train the SIT and ADM to adapt to a wide variety of systems including real world systems from the distribution $\mathcal{W}$. During training, we cycle between a data collection phase and a model training phase.

4.5.1 Data Collection

During the data collection phase, we first generate a set of new systems in simulation using PyBullet. To generate a new system, we randomize link dimensions, link inertial terms, scaling of steering and throttle commands, motor torque and PID values, contact parameters (friction, stiffness, and damping), and suspension parameters (limits, stiffness, damping). For each system, we collect a set of trajectories by driving the system using the current SIT and ADM models within the Risk Aware MPPI framework. At each time step during driving, the policy is adapted to the particular system by feeding all prior collect data on that system into the SIT.

4.5.2 Neural Network Training

During the model training phase, we sample a system and time step from the dataset and use the SIT and ADM to predict the state-transition given the robot's current state, action, and all state-transition observations collected on the particular system prior to that time step. We update the neural network parameters of SIT and ADM using a negative log-likelihood loss with an Adam optimizer [117].

4.5.3 Distributed Simulation & Training

We opted for a distributed simulation architecture to better facilitate large scale multi-system data collection. With this architecture, a server trains a SIT and ADM models while an arbitrary number of clients collects training data in simulation. During each training cycle, the server generates a new set of simulated systems, described by a set of robot and physics parameters, and sends them to the clients along with the current neural network parameters of the SIT and ADM models. Each client loads the particular simulated system, collects data on the system using the current policy, and sends the results back to the server. Asynchronously, the server trains the SIT and ADM models, reloading the dataset as new data is received from the clients.

Since each instance of MPPI along with the neural-network training procedure was extremely GPU intensive, running more than one process on a machine would have been difficult. This distributed architecture was invaluable as it allowed us to train the neural network, and collect data on multiple systems in parallel using multiple cloud instances. Furthermore, it allowed us to add or remove simulation clients depending on computational resources.

4.6 Experimental Results

We compare our method against a different baselines in simulation and on a real world robot. In simulation, we run large statistical tests comparing the performance metrics of different approaches on newly generated systems and tracks, none of which were seen during training. On the real world system, we evaluate whether the trained model and resultant policy can safely adapt to different real world systems. We vary the dynamics of the real world system by changing the robot's configuration and varying the type of terrain used. Both the simulated and real world systems use a four-wheeled robot with flexible solid-axle suspension and all wheel steering.

4.6.1 Fast and Continual Adaptation to New Dynamics

In the first experiment, we evaluate this method's ability to generate a safe and effective policy for a new system upon initialization and then continually adjust that policy to better adapt to the target system. For each newly generated system, we run trials over randomly generated tracks starting with no state-transition observations. These new state-transition observations created by driving the system are collected and used to adapt the model at every time step.

For the baseline comparison, we use a model-based reinforcement learning policy where the neural network dynamics model is reinitialized for each new system and trained using only data collected on that particular system. In this baseline approach, the dynamics model uses the same architecture as our Adaptable Dynamics Model, but is given a fixed zero vector for the context input. We collect training data by driving the robot using the baseline model and retraining the model every 250 time steps.

We evaluated the performance of both methods as a function of time steps collected for training or adaptation. We fix the models created given different amounts of data and use them to drive the robot down a new test track. The test track is fixed between all methods

and models for a particular system, but varied for the different systems or trials. Note that for our method, we allowed the model to continue adapting on the test track run since adaptation involved simple SIT inference, which could be computed at each time step.

During each evaluation, we record the lap time (in time steps of 0.1 seconds) needed to complete the test track as well as the number of constraint violations (either the robot driving off track or exceeding the lateral acceleration limit). We also record the number of times the robot made no progress, or was stationary for too long, due to MPPI struggling to find a non-trivial solution. In cases where the robot makes no progress or violates the constraints, the robot is reset to the center of the track at the last progress point and allowed to continue. The average lap time and number of constraint violations for both methods across 230 systems are shown in Figure 4.2.

Compared to the baseline, our method was able to reach much higher levels of performance in low-data regimes. With our method, the robot was able to drive down the test track even when initialized with zero data. With the baseline method, we were unable to evaluate its performance with less than 500 time steps of training data, as the robot often could not finish the track. When comparing our method given zero data and the baseline given 500 time steps of data, our method had a much faster lap time and exhibited many fewer constraint violations. Furthermore, the baseline method averaged 3.4 incidents of no progress per test track run, where the robot needed to be reset due to not making any progress. In comparison, our method averaged 0.004 incidents. Unlike our approach, the baseline approach is impractical to deploy on real world systems due to the high number of constraint violations and resets needed in low-data regimes. This evidence supports **hypothesis 1**, since our approach enables safer control in absence of historical observation data.

For both methods, the policy's performance improved with more training data. When given 5000 time steps of data, the baseline method exhibited an average lap time of 69.43 time steps and averaged 0.0087 constraint violations. In contrast, after only 500 time steps

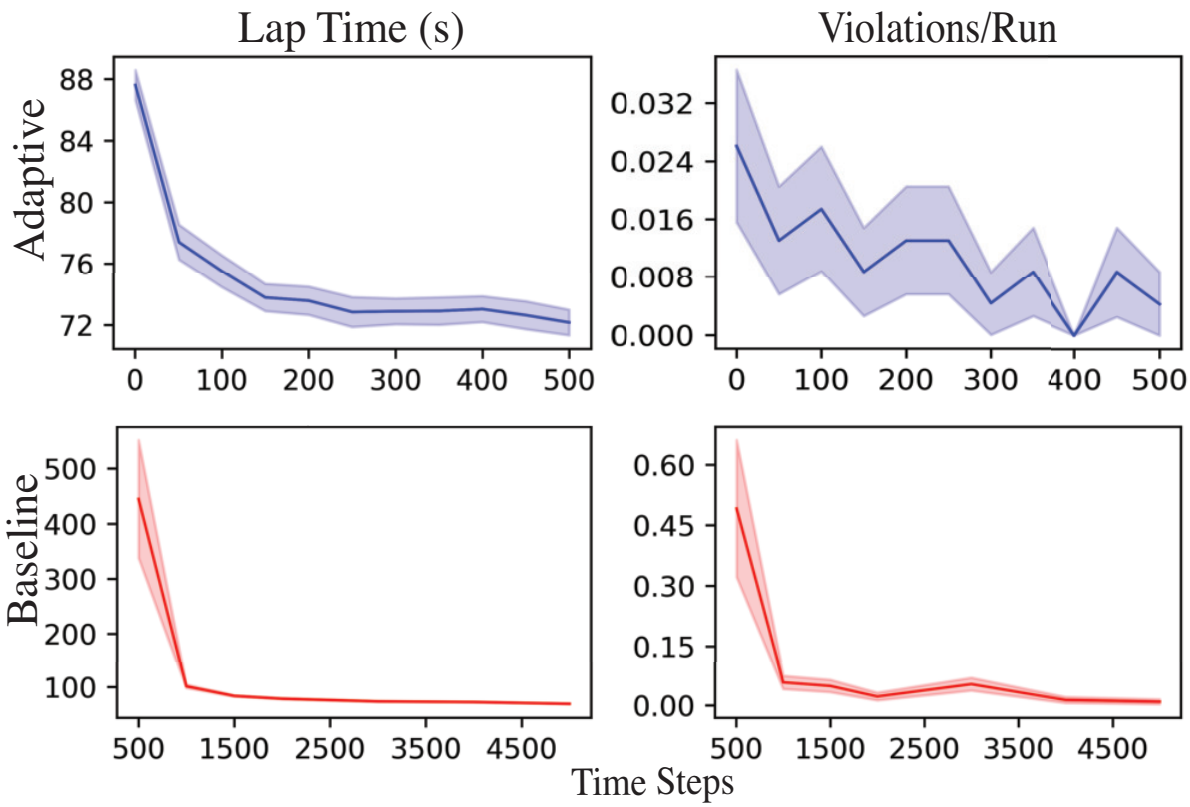


Figure 4.2: Adaptive Method vs. Baseline Method. For the baseline, a new policy was trained on each target system. The standard error is shown with shaded region. For each time step at fixed intervals, we take the model trained at that time step to run each system on a test track in simulation. We averaged out the lap time and violations across all systems as it completed the track.

our approach had an average lap time of 72.17 and averaged 0.0043 constraint violations.

By using attention mechanisms in the SIT, our approach can use variable length state-transition observation sequences to tailor the ADM to a particular system. This allows for continual improvement of the policy for a potentially long period of time, where the observation sequence is long. This is shown in our experiment (Figure 4.2), where our method exhibits gradual performance improvements from 0 to 500 time steps of data. Furthermore, at 500 time steps of data, the performance of our method is comparable to the performance limits of training a policy from scratch for the particular system. This supports **hypothesis 2**, since the adaptive method continually improves in lap-time performance as the number of state-transition observations increases.

Note that when give 0 seconds of historical observations, our method is similar in concept to domain randomization, where a policy is trained to be robust across a wide range of simulated dynamics. When given no historical observations the probabilistic ADM predictions capture the wide distribution of dynamics simulated during training. As a result, similar to domain randomization, our method is initially robust across a wide range of systems leading to few constraint violations. However, our approach tailors the model to each particular system allowing for increased performance as more observations are collected.

4.6.2 Safety During Adaptation

We evaluated our method's ability to remain safe during initial periods of adaptation by comparing it to a baseline that did not consider uncertainty in MPPI. During MPPI, this baseline calculated the cost of an action sequence by predicting the resulting trajectory using the deterministic transition model $\hat{s}_{t+1} = \mathbb{E}[\mathcal{P}_\theta(\hat{s}_{t+1}|\hat{s}_t, a_t, c_t)]$. This is in contrast to our method that calculates a CVaR cost based on predicting multiple possible trajectories under the stochastic dynamics $\mathcal{P}_\theta(s_{t+1}|s_t, a_t, c_t)$, described in Section 4.4.

We compared the two methods by generating 1000 new systems in simulation and one track per system. For each system, we use both methods to drive the robot down the same track 5 times, starting with zero state-transition observations on the first run and adapting the model at each time step throughout the 5 runs. For the two methods, we plot the average lap time and number of constraint violation for the 5 runs, across the 1000 systems, in Figure 4.3.

Our method exhibited far fewer constraint violations than the baseline method that did not use risk aware MPPI. The average number of violations among all runs were 0.015 for our method and 0.49 for the risk unaware MPPI method. The risk unaware MPPI method exhibited more violations on the first run, with an average of 0.59, than on the last run, with an average of 0.45, due to the model adapting and improving. For both methods, the lap time dramatically improved from the first run to the second run, but had minimal

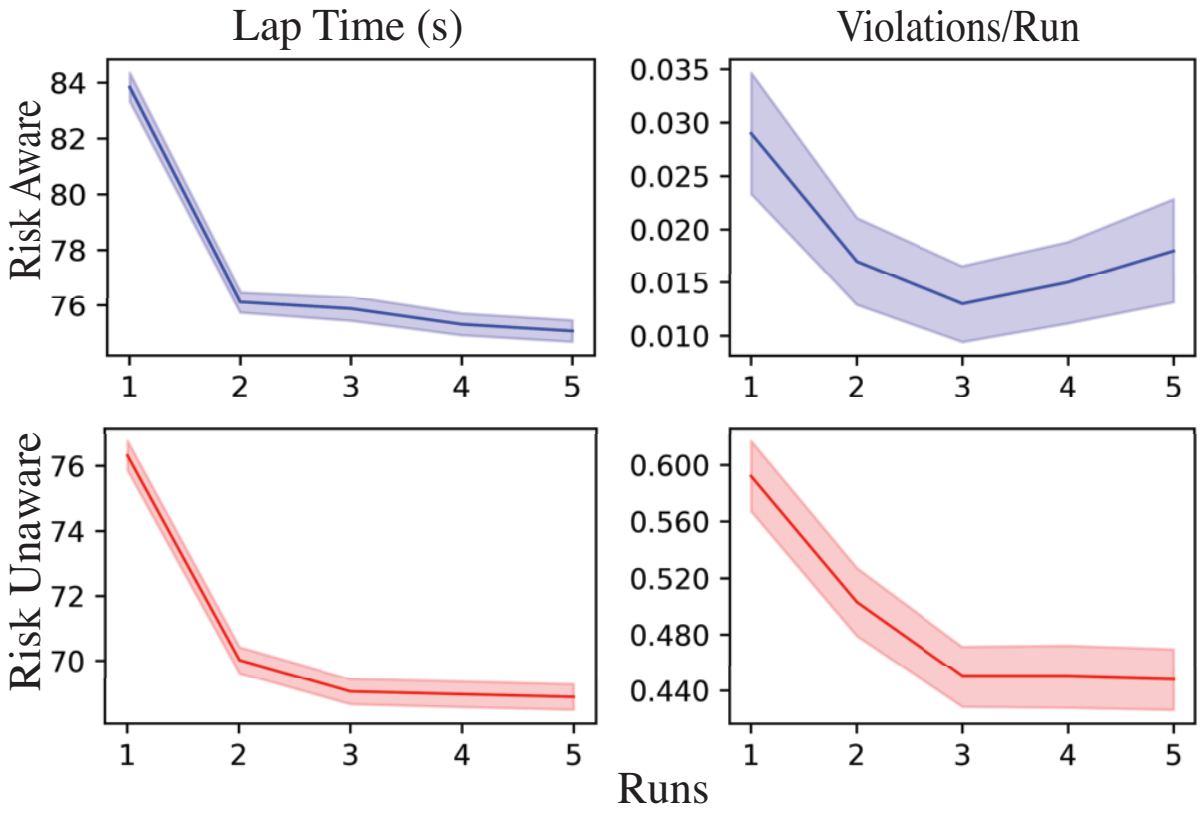


Figure 4.3: Risk Aware vs Risk Unaware MPPI with Adaptive Model. Lap time and number of violations per run are shown, with the average over all systems as the solid line and the standard error shown as the shaded region.

improvements after the second run. We attribute this to the robot driving down the same track for all runs leading to saturation of useful information that could be extracted after the first run. The risk unaware MPPI method had a significantly faster lap time than the risk aware MPPI method. However, it achieved faster lap times by driving aggressively off track and leveraging the penalty-free resets to the middle of the track whenever a constraint was violated. This evidence supports **hypothesis 3**, since the risk aware MPPI is shown to have significantly fewer constraints compared to the risk unaware version.

4.6.3 Sim2real Transfer

In this experiment, we evaluated the ability of our method to transfer to real world dynamics, shown in Figure 4.4. As a baseline, we trained a model-based reinforcement learning policy

in simulation using only the fixed nominal dynamics, meaning simulation parameters were a best approximation of the real world system and not varied. The training procedure for this baseline method followed closely to that from Section 4.6.1. We then ran the policy from both methods on a real world robot. Between trials, we introduced variations to the system's dynamics by changing the terrain type (concrete, dirt, and gravel) and the robot's configuration by changing the scaling of steering and throttle commands as well as swapping the standard rubber tires with low friction PLA 3D printed tires. For each new system dynamics, we reinitialized our method and allowed it to adapt to the new system's dynamics. The baseline method was fixed and therefore was not retrained whenever the system changed. In total, we ran 10 trials of each.

For the real world system, MPPI controls was ran on board at 10 Hz using an onboard computer with an Intel i7 processor and NVIDIA GeForce RTX 2060 GPU. For odometry, we used a Microstrain 3DM-GQ7-GNSS/INS sensor. This sensor comprised of a dual antenna, multi-band RTK GPS sensor along side an IMU. Signals from the RTK GPS sensor and the IMU were fused using an extended kalman filter. To account for noise in real-world odometry measurements, we also added Gaussian noise to the measured state in simulation during training. The variance in noise added was determined by keeping the real-world robot stationary, and measuring the variance in the odometry readings.

For each trial, we used both methods to drive the robot down a fixed track 5 times. For our adaptive method, the robot was given no state-transition observations at the start of the first run, but allowed to adapt using collected observations at each time step throughout the 5 runs. For the baseline method, performing more runs had no effect since there was no mechanism for adaptation. As such, we averaged the performance over all runs for the baseline method. For all runs, the robot was automatically stopped anytime a constraint was violated and manually placed on the center of the track. For every reset, we assigned a 10 second penalty, as the manually resets usually took longer than 10 seconds. The penalized lap times for both methods are shown in Figure 4.5.

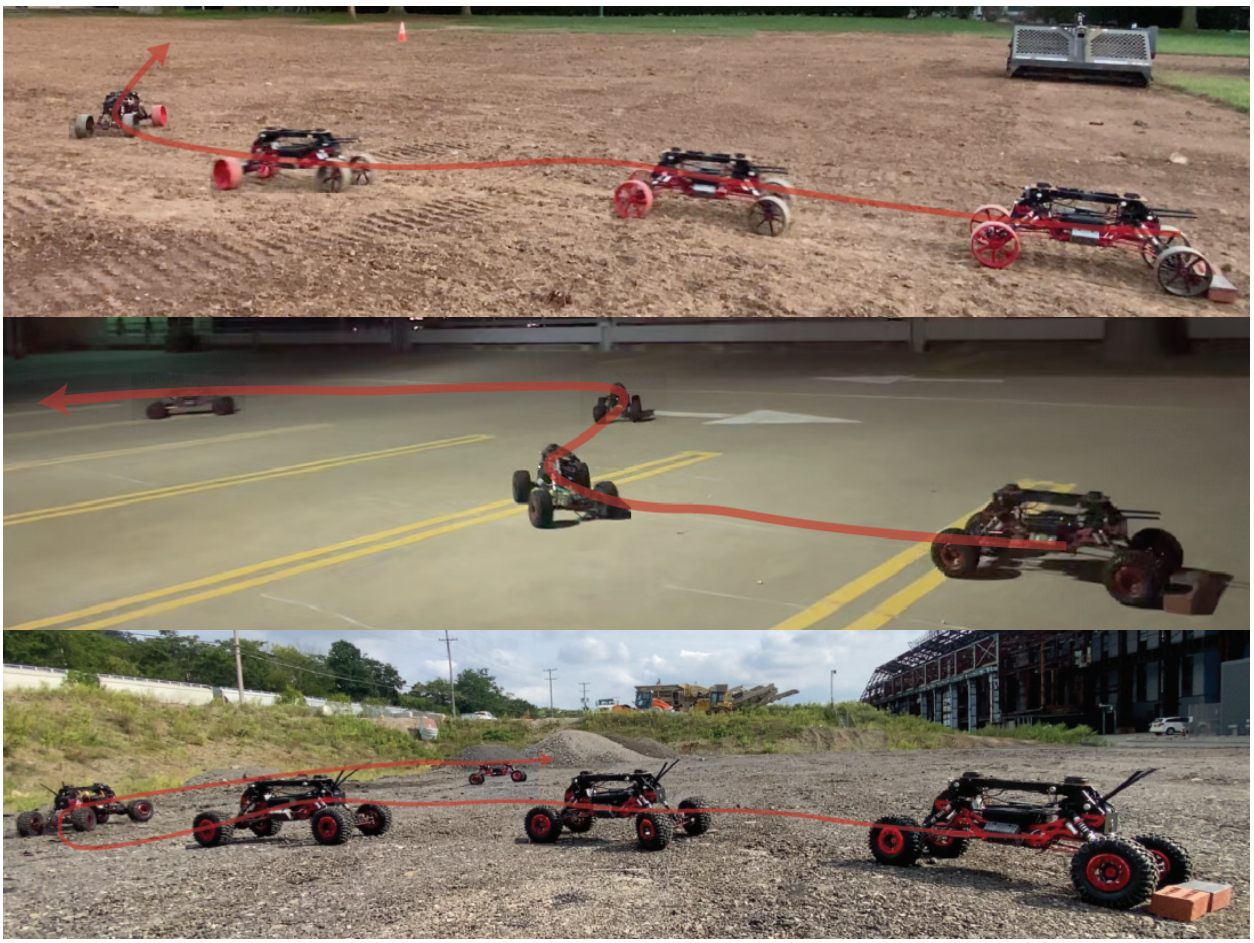


Figure 4.4: Sim2real Experiments. From top to bottom, the wheeled robot driving on dirt, concrete, and gravel. The red line indicates the predefined track for each trial. The tires were changed from compliant rubber tires to hard plastic tires in the dirt experiment shown.

Our method completed all runs with a 100% success rate, where success is defined as completing the track with no constraint violations. This is much higher than the baseline, which had a 40% success rate. Furthermore, we ran a paired t-test between the first and second run's laptime for our method and found significant improvement for the second run with a p value of 0.017. However, none of the successive runs showed any further significant improvement ($p < 0.05$) from the second run. Again, we attribute this to the fixed track leading to saturation of useful adaptation information after the first run. For the baseline method, which used a non adaptive model, there was no statistically significant difference in lap times between runs. This provides additional evidence for **hypotheses 1** and **2**,

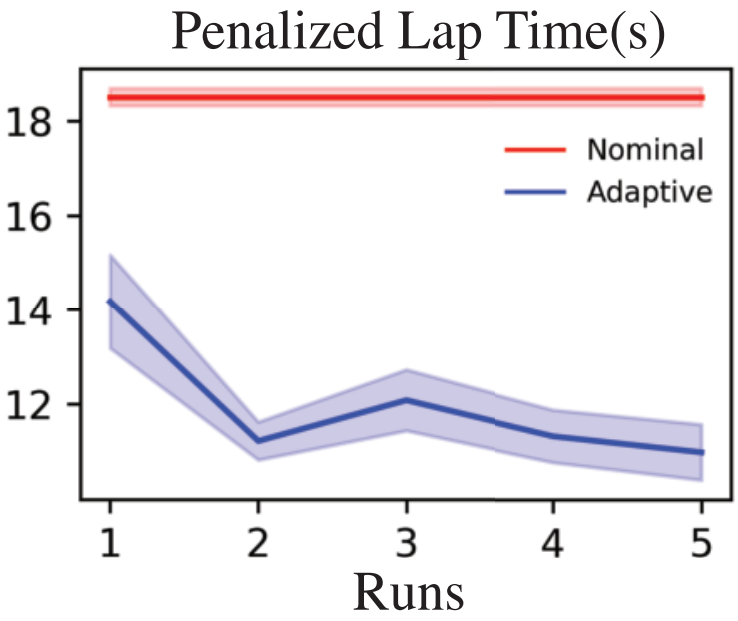


Figure 4.5: Adaptive Model vs. Nominal Model for Sim2real. For the adaptive model, we show average lap time of different systems across each run. Given the nominal model's inability to adapt leading to no difference in method between runs, we plot the average across all runs and systems. Shaded region indicates standard error.

since the adaptive approach is shown to remain safe in low-data regimes, while continually improving as it collects more observation data across different real-world environments.

4.7 Discussion

Here, we note some benefits of using System Identification Transformer with Adaptive Dynamics Models observed in our experimental results. We also note some potential future extension that could help extend the capabilites of this framework.

4.7.1 Benefits of Method

Our approach in this chapter allows for effective robust controls in scenarios where target system observations is extremely limited, or even non-existent. In our experiments, both in simulation and in the real world, we demonstrate the effectiveness of our approach in

safely driving unseen systems right from the initialization, with zero state-transition observations. The robustness comes as a result of capturing uncertainty in predictions and using an uncertainty-aware decision making process (Risk-Aware MPPI). While we also took the approach of capturing uncertainty and using a robust decision making framework in Chapter 2, the prior framework was unable to effectively control the robot when target system observations were extremely limited. The drastic increase in sample efficiency for this new framework stems from the use of large training datasets collected on a diverse range of simulated systems. In this framework, we assume a general understanding of the robot's morphology, but not its specific. For example, we might know that system is a wheeled robot with front and rear steering, but not know the exact frictional forces of its tires. During training, we sample different variations of this morphology and train the System Identification Transformer to capture an understanding of the target system's specifics and the adaptive dynamics model to probabilistically predict state-transitions given the understanding. While the framework from Chapter 2 viewed epistemic uncertainty as uncertainty in neural network parameters stemming from insufficient data, this new framework effectively views epistemic uncertainty as uncertainty in the target system's specifics within the known morphology. In scenarios where target system observations are extremely limited, dynamics model predictions still carry a higher level of epistemic uncertainty. However, they maintain a basic understanding of the robot's morphological characteristics, enabling them to be effectively utilized in decision making.

Furthermore, our method facilitates continual model adaptation and thus policy enhancements as the robot operates and accumulates more state-transition observations. This adaptability stems from the attention mechanisms in the System Identification Transformer, which can process variable-length observations and focus on pertinent segments of extended sequences to distill insights about the system's dynamics. In our experiments, as more state-transition observations were gathered, our method exhibited better performance, indicating its adaptability to the particular dynamics of the target system. This adaptability was par-

ticularly evident in trials where observations were collected across different tracks. Each track appeared to enrich the system's understanding, subsequently improving the resulting policy's performance. This improvement in policy performance can be seen as a result of reduced epistemic uncertainty. As more observations are available, the System Identification Transformer is able to gain a clearer understanding of the target system's particular dynamics. Thus, decision making can better tailor the driving strategy to the exact capabilities of the target system.

4.7.2 Future Work

One notable limitation of our approach in this chapter is the assumption of static system dynamics. We assume that all state-transition observations collected on the target system are a result of driving under the same system dynamics. Furthermore, we assume that these system dynamics do not change in the future. However, in real-world settings, a robot's dynamics can often change due to factors such as encounter new types of terrain, degradation of hardware, or change of payload. A relatively simple extension to this framework would be in-situ detection of changes in system dynamics, and subsequent re-initialization of the adaptation process, i.e. restarting the set of state-transition observations $\mathcal{H}$ and collecting new observations. Future work could also potentially implement a more sophisticated alternative to the System Identification Transformer that actively predicts how the system's dynamics might change using onboard sensors such as RGB cameras potentially combined with pre-existing terrain segmentation frameworks [12, 29, 30].

Another potential concern for this framework is the requirement for diverse simulation training data. This framework does not assume that any sampled simulated system matches the real-world system exactly. However, it does assume sufficient diversity within the training systems to encompass the range of potential real-world systems. Furthermore, the performance of the System Identification Transformer is reliant on the diversity of observations seen on training systems. This is relatively easy for the simple task of driving over flat-terrain

as the variety of possible observations are somewhat limited. However, more complex tasks might have a wider range of potential observations. For example, for the task of driving over obstacles, real-world observations might include interactions with a wide range of differently shaped objects. This increased complexity can be better seen and is discussed further in Chapter 5. Careful consideration is required to ensure enough diversity in simulation training dataset.

4.8 Conclusion

In this chapter, we introduce our sim2real transfer framework that balances robustness with adaptability. Our approach trains two neural network models, the System Identification Transformer (SIT) and the Adaptive Dynamics Model (ADM), in simulation while randomizing simulation dynamics parameters. The SIT leverages attention mechanisms to distill state-transition observations collected on the target system into a context vector, which succinctly encodes knowledge about the particular system's dynamics. The ADM predicts state-transition distributions given the robot's current state, action, and context vector from the SIT. Together, the SIT and ADM capture a probabilistic understanding of a target system's dynamics from state-transition observations on the target system. In real-time, our framework utilizes MPPI combined with a CVaR cost to safely control the system under its current understanding of dynamics.

The experimental results in this Chapter show that leveraging data from many simulated systems can heavily reduce the requirement for real-world training data. We show that even upon initialization, when there is no real-world data, the dynamics model can still result in safe, but low performing, driving policies. Furthermore, the model can be tailored to the target system, resulting in increased performance, with minimal amounts of data from the target system. This addresses the first key question posed in Chapter 1,"Can leveraging low quality simulation data provide insights into real-world vehicle dynamics and reduce the

amount of real-world data needed?"

Furthermore, the quantification of modeling uncertainty in this framework is a vital component to answering the second key question posed in Chapter 1, "Can quantifying uncertainty in an imperfect dynamics model improve long-horizon decision making?" We will demonstrate this further in Chapter 5.

In this chapter, we opted for the more straightforward task of driving along a flat-terrain track for our experiment setup to demonstrate the capabilities of our approach. For this purpose, we used risk-aware Model Predictive Path Integral (MPPI) as our decision-making strategy, given its suitability. However, it's important to note that the modeling approach and the benefits derived from this chapter are not confined to this task alone. This approach is broadly applicable to any uncertainty-aware decision-making process. Therefore, the insights and methodologies developed here can be extended to a wide range of tasks that demand a balance of robustness and task performance. In Chapter 5, we combine this modeling approach with insights from Chapter 2 to form an adaptive rough terrain navigation framework.

Chapter 5

Robustly Adapting Non-Myopic Navigation to New Dynamics

Off-road navigation requires the robot to drive at the edge of its physical capabilities while considering the obstacles ahead to avoid reaching dead-ends. The robot's decision making must be non-myopic and choose long-horizon driving routes that avoid non-traversable obstacles on the way to the goal. At the same time, decision making must be aware of the robot's dynamic capabilities to understand what obstacles can be traversed and decide how the robot should traverse them. This is a challenging task as the vehicle's dynamics can be highly complex and difficult to model. We argued in Chapter 1, that, in many cases, it is impossible to perfectly model the system's dynamics and model predictions will inevitably have some prediction error.

In Chapter 2, discussed our model-based reinforcement learning framework for rough terrain navigation. In this framework, we trained a terrain-aware dynamics model using real-world data to approximate the system's dynamics when driving over uneven terrain. With the understanding that this model will never be perfect, we quantified aleatoric and epistemic uncertainty in the model's predictions. This framework then used a closed-loop divergence constraint during decision making to avoid trajectories that could possibly be problematic

for the low-level trajectory tracker given the estimated potential trajectory prediction errors. This framework enables long-horizon decision making under highly complex rough terrain driving dynamics.

This decision making framework is robust towards epistemic uncertainty, which stems from having insufficient training data. Thus, we've shown that it could effectively navigate the robot even given limited training data collected offline and off policy (with a human driver a priori). However, when real-world training data is extremely limited, the extreme amounts of epistemic uncertainty will still render the model useless for decision making.

In Chapter 4, we offered a different perspective on epistemic uncertainty, supported by the availability of cheap simulations. Instead of viewing epistemic uncertainty as uncertainty in neural network model parameters, it could be viewed as uncertainty about the exact specifics of the true system within a domain of systems. We described a framework where different system variations within the given system morphology were simulated and used to train a system identification transformer and adaptive dynamics model. Given whatever limited observations are available from the target system, the system identification transformer extracts a succinct latent context vector that encodes an understanding about the particulars of the given system. This context vector is fed into the adaptive dynamics model to tailor its predictions to the target system. The probabilistic predictions from the dynamics model then capture not only aleatoric uncertainty inherent to the system, but also epistemic uncertainty stemming from having insufficient observations to properly identify the specifics of the target system.

This alternative dynamics modeling approach and perspective on epistemic uncertainty has a number of benefits when combined with an uncertainty-aware decision making process. In situations where real-world data is extremely limited, the resulting dynamics model still provides useful predictions, although with higher levels of uncertainty. In fact, the robot can be safely controlled even with no real-world observations, albeit at a sacrifice of performance. Furthermore, as more observations are collected for the target system, the dynamics model

and resulting policy becomes more tailored towards the target system, resulting in higher performance.

In this chapter, we extend and combine concepts from prior chapters to form a robust and adaptive rough terrain navigation framework via training in a diverse set of simulated systems. This framework is a culmination of the concepts discussed throughout this dissertation. We demonstrate that this framework: 1) can perform long-horizon dynamics-aware decision making, 2) is capable of adapting to the dynamic capabilities of a new target system, and 3) is aware of and robust towards uncertainty stemming from insufficient target system observations.

In Section 5.1, we discuss an approach to training an adaptive terrain-aware dynamics model that can be used for closed-loop trajectory prediction while capturing prediction uncertainty. In Section 5.2, we discuss how the divergence constraint from Chapter 2 can be used within an RRT planning framework [55]. In Section 5.5, we discuss the findings from the results and potential future improvements. Finally, we make some concluding remarks in Section 5.6.

5.1 Adaptive Closed-Loop Terrain-Aware Predictions

Rough terrain can have a drastic impact on the vehicle's driving dynamics. For example, a robot may be capable of driving down stairs, but incapable of driving up them. Furthermore, the exact properties of the robot and the type of terrain it is driving on also all effect the vehicle's driving capabilities. For example, low friction tires or loose soil may inhibit its ability to drive up inclines. Building upon concepts from Chapter 4, we train an Adaptive Dynamics Model (ADM) and System Identification Transformer (SIT) on a wide range of simulated dynamics. These two networks work in conjunction to adapt dynamics predictions towards a particular target system given state-transition observations from the target system. Furthermore, dynamic predictions capture epistemic uncertainty stemming from insufficient

target system observations for proper system identification, as well as aleatoric uncertainty inherent in the system. Building upon concepts from Chapter 2, we use a terrain-aware architecture for both the ADM and SIT. We also train the ADM to capture closed-loop dynamics, that is predict the robot's state transitions given different reference trajectories and the robot's low-level trajectory tracker's capabilities.

5.1.1 Terrain-Aware System Identification Transformer

Following the framework from Chapter 4, we train a System Identification Transformer to extract relevant information about the target system's dynamics given observations collected as the system operates. Similar to the prior framework, observations consists of an arbitrary length set of robot states s_t, actions a_t, and resulting state transition s'_t. However, in this framework, we also aim to extract information about the effects that uneven terrain has on the particular target system. As such, we provide terrain heightmap observations to the SIT.

Following the framework from Chapter 2, we transform all the System Identification Transformer inputs into a robot-centric form for sample efficiency. For each individual observation (s_t, a_t, s'_t), the corresponding robot-centric observation consists of a robot-centric form of the robot state, denoted as s^l_t, a robot-centric terrain heightmap, denoted as m^l_t, the robot action a_t, and the state transition s'_t. Following Chapter 2, we assume that the robot state s_t includes a global terrain heightmap from which m^l_t is generated. Note that the robot action, corresponding to the throttle and steering, is already inherently in a robot-centric form. Similarly, state-transitions were inherently in a robot-centric form as they consisted of the robot's next state s_{t+1} relative to the robot's current frame at state s_t. In our implementation, we also include the prior state-transition s'_{t-1} in each observation to provide the System Identification Transformer with insight about the robot's current velocity as it effects the robot's driving capabilities. For example, the robot may be capable of traversing a slippery incline depending on if it built up momentum.

For the SIT's neural network architecture, we use a combination of Convolutional Neu-

ral Networks [89] and Self Multi-head Attention Networks [133]. For each robot-centric observation, we first process the robot-centric heightmap using a shallow ResNet like architecture [134], where each layer consists of CNNs with a residual connection. The output of the ResNet is concatenated with the rest of the robot-centric observations (robot state, action, and state-transition). All of the processed robot-centric observations are then fed into an attention network. In this framework, we opt for a simpler self attention network rather than the transformer encoder used in Chapter 4. We use a single multi-head self-attention network. Similar to before, we opt not to use positional encoding for the observation input sequences. All outputs of the multi-head self attention network are then aggregated by taking their mean, resulting in a single context vector c.

Similar to before, this context vector c succinctly encapsulates an understanding of the target system's dynamics informed by whatever state-transition observations are available from the target system. Downstream, probabilistic predictions from the ADM are conditioned on this context vector to adapt the model to the current understanding of the target system's dynamics.

5.1.2 Closed-Loop Terrain-Aware Adaptive Dynamics Model

We train an Adaptive Dynamics Model that captures the effects of uneven terrain as well as the effects of the low-level trajectory tracker. In Chapter 2, we trained a dynamics model that predicted state-transition probability distributions given the robot states, actions, and terrain heightmap. In that framework, we captured the effects of the low-level trajectory tracker by assuming closed-form knowledge of the low-level trajectory tracker in the form:

$$a_t = \bar{a}_t + f_{track}(s_0, ..., s_t, \bar{d}), \tag{5.1}$$

where $s_0, ..., s_t$ are the robot's actual states, and $\bar{d} = (\bar{s}_0, ..., \bar{s}_T)$ 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.

However, we acknowledge that this assumption may not always hold. For example, if an optimization based trajectory tracker is used, then f_{track} may not be a closed-form equation and may require significant computational resources to compute. Or, more complex trajectory trackers may require additional observations, such as RGB-D images, which are unavailable given only states $s_0, ..., s_t$. Here, we take an alternative approach to modeling the closed-loop effects of the low-level trajectory tracker to generalize the methods to a wider range of trajectory tracking controllers. Instead of conditioning state-transition predictions on actions, we directly condition predictions on the reference trajectory directly. Later in Section 5.2, we discuss how this transforms decision making from a search in the nominal action space to a search in the reference trajectory space.

The Adaptive Dynamics Model in this framework, denoted as

$$\mathcal{P}_\theta(s_{t+1}|s_t, \bar{d}^l_{s_t}, c_t), \tag{5.2}$$

predicts a probability distribution of the robot's next state s_{t+1}, given its current state s_t, context-vector c_t from the System Identification Transformer described in Section 5.1.1, and the reference trajectory of the robot's low-level trajectory tracker.

As with the SIT network, we transform all ADM inputs into a robot-centric form for better sample efficiency. Just like the dynamics model from Chapter 2, the ADM's inputs includes a robot-centric form of the robot's state, denoted as s^l_t, and a robot-centric heightmap, denoted as m^l_t. In our prior framework (Chapter 2), we conditioned dynamic model predictions on actions (throttle and steering) and calculated the feedback action using Equation 5.1 during trajectory prediction to capture the effects of the low-level trajectory tracker. In this framework however, we directly condition predictions on the reference trajectory. At each time step during trajectory prediction, we use the predicted robot state $\hat{s}_t$

to transform the reference trajectory $\bar{d}$ into a robot-centric form $\bar{d}^l_{\hat{s}_t}$, which is then fed into the ADM. We denote this mapping function as $\bar{d}^l_{s_t} = \mathcal{RC}(\bar{d}, s_t)$.

The output of the neural network is the mean μ and covariance matrix Σ corresponding to the multivariate Gaussian distribution estimate of the robot's state transition, or robot's s_{t+1} relative to its state s_t. More specifically, the covariance component of the neural network output is the off-diagonal and positive diagonal terms of a lower triangular matrix L, where $\Sigma = LL^T$. By sampling from this distribution at each time step and chaining individual time step predictions, the ADM can be used to predict the robot's trajectory distribution as it drives over uneven terrain tracking a reference trajectory. Furthermore, the predicted trajectory distribution can be adapted to the target system by providing state-transition observations. This trajectory distribution prediction process is summarized in Algorithm 7.

Algorithm 7: Adapted Closed-Loop Trajectory and Divergence Prediction

Input : Reference Trajectory: $\bar{d} = (\bar{s}_0, ..., \bar{s}_T)$
Initial State Distribution: $\hat{s}^1_0, ..., \hat{s}^I_0$
Target System Observations: $\mathcal{H}_0$

Output: Predicted Trajectory Distribution: $\hat{D} = (\hat{d}^1, ..., \hat{d}^I), \hat{d}^i = (\hat{s}^i_0, ..., \hat{s}^i_T)$
Predicted Divergence: u

```
// Infer context from target system observations
```
$c_0 \leftarrow \mathcal{T}_\theta(\mathcal{H}_0)$

```
// Predict trajectory distribution
```
$u \leftarrow 0$

for $t \in [0, T-1]$ **do**

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

    ```
    // Transform reference trajectory to be robot-centric
    ```
    $\bar{d}^l_{\hat{s}^i_t} \leftarrow \mathcal{RC}(\bar{d}, \hat{s}^i_t)$

    ```
    // Calculate state-transition distribution and sample next state
    ```
    $\hat{s}^i_{t+1} \sim p_{\theta^i}(s_{t+1}|\hat{s}^i_t, \bar{d}^l_{\hat{s}^i_t}, c_0)$

  ```
  // update divergence
  ```
  $u \leftarrow \max(u, \frac{1}{I}\sum_{i=1}^{I}\|\hat{s}^i_t - \bar{s}_t\|_2^2)$

5.2 Divergence Constrained Planning

In our prior decision making framework, we used a divergence constraint to prevent the optimizer from exploiting modeling errors. This divergence constraint was defined as

$$u(\bar{d}, \hat{D}) < U_{max}, \tag{5.3}$$

where U_{max} is a hyperparameter which is set based on the capabilities of the low-level trajectory tracker and u is the divergence metric. The divergence metric is defined as

$$u(\bar{d}, \hat{D}) = \max_t \frac{1}{I} \sum_{i=1}^{I} \|\hat{s}_t^i - \bar{s}_t\|_2^2, \tag{5.4}$$

where $\bar{d} = (\bar{s}_0, ..., \bar{s}_T)$ is the reference trajectory and $\hat{D} = \{\hat{d}^0, ..., \hat{d}^I\}$ is the predicted closed-loop trajectory distribution, where $\hat{d}^i = (\hat{s}_0^i, ..., \hat{s}_T^i)$ is a single trajectory roll out predicted using the dynamics model. Satisfying this divergence constraint ensures that the robot robot's true state is unlikely to deviate too far from the reference trajectory, meaning that the low-level controller can effectively track the reference trajectory. The process for calculating divergence metric during closed-loop probabilistic trajectory prediction is shown in Algorithm 7.

This divergence constraint was previously used within a constrained trajectory optimization framework utilizing a gradient-based Augmented Lagrangian optimizer [139]. However, in Chapter 2, we noted a few challenges of this constrained trajectory approach. First, this optimizer, based on gradient descent, was prone to local minima issues, resulting in poor solution trajectories that were locally optimal but did not reach the goal. In our prior work, we mitigated this issue by executing multiple optimizations simultaneously, each initiated with distinct random seeds, in the hope that at least one would yield a good solution. Second, calculating gradients can be computationally intensive, especially for longer horizon trajectories since individual time step predictions are chained along the horizon.

While previously we opted to use the divergence constraint within constrained trajectory optimization, this divergence constraint can also be used to inform other decision making frameworks about the robot's dynamics and capabilities. In this section, we describe how this divergence constraint can be used within a Rapidly exploring Random Trees (RRT) planner [55] to find a feasible path over rough terrain to the goal. This approach is similar to Convergent Planning [65], where kinodynamic RRT is guided by divergence metrics. Unlike trajectory optimization, RRT can explore large areas quickly and offers probabilistic completeness so that the probability of finding a feasible path to the goal approaches 1 as the number of search iterations increases, given that a solution exists. While RRT offers probabilistic completeness guarantees, the solution found is unlikely to be optimal. In this application, we are primarily concerned with finding a path to the goal that is feasible (satisfies the divergence constraint), and less concerned with optimality. However, in our approach, we perform path smoothing after the RRT search process to improve optimality of the final solution, similar to other approaches [140–142].

5.2.1 Divergence Constrained RRT

In RRT, a search tree, rooted at the initial state, grows by selecting random points in the search space and connecting them to the nearest existing node in the tree, while ensuring that these connections satisfy a collision check to avoid obstacles in the environment. RRT is normally used in environments where obstacles are well defined, such as environments with walls, other vehicles, or pedestrians. For rough terrain navigation, what objects are considered an obstacle are not as well defined, but depends on the robot's capabilities as well as how it approaches the object. Instead of traditional collision detection, our RRT approach uses the divergence constraint to inform whether a trajectory is likely to cause failure when executed.

During each iteration of RRT search, we sample a random state s_{rand} within the configuration space or choose the goal with probability p_{goal}. We then choose the closest node

on the existing search tree s_{near} and generate a reference trajectory of max length T_{expand} from s_{near} to s_{rand} using a steer function denoted as $\bar{d} = Steer(s_{near}, s_{rand}, T_{expand})$. In our implementation, we used a pure pursuit based steer function, as described in Section 5.3 to generate a trajectory from s_{near} to s_{rand}, but the choice of steer function is not unique and may be robot dependant. We then use Algorithm 7 to predict the robot's trajectory distribution and divergence as it tracks this reference trajectory. If the predicted divergence satisfies the constraint from Equation 5.4, we add the last state of the reference trajectory s_{new} as a new node to the search tree with the associated edge (s_{near}, s_{new}).

Note that given the sequential nature of trajectory prediction and divergence evaluation, we find that it is important to reinitialize trajectory prediction (Algorithm 7) with the previously predicted state distribution. In our implementation, we also include the predicted last state distribution whenever adding a new new node to the search tree. If this new node is then expanded in the future, we reinitialize trajectory prediction with the saved last state distribution.

Once a node that matches the goal is added to the search tree, the planner then works backwards through the tree, concatenating all reference trajectories from parent nodes to children nodes until the initial root node is found. The result of this process is a continuous reference trajectory from the start state to the goal that satisfies the divergence constraint. Thus, if the robot tracks this trajectory with its low-level trajectory tracker, it should be able to traverse any obstacle along the path and reach the goal. The Divergence Constrained RRT search process is depicted in Algorithm 8. In Section 5.3, we provide details on our implementation the setup of our experiments. In Section 5.4, we provide some qualitative results illustrating the capabilities of this framework.

5.2.2 Divergence Constrained Trajectory Pruning

While Divergence Constrained RRT (Algorithm 8) results in a solution reference trajectory to the goal that satisfies the divergence constraint, if one exists, its solutions are often

Algorithm 8: Divergence Constrained RRT Search

Input : Starting State: s_{start}
Goal Location: s_{goal}
Target System Observations: $\mathcal{H}$
Output: Solution Trajectory: $\bar{d}_{RRT}$

```
// Initialize tree root and initial prediction distribution at s_start
```
$z_{new} \leftarrow \text{NewNode}(s_{start}, \hat{D}_{last} = (s_{start}, ..., s_{start}))$
$T \leftarrow \text{InitializeTree}(z_{new})$
while $z_{new} \neq goal$ **do**
```
    // Generate reference trajectory to random state
```
    $s_{rand} \leftarrow$ RandState() or $goal$ with probability p_{goal}
    $z_{near} = (s_{near}, \hat{D}_{last}) \leftarrow \text{Nearest}(T, s_{rand})$
    $\bar{d} \leftarrow \text{Steer}(s_{near}, s_{rand}, T_{expand})$
```
    // Predict Trajectory Distribution & Divergence with Algorithm 7
```
    $\hat{D}, u \leftarrow \text{Predict}(\bar{d}, \hat{D}_{last}, \mathcal{H})$
```
    // Add new node to tree if divergence constraint satisfied
```
    if $u < U_{max}$ **then**
        $s_{new}, \hat{D}_{last} \leftarrow \text{LastTimeStep}(\bar{d}, \hat{D})$
        $z_{new} \leftarrow \text{NewNode}(\bar{s}_{new}, \hat{D}_{last})$
        $T \leftarrow \text{InsertNode}(T, z_{near}, z_{new})$

```
// Work backward through tree to find solution reference trajectory
```
$\bar{d}_{RRT} \leftarrow \text{Backward}(T, z_{new})$

suboptimal. The issue of suboptimal solutions stems from the use of standard Rapidly-exploring Random Tree Algorithm. RRT* addresses the problem of suboptimality by adding a rewiring step that checks if new nodes added to the search tree may serve as a more optimal parent to neighboring nodes in the tree [143]. Unfortunately, we found that extending Divergence Constrained RRT to RRT* was non-trivial due to the sequential and continuous nature of trajectory prediction an divergence evaluation.

Given two reference trajectories, where both trajectories satisfies the divergence constraint individually and the first trajectory ends at the starting state of the second trajectory, combining the two trajectories into one continuous reference trajectory may result in a trajectory that does not satisfy the divergence constraint. In some cases, the divergence evaluation at the end of the first trajectory may be nearing, but not past, violation. If

trajectory prediction and divergence evaluation for the second trajectory is incorrectly reinitialized with the starting state of the second trajectory, the divergence captured from the first trajectory will be lost and reset to zero. Divergence for any given trajectory must be evaluated in the trajectory's entirety.

Instead of using RRT*, we opted to improve the solution trajectory from Divergence Constrained RRT using random pruning. In this approach, we iteratively improve upon a solution trajectory by sampling a random subtrajectory and attempting to replace it with a better subtrajectory. For each sampled subtrajectory, we generate multiple candidate replacements. Out of all the candidates and original subtrajectory, we choose the shortest replacement such that the resulting complete solution trajectory still satisfies the divergence constraint. Furthermore, with probability p_{goal}, we choose a random state within the solution trajectory and generate a direct reference trajectory from the chosen state to the goal. We replace the subtrajectory after the chosen state with the direct path to the goal if the resulting trajectory satisfies the divergence constraint. This process is depicted in Algorithm 9.

5.3 Simulation Experiment

We demonstrate the capabilities of this robustly adapting rough terrain navigation framework in simulation. In our experiments, different variants of a wheeled robot are tasked with driving over rough terrain to the goal. For each variant, we evaluate whether the robot can gain an understanding of its specific rough terrain traversal capabilities and align its navigation strategy to match. Furthermore, we evaluate if the robot is capable of adopting a more conservative navigation strategy in situations where observations from the specific system are virtually non-existent.

Similar to Chapter 4, we use a four wheel drive vehicle with independent front and rear steering and vary robot parameters to generate different system variants. For simplicity however, we primarily vary tire traction in this implementation and keep other parameters

Algorithm 9: Divergence Constrained Trajectory Pruning

Input : Initial Solution: $\bar{d}_{RRT}$
Goal Location: *goal*
Target System Observations: $\mathcal{H}$
Output: Refined Solution: $\bar{d}_{prune}$

$\bar{d}_{prune} \leftarrow \bar{d}_{RRT}$
for *number of prune iterations* **do**
  `// Choose random segment to replace`
  $t_s, t_e \leftarrow \text{RandomWindow}(\bar{d}_{prune})$
  $\bar{d}_{best} \leftarrow \bar{d}_{prune}[t_s : t_e]$
  `// Iterate through possible replacements to find best option`
  for $\bar{d}_{sub} \in \text{CandidateSubTrajs}(\bar{d}_{prune}[t_s], \bar{d}_{prune}[t_e])$ **do**
    `// Predict divergence of resulting complete trajectory`
    $\bar{d}_{cand} \leftarrow \text{Substitute}(\bar{d}_{prune}, t_s, t_e, \bar{d}_{sub})$
    $u \leftarrow \text{EvalDivergence}(\bar{d}_{cand}, \mathcal{H})$
    `// Choose substitute if feasible and more optimal`
    if $u < U_{max}$ ***and*** $\text{Cost}(\bar{d}_{sub}) < \text{Cost}(\bar{d}_{best})$ **then**
      $\bar{d}_{best} \leftarrow \bar{d}_{sub}$
  $\bar{d}_{prune} \in \text{Substitute}(\bar{d}_{prune}, t_s, t_e, \bar{d}_{best})$

mostly fixed. However, this variation in tire traction results in a drastic difference in the robot's ability to traverse different obstacles. The environments that the robot is tasked with navigating through are shown in Figure 5.1 and Figure 5.2. For simplicity, we kept the environment mostly fixed during training and evaluation, with only small amounts of Perlin noise added to the terrain.

Data Collection & Model Training Our approach to data collection and model training follows closely to that of Chapter 4. We use a distributed training architecture, where training data is generated asynchronously while the System Identification Transformer and Adaptive Dynamics Model is trained. For each system, we generate training data by driving the robot to random goal configurations in the environment using the steer function to generate a reference trajectory and the trajectory tracker to execute it. We collect the following training data: robot's states (x, y, heading angle, prior state transition), robot's actions (throttle,

front steer, rear steering), world terrain map, reference trajectory, robot state transitions (x, y, heading angle change relative to prior body frame). For each system, we collect 256 time steps of data, resetting the robot whenever it reaches the end of the reference trajectory, or upon failure, i.e. robot flips over or deviates too far from the reference trajectory.

During model training, we iteratively sample a system variant from the collected data. We train both the System Invariant Transformer and the Adaptive Dynamics Model end-to-end. From the 256 time steps of data, we choose a 32 continuous time step trajectory to predict with the Adaptive Dynamics Model. We provide the System Invariant Transformer with all the data collect prior to that randomly chosen 32 time step trajectory, thus varying the amount of data given to the System Invariant Transformer.

We used an Adam Optimizer [117] to optimize the neural networks. Similar to [133], we linearly increased the learning rate initially during a warm up period then decreased it proportionally to the inverse square root of the training iteration count. For our implementation, we used a warm up period of $10,000$ training steps, where the learning rate increased up to 0.001.

Pure Pursuit Trajectory Tracker For the robot's trajectory tracker, we use a pure pursuit trajectory tracker [144], but adapted for a vehicle with all-wheel steering. In the standard formulation of a pure pursuit path tracker for an Ackermann steered (front-wheel steer only) vehicle, the vehicle chases a look-ahead point on the desired path. The steering angle of the vehicle is chosen such that the turn arc of the vehicle passes through that point.

For a front-wheel steered vehicle, the vehicle's heading is always tangent to the turn arc, so the vehicle has no control over what orientation it reaches the look-ahead point at if it only holds its steering constant. For a all-wheel steered vehicle however, the robot can control its orientation by changing the ratio of steering angle for the front and rear wheels. This increased maneuverability gives the robot the ability to reach a desired look-ahead point at a desired heading angle.

In our pure pursuit trajectory tracker implementation, the reference trajectory is defined as a sequence of desired poses. During tracking, the robot chooses a look-ahead pose based on a predetermined time horizon. It then chooses steering angles (front and rear) such that if held constant, the chosen steering angles will guide the vehicle to the look-ahead pose. For throttle control, we calculate the arc length between the robot and the look-ahead pose and use this distance as an error input to a proportional (P) controller.

This variant of pure pursuit gives the high level path planner additional control of the vehicle. By choosing an appropriate trajectory, it result in behaviors such as strafing (crab steering), or tight turning.

Steer Function The steer function generates a trajectory to either a pose (x, y position and heading angle) or a location (just x, y position). It was used during to RRT and during data collection to drive the robot to random poses. For generating a trajectory to a pose, we used a similar approach to that of the pure pursuit trajectory tracker. We calculated the front and rear steering angle needed such that if robot drove holding the steering angles constant, the robot would reach the desired pose. We then discretized the expected trajectory of the robot if it held these two steering angles constant and drove at a constant speed.

For steering to a location, the path consisted of a tight turn, where the robot's heading was aligned with the goal location at the end of the turn, followed by a straight line path to the goal. The turn radius of the tight turn was defined by the tightest turn the robot was capable of when steering both front wheels and rear wheels together.

Candidate Subtrajectories for Pruning During trajectory pruning (Algorithm 9), we randomly choose a window in the trajectory and attempt to replace the subtrajectory with a more optimal one. During this process, multiple candidate subtrajectories are generated and evaluated for cost and feasibility. All candidate subtrajectories start and end at the same state as the original subtrajectory. We use two options to generate candidate subtrajectories: The steer function described previously, and the Dubins path [145] given the turn capabilities

of the robot.

5.4 Simulation Results

We evaluated the navigation framework's ability to 1) choose robust strategies when unsure about the target system's capabilities, and 2) adapt its strategy to the target system's capabilities given target system observations. We chose two systems for evaluation. One system had low traction, and could only drive up gentle slopes. On the other hand, the other system had high traction and could drive up steep slopes and even some stairs. Here, we qualitatively analyzed the framework's decision-making process and strategy adaptation.

For each test, we generated feasible trajectories to the goal using the divergence constrained RRT (Algorithm 8) and trajectory pruning (Algorithm 9), then tracked the trajectories using the robot's low-level trajectory tracker. For each system, we ran two trials. One trial before using the trained predictive model before any adaptation data was collected on the particular system. Then one trial after giving the System Identification Transformer data to adapt the model to the specific system. Adaptation data was collected through one short trial of the robot tracking a manually defined trajectory.

The initial trail is similar to Domain Randomization, where the robot's policy is trained to be robust across a wide spectrum of systems. Before the model is adapted to any particular system, its probabilistic predictions capture possible outcomes for a wide spectrum of systems. The divergence constrained RRT planner then finds a strategy that will be effective across this range of dynamics. On the other hand, after adaptation, the model and resulting driving strategy is tailored to the particular target system.

5.4.1 Driving Up Stairs

In the first test, the robots were tasked with driving to the top of a set of stairs, which were surrounded on both sides by ramps. Both systems were capable of driving up the ramp, but

only the high-traction system could drive directly up the stairs. While driving up the ramp is a safer route, driving directly up the stairs is a more direct and lower cost route. The results of the trials for both systems before and after adaptation are shown in Figure 5.1.

Before adaptation, the path planner chooses the more conservative path of driving up the ramp. For both systems, we collected adaptation data by having the robots track a trajectory going straight up the stairs. Notice that the high-traction system was able to climb the stairs and track the trajectory faithfully, whereas the low-traction system failed and got stuck at the bottom of the stairs. After adapting the dynamics model to the low friction system, the strategy chosen by the planner did not change much and still avoided going straight up the stairs. However, after adapting to the high friction system, the planner chose the more aggressive strategy of driving directly up the stairs, showing a gained understanding of the robot's capabilities. Notice that the conservative strategy of driving up the ramp still required the robot to drive across the stairs. This means that the planner must not simply avoid stairs. The model must understand that ascending stairs may be problematic, while descending or driving across them is feasible.

5.4.2 Slope Driving Given Stair Demonstration

As another simple example, we tested the approach in an environment again containing stairs surrounded by ramps. However, in this scenario, the low-traction system was unable to drive up the ramp sides. The robot was placed on one side of the obstacle with the goal on the other. The direct route to the goal involved driving up and down the ramp sides of the obstacle, which only the high-traction system was capable of. The safer, but less direct, route involved driving around the obstacle to the goal. The results of the trials for both systems before and after adaptation are shown in Figure 5.2.

Similar to the results before, the path planner chooses the more conservative path of driving around the obstacle for both systems. Again, we collected adaptation data by having the robots track a trajectory going up the stairs. Again, only the high-friction system was

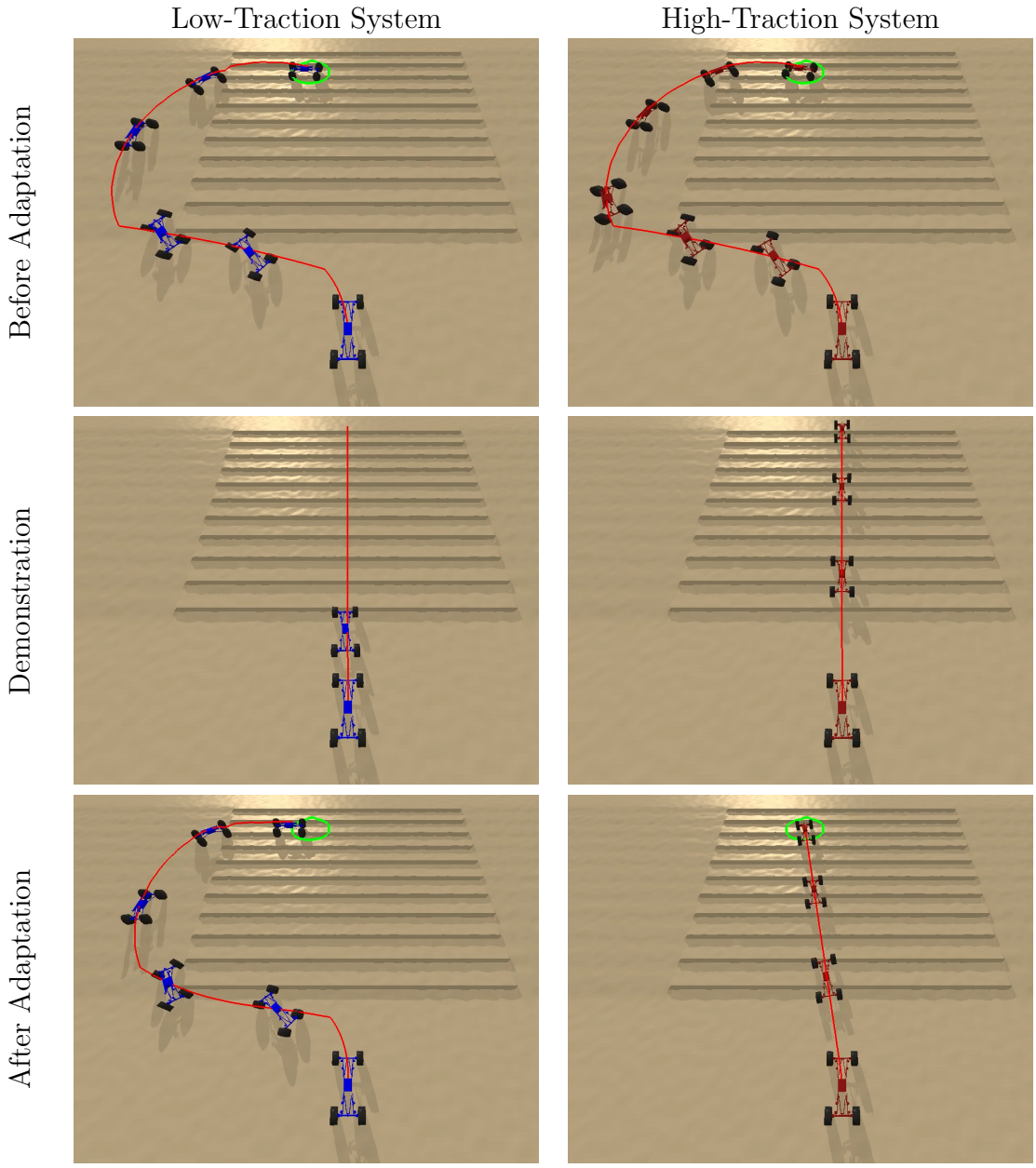


Figure 5.1: A low-traction robot and high-traction robot navigating a stair obstacle. Before adaptation, both systems choose a safe route that avoids driving directly up the stairs (top). The dynamics model was adapted to each system using data collected through one demonstration (middle). After adaptation, the planner matches the route to the capabilities of each system, with the high-traction system's route going directly up the stairs (bottom). The reference trajectory chosen by the planner or manually chosen (for demonstration) is shown in red. The goal is shown in green. Robot's actual path is overlaid.

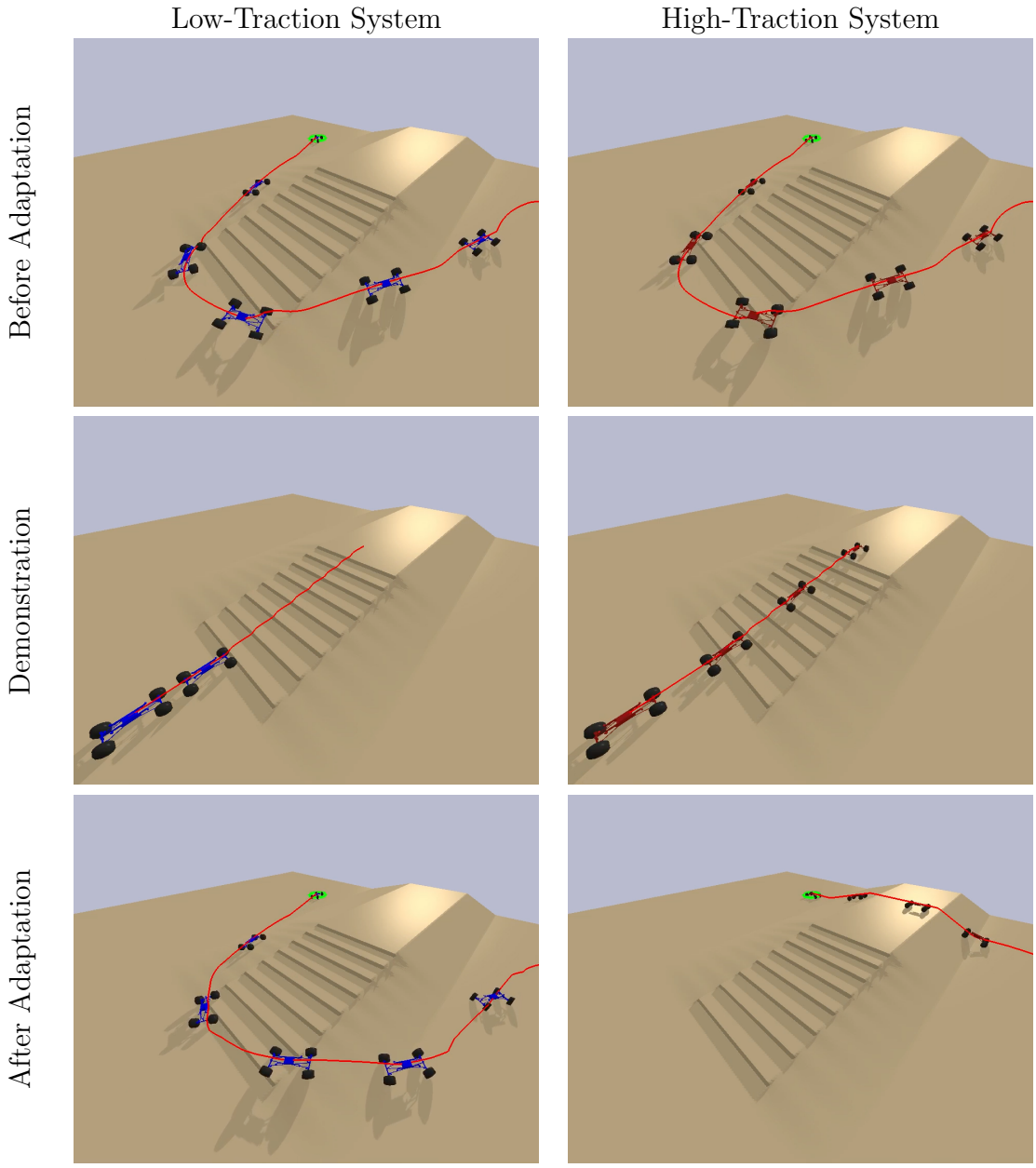


Figure 5.2: A low-traction robot and high-traction robot navigating a ramp obstacle. Before adaptation, both systems choose a safe route that avoids driving directly up the ramp (top). The dynamics model was adapted to each system using data collected through one demonstration (middle). After adaptation, the planner matches the route to the capabilities of each system, with the high-traction system's route going directly up the ramp towards the goal (bottom). The reference trajectory chosen by the planner or manually chosen (for demonstration) is shown in red. The goal is shown in green. Robot's actual path is overlaid.

able to track the trajectory faithfully and make it up the stairs. Again, after demonstration, the planner changed only the plan for the high-traction system to be more aggressive and direct. However, in this scenario, the model took an observation of the two systems on stairs and correctly predicted each system's capabilities on a different steep ramp obstacle. This suggests that the System Identification Transformer and Adaptive Dynamics Model are not simply memorizing obstacles, but gaining a generalized understanding of the system's capabilities.

5.5 Discussion

Strengths of Method

While the results in this chapter are drawn from relatively simple toy environments, they highlight some key advantages of the adaptive rough terrain navigation framework developed in this chapter. Note that many of these key advantages stem from combining concepts from previous chapters (Chapter 2 and Chapter 4).

First, this framework is capable of non-myopic dynamics-aware decision making. In situations where the dynamics model has determined that the robot may be incapable of driving over the obstacle, the path planner chooses the path around the obstacle despite it being longer and less direct. This demonstrates that this navigation framework is capable of non-greedy long-horizon decision making. Furthermore, the planner seems to be considering the robot's dynamics capabilities and not simply just avoiding certain obstacles. This can be seen in the stair example. If the planner just avoided the stairs completely, the robot would not be able to make it to the goal. Instead the planner seems to distinguish the differences in traction encountered when ascending the stairs, as opposed to descending or driving across them. In certain cases, it correctly chooses to avoid driving directly up the stairs, but not avoid driving across them to get to the goal.

This framework also allows for robust navigation given the current understanding of the

target system's particular dynamics and capabilities. In the trials, the planner chose the more conservative route upon initialization when target system observations are virtually non-existent and provide a vague understanding of the target system's particular capabilities. Furthermore, the understanding of the target system's capabilities is not static but improves as more observations are collected. This can be seen with the high-traction system. After collecting observations through one demonstration of driving up stairs, the planner changes its path to drive directly up the stairs, matching the robot's capabilities.

Furthermore, adaptations to the divergence constraint in this chapter make it relevant to other high-level planning frameworks and low-level planners. In this framework we show the divergence constraint used in an RRT-based decision making frameworks. We also show the divergence constrained using a different trajectory tracker (pure-pursuit). Although this tracker is still relatively simple, the modifications to the divergence constraint does open up many more possibilities. In Chapter 6, we delve more into the implications of these adaptations and potential future directions.

Current Limitations

Currently, the System Identification Transformer trained is only effective within the two environments shown in Figure 5.1 and Figure 5.2. Ideally, the System Identification Transformer could effectively extract an understanding of the system's capabilities through observing the system interacting with any type of obstacles.

However, this would require training the System Identification Transformer with a large dataset containing a diverse set of target system observations ($\mathcal{H}$) with many different combinations of obstacle interactions, along with a diverse set of simulated dynamics. To simplify the problem, we used only a couple environments for training and testing to limit the number of obstacles the robot might encounter.

When we tried training the System Identification Transformer by randomizing the terrain, following the approach in Chapter 2, the System Identification Transformer was unable to

properly distinguish between different systems. This could be perhaps due to the sparsity of relevant information in the observation set $\mathcal{H}$ during training. Or perhaps this approach just required more data, a larger neural network architecture, and more engineering effort.

Another limitation of our current planning framework is its inability to vary velocity, thus not fully exploiting the vehicle dynamics potential, as could be achieved with the optimization framework discussed in Chapter 2. Consequently, the robot is restricted from performing dynamic behaviors, such as accelerating to overcome larger obstacles. In our current implementation, the RRT planner's state space includes only the robot's position and orientation, with the steer function generating trajectories at a constant velocity between states. Theoretically, this framework could be extended to incorporate velocity variations, enhancing the robot's dynamic maneuverability. This would involve extending the state space to include the robot's velocity and extending the steer function to interpolate velocities between states. This would significantly improve the planner's capability to navigate more difficult environments dynamically.

Another potential extension to the framework is to incorporate safety constraints into the planner. In the current framework, the planner only constrains solutions to satisfy the divergence constraint, which mainly focuses on dynamic feasibility given prediction uncertainty. In practice, there may be other important considerations such as vehicle rollover [146]. In the future, the planner can also incorporate other important safety constraints, such as the track boundary or later acceleration constraints from Chapter 4.

5.6 Conclusions

This chapter serves as the culmination of this dissertation, integrating concepts discussed in previous chapters into a cohesive rough terrain navigation framework. Following Chapter 4, this framework uses a large range of simulated systems to train a dynamics model that can quickly adapt to any new target system given limited target system observations. During

adaptation, the model captures both aleatoric uncertainty inherent in the stochastic system as well as epistemic uncertainty stemming from having insufficient target system observations to properly identify the specific target system's dynamics and capabilities. Following Chapter 2, this navigation framework uses a divergence constraint to find routes over rough terrain to the goal that are robust under the current probabilistic understanding of target system dynamics.

We demonstrated this navigation framework in two toy simulated scenarios and showed that it can tailor navigation strategies to the capabilities of different target systems while also remaining robust before adaptation. When no observations are available to adapt the model to the particular system, the navigation framework chooses conservative paths that can be effectively executed by a large range of systems. These initial routes mostly avoid obstacles choosing the longer path containing less risky terrain. However, the framework can also adapt to the particular target system and choose shorter routes over certain obstacles depending on the system's capabilities. We show that providing observations of a capable system driving up stairs resulted in more aggressive driving strategies that drove over obstacles, matching the system's capabilities.

Chapter 6

Conclusion

This dissertation tackles the challenges of rough terrain navigation for unmanned ground vehicles (UGVs). It primarily focuses on the factors that hinder these robot's ability to effectively make long-horizon decisions that utilize the dynamic capabilities of the robot, which is vital for navigating challenging rough terrain. In Chapter 1, we explored the difficulties of creating accurate dynamics model and introduced two pivotal questions driving this dissertation. Below we summarize our findings that begin to answer these questions.

Can leveraging low quality simulation data provide insights into real-world vehicle dynamics and reduce the amount of real-world data needed?

We started addressing this question by exploring if data collected on one simulated system could potentially be used to the increase accuracy of data-driven dynamics models for real-world systems when real-world data is limited.

In Chapter 3, we discussed our framework that trains dynamics models for a target system using large quantities of data from a proxy system and small quantities of data from the target system. This approach aimed to learn the underlying dynamics shared between the two systems and the target system's relationship to it. In our results, we found that the resulting dynamics model for the target system had higher prediction accuracy than if

a naive dynamics model was trained using only the limited amounts of target system data. This increase in model accuracy was more pronounced when the proxy system's dynamics were more similar to the target system's dynamics. The results from this chapter suggest that insights could be drawn from data collected on one simulated system (proxy) to improve the accuracy of dynamics models for a real-world system (target) when real-world data is limited.

While these results are promising, this approach is not without limitations. First, its benefits for sim2real applications is dependent on how similar the simulation is to the real-world system. This may require the tedious task of fine tuning of simulation parameters to match the real world. Furthermore, the resulting dynamics models are never perfect, especially when target system data is extremely limited. The deterministic nature of these models provide no indication into how imperfect model predictions might be.

After establishing the potential of using data from one simulated system, we then explored whether using multiple simulations could provide additional benefits. In Chapter 4, we discussed an improved sim2real framework that leverages a large variety of simulated systems, each with different dynamics but the same morphology, to train a System Identification Transformer and Adaptive Dynamics Model. These two models work in conjunction to quickly adapt predictions to any new target system by extracting an understanding of system dynamics from any available target system observations. Furthermore, this model captures uncertainty in predictions that is either inherent in the system or stems from having insufficient target system observations. We show that using these models within Risk-Aware MPPI allows for safe and conservative driving when no observations are available, and an increase in performance as more data becomes available. In our performance comparison, the baseline approach of training a dynamics model with only target system data was not able to complete the course without driving off track at 500 time steps of data. In comparison, our approach already converged to the performance limits of the vehicle, and rarely drove off track even when no target system data was available yet. These benefits are also shown for

the sim2real application where a simulation trained model was used to safely and effectively control different real-world systems. This shows that leveraging cheap simulations, that do not perfectly match the real-world, can help heavily reduce the need for real-world data. Furthermore, the insights drawn from simulation are enough to conservatively control the robot with no real-world data.

Can quantifying uncertainty in an imperfect dynamics model improve long-horizon decision making?

Our approach to addressing this question can be decomposed into two components: 1) quantify prediction uncertainty in the inevitably imperfect dynamics model, and 2) improve long-horizon decision making in the presence of modeling uncertainty.

In Chapter 2, we discussed our model-based reinforcement learning framework for rough terrain navigation that addresses both components. Addressing the first component, this framework trains a dynamics model to predict the robot's state transitions in rough terrain while capturing aleatoric and epistemic uncertainty in predictions. In this approach, we view epistemic uncertainty as uncertainty in neural network parameters stemming from insufficient training data. Addressing the second component, we manage the growth of uncertainty along longer-horizon trajectories by considering the effects of the low-level trajectory tracker during trajectory prediction. Furthermore, we introduce a divergence constraint to indicate when model predictions are unreliable, i.e. modeling errors may potentially cause the low-level tracker to fail. During decision making, we use this closed-loop divergence constraint to facilitate better long-horizon decision making while ensuring that solution trajectories could be effectively tracked, mitigating the potential of solutions exploiting modeling errors. We show in our experiments that this approach results in effective non-myopic driving strategies through obstacle field environments, despite using an imperfect dynamics model trained with limited data collected offline. The resulting driving strategies were able to take full advantage of the robot's dynamic capabilities. For example, they avoided ascending step obstacles, but

drove off similar height steps at speed, matching the vehicle's capabilities. Furthermore, we show quantitatively that this approach is far more reliable than if uncertainty in the model was ignored, which often resulted in solution trajectories that exploited modeling errors and resulted in failure when executed.

In Chapter 4, we provided an alternative view on epistemic uncertainty. We discuss a framework that leverages a variety of simulation, each with the same morphology but different dynamics, to train an adaptive model for the given morphology. Online, the model probabilistically adapts to a new target system's dynamics by extracting an understanding of the system's particular dynamics from any available observations. By assuming knowledge of the target system's morphology, this framework transforms epistemic uncertainty from uncertainty in neural network parameters stemming from insufficient data to uncertainty about the target system's specifics within the given morphology.

In Chapter 5, we combine this adaptive dynamics model and alternative view on epistemic uncertainty with the closed-loop divergence constrained navigation framework. We used this framework to drive a couple new target systems through toy obstacle filled environments. The resulting driving strategies matched the insights provided by available observations into vehicle dynamics. Upon initialization, when no observations were available from the target system, the driving strategies defaulted to more conservative driving behaviors. After adapting the dynamics model using observations collected through one demonstration, the driving strategies changed to match the witnessed capabilities of the system. For higher traction systems, this involved driving directly over obstacles. Again the driving strategies were non-myopic, opting for less direct paths to avoid obstacles if necessary. They also took advantage of the robot's dynamic capabilities, choosing specific directions to traverse certain obstacles depending on tire traction. Even in scenarios where no target system data was available, driving strategies avoided ascending stairs, but did not avoid descending or driving across them, matching the capabilities of the lowest traction system.

Despite dynamics models being inevitably imperfect, decision making for rough terrain

navigation was capable of making non-myopic decisions that utilized the dynamic capabilities of the robot through capturing prediction uncertainty and using the closed-loop divergence constraint.

6.1 Future Directions

Here, we note some potential future extensions to the concepts from this dissertation that could make them applicable more broadly to other planning and controls frameworks as well as to other robotic applications.

6.1.1 Divergence Constraint within Other Hierarchies

The closed-loop divergence constraint is a great metric to facilitate a better understanding of the robot's dynamics and capabilities within high-level decision making. It helps indicate the low-level controller's ability to track a reference plan in the presence of modeling uncertainty. By constraining planned trajectories to have low closed-loop divergence ensures that the low-level controller could faithfully execute the plan and mitigates the risk of the planner exploiting modeling errors.

Extending to Other High-Level Planners

Originally, this metric was used within a constrained trajectory optimization framework. However, in Chapter 5, we adapted the divergence constraint to be used within an RRT-based path planner, similar to [65]. Here, the divergence constraint served as the collision check and indicated when different candidate reference trajectories would potentially result in failure. In this adaptation, the dynamics model directly predicts robot state transitions given the reference trajectory instead of the robot's action.

In the broader context of robotics, the closed-loop divergence constraint can help any high-level decision making framework determine whether a candidate trajectory is dynam-

ically feasible under the capabilities of the low-level trajectory tracking controller and the robot's dynamics. It is particularly well suited for sampling based planners and optimizers as samples that violate the divergence constraint could simply be discarded or heavily penalized. A few decision-making frameworks that could potentially benefit from the divergence constraint include, but are not limited to Model Predictive Path Integral (MPPI) [14, 15], Cross Entropy Method (CEM) [147], Bayesian Optimization [148], and Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [149].

While we believe that many high-level decision making frameworks could potentially benefit from the divergence constraint, we also note some limitations. Most notably, calculating the divergence constraint metric requires significant computational resources. When computing the divergence constraint, multiple trajectory predictions are performed using a probabilistic dynamics model and considering the effects of the low-level tracker. The deviation of each prediction from the reference trajectory is averaged. Due to the stochastic nature of trajectory predictions, using more predictions provides a more consistent metric evaluation, but at a cost of computation. This problem is exacerbated with more complex and computationally heavy models, such as the terrain-aware model we used that requires generating a robot-centric terrain height map at each time step during prediction.

While there are numerous potential optimizations to this computation process, most notably moving from Python to C++ utilizing CUDA [150], we wonder if there is a better alternative to bypass the need for so many trajectory predictions in parallel. Potentially collecting a dataset of the robot attempting to track different reference trajectories could allow us to train a neural network model that directly predicts the divergence metric.

Another important consideration for adapting the divergence constraint to other planning frameworks is the continuity requirement for computing the divergence constraint. Due to the sequential nature of performing multiple trajectory predictions in parallel, the divergence constraint effectively has a rather large "hidden state", comprised of the current state of each individual trajectory prediction. This factor might limit it from graph search methods that

require substituting the parents of different nodes with more optimal parents, as is the case for RRT* [143] and even RRT-connect [151]. The divergence constraint could still be relevant to such planning algorithms, but require more careful consideration into how the divergence constraint propagates along edges in the graph.

Extending to Low-Level RL Policies

The original formulation of the closed-loop divergence constraint required trajectory tracker to have a simple closed-loop form since the feedback action of the trajectory tracker was calculated at each prediction time step given the current state prediction. This requirement limited the framework to relatively simple trajectory trackers such as the PD tracker originally used. However, we lifted this restriction in Chapter 5 by revising the dynamics model to directly predict the robot's state transitions given the reference trajectory of the trajectory tracker. While we still used a relatively simple pure-pursuit trajectory tracker, this revision opens up the possibility of using far more complex low-level controllers such as computationally heavy model predictive controllers or potentially model-free policies that rely on complex observations not included in the system's state, e.g. RGB-D observations.

A potentially promising avenue for future work would be to use the divergence constraint to bridge the gap between model-free and model-based reinforcement learning within a hierarchical framework. Recent advancements in reinforcement learning has led to impressive policies that achieve agile maneuvering in the precise of obstacles and rough terrain [152–154]. However, while these policies demonstrate impressive locomotion capabilities moving the robot in a direction or to a nearby waypoint, they are often poorly suited for the task of navigation that requires long-horizon decision making. Beyond lacking interpretability, these policies often struggle to strategize over extended timeframes and generalize to new environments with new obstacle layouts. Using these learned policies at the low-level with the divergence constrained planning framework could combine the agile abilities of these policies with the robust long-horizon planning abilities of our framework. This potential extension is

similar in some ways to some hierarchical reinforcement learning approaches [21, 155], where a low-level policy follows instructions from a high-level model-based planner, which chooses actions based on a learned model that predicts the outcomes of different instructions.

6.1.2 Adaptive Dynamics Models

In our framework, we use a System Identification Transformer to adapt the Adaptive Dynamics Model to the target system by extracting an understanding of the particular system's dynamics from observations collected on the target system. However, we note a few challenges of the framework and potential future directions for improvements.

Training a System Identification Transformer requires a large variety of simulation data. Not only in terms of variety in simulated system dynamics, but also in terms of variety in collected target system observations given to the System Identification Transformer. For the original simple task of driving over flat terrain, generating enough diversity in target system observations was rather simple. However, generating enough diversity was significantly more difficult when we moved to uneven terrain in Chapter 5, due to the large variety of obstacles the vehicle might encounter. The System Identification Transformer must learn to extract system dynamics regardless of what obstacles the robot interacted with in the observation dataset. In Chapter 5, we limited the robot to just a couple environments to limit the number of obstacles it may encounter.

Ideally, this restriction could be lifted and the System Identification Transformer could extract an understanding of the vehicle's dynamics from prior interactions with any type of obstacle in any type of environment. This is still likely possible through additional engineering effort, i.e. scaling up the simulation framework and fine tuning of the neural network training. However, a potential alternative to alleviate this burden would be to move from neural network inference of the context vector to an optimization based approach, where an optimization scheme is used to find the best context vector to describe the target system. This is in some aspects similar to some existing approaches [129–131, 156], but focused on

generating a probabilistic dynamics model for robust model-based planning or controls.

Another limitation to using a System Identification Transformer is its assumption of static dynamics. While this approach allows the dynamics model to quickly adapt to a new system, it assumes that dynamics for that system does not change. Future work could address this limitation by detecting changes to the vehicle's dynamics or actively predicting the effects that different terrain in the robot's horizon has on vehicle dynamics.

We believe that the ideas and concepts discussed throughout this dissertation hold vast potential, not only in the context of UGV navigation in rough terrain environments, but potentially also in other robotic systems, such as UAV, AUVs, legged systems, or manipulation. The approaches developed are especially useful for any task that involves complex robot dynamics and requires non-myopic decision making under careful consideration of the robot's dynamic capabilities.

Bibliography

[1] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI magazine*, vol. 29, no. 1, pp. 9–9, 2008.

[2] J. Qian, B. Zi, D. Wang, Y. Ma, and D. Zhang, "The design and development of an omni-directional mobile robot oriented to an intelligent manufacturing system," *Sensors*, vol. 17, no. 9, p. 2073, 2017.

[3] C. Chen, E. Demir, Y. Huang, and R. Qiu, "The adoption of self-driving delivery robots in last mile logistics," *Transportation research part E: logistics and transportation review*, vol. 146, p. 102214, 2021.

[4] D. Jennings and M. Figliozzi, "Study of sidewalk autonomous delivery robots and their potential impacts on freight efficiency and travel," *Transportation Research Record*, vol. 2673, no. 6, pp. 317–326, 2019.

[5] "Mars exploration rovers overview," https://mars.nasa.gov/mer/mission/overview/, 2023, accessed: 2023-12-31.

[6] M. Dunbabin and L. Marques, "Robots for environmental monitoring: Significant advancements and applications," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 24–39, 2012.

[7] M. Trincavelli, M. Reggente, S. Coradeschi, A. Loutfi, H. Ishida, and A. J. Lilienthal, "Towards environmental monitoring with mobile robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 2210–2215.

[8] U. Ozguner, C. Stiller, and K. Redmill, "Systems for safety and autonomous behavior in cars: The darpa grand challenge experience," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 397–412, 2007.

[9] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[10] A. Kelly, A. Stentz, O. Amidi, M. Bode, D. Bradley, A. Diaz-Calderon, M. Happold, H. Herman, R. Mandelbaum, T. Pilarski *et al.*, "Toward reliable off road autonomous vehicles operating in challenging environments," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 449–483, 2006.

[11] L. D. Jackel, E. Krotkov, M. Perschbacher, J. Pippine, and C. Sullivan, "The DARPA LAGR program: Goals, challenges, methodology, and phase I results," *Journal of Field robotics*, vol. 23, no. 11–12, pp. 945–973, 2006.

[12] F. Islam, M. Nabi, and J. E. Ball, "Off-road detection analysis for autonomous ground vehicles: a review," *Sensors*, vol. 22, no. 21, p. 8463, 2022.

[13] P. Papadakis, "Terrain traversability analysis methods for unmanned ground vehicles: A survey," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 4, pp. 1373–1385, 2013.

[14] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1433–1440.

[15] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1714–1721.

[16] J. Li, M. Ran, H. Wang, and L. Xie, "Mpc-based unified trajectory planning and tracking control approach for automated guided vehicles," in *2019 IEEE 15th International Conference on Control and Automation (ICCA)*. IEEE, 2019, pp. 374–380.

[17] N. D. Van, M. Sualeh, D. Kim, and G.-W. Kim, "A hierarchical control system for autonomous driving towards urban challenges," *Applied Sciences*, vol. 10, no. 10, p. 3543, 2020.

[18] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[19] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[20] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[21] T. Li, N. Lambert, R. Calandra, F. Meier, and A. Rai, "Learning generalizable locomotion skills with hierarchical reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 413–419.

[22] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[23] R. Behringer, S. Sundareswaran, B. Gregory, R. Elsley, B. Addison, W. Guthmiller, R. Daily, and D. Bevly, "The darpa grand challenge - development of an autonomous vehicle," in *IEEE Intelligent Vehicles Symposium, 2004*, 2004, pp. 226–231.

[24] C. Urmson, J. A. Bagnell, C. Baker, M. Hebert, A. Kelly, R. Rajkumar, P. E. Rybski, S. Scherer, R. Simmons, S. Singh *et al.*, "Tartan racing: A multi-modal approach to the darpa urban challenge," Carnegie Mellon University, Pittsburgh, PA, Tech. Rep., April 2007.

[25] F. Fahimi, *Autonomous Robots: Modeling, path planning, and Control*. Springer, 2010.

[26] P. Polack, F. Altché, B. d'Andréa Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" in *2017 IEEE intelligent vehicles symposium (IV)*. IEEE, 2017, pp. 812–818.

[27] P. Fankhauser and M. Hutter, *A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation*. Cham: Springer International Publishing, 2016, pp. 99–120. [Online]. Available: https://doi.org/10.1007/978-3-319-26054-9_5

[28] T. Shan, J. Wang, B. Englot, and K. Doherty, "Bayesian generalized kernel inference for terrain traversability mapping," in *In Proceedings of the 2nd Annual Conference on Robot Learning*, 2018.

[29] K. Viswanath, K. Singh, P. Jiang, P. Sujit, and S. Saripalli, "Offseg: A semantic segmentation framework for off-road driving," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2021, pp. 354–359.

[30] T. Guan, D. Kothandaraman, R. Chandra, A. J. Sathyamoorthy, K. Weerakoon, and D. Manocha, "Ga-nav: Efficient terrain segmentation for robot navigation in unstructured outdoor environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 8138–8145, 2022.

[31] T. M. Howard, C. J. Green, and A. Kelly, "Receding horizon model-predictive control for mobile robot navigation of intricate paths," in *Field and Service Robotics: Results of the 7th International Conference*. Springer, 2010, pp. 69–78.

[32] J. Shin, D. Kwak, and K. Kwak, "Model predictive path planning for an autonomous ground vehicle in rough terrain," *International Journal of Control, Automation and Systems*, vol. 19, pp. 2224–2237, 2021.

[33] A. Tahirovic and G. Magnani, "Passivity-based model predictive control for mobile robot navigation planning in rough terrains," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 307–312.

[34] J. V. Frasch, A. Gray, M. Zanon, H. J. Ferreau, S. Sager, F. Borrelli, and M. Diehl, "An auto-generated nonlinear mpc algorithm for real-time obstacle avoidance of ground vehicles," in *2013 European Control Conference (ECC)*. IEEE, 2013, pp. 4136–4141.

[35] D. Q. Mayne, "Differential dynamic programming–a unified approach to the optimization of dynamic systems," in *Control and dynamic systems*. Elsevier, 1973, vol. 10, pp. 179–254.

[36] L. T. Biegler and V. M. Zavala, "Large-scale nonlinear programming using ipopt: An integrating framework for enterprise-wide dynamic optimization," *Computers & Chemical Engineering*, vol. 33, no. 3, pp. 575–582, 2009.

[37] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta numerica*, vol. 4, pp. 1–51, 1995.

[38] T. M. Moerland, J. Broekens, A. Plaat, C. M. Jonker *et al.*, "Model-based reinforcement learning: A survey," *Foundations and Trends® in Machine Learning*, vol. 16, no. 1, pp. 1–118, 2023.

[39] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, 2017.

[40] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[41] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2019.

[42] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine *et al.*, "Model-based reinforcement learning for atari," *arXiv preprint arXiv:1903.00374*, 2019.

[43] X. Lei, Z. Zhang, and P. Dong, "Dynamic path planning of unknown environment based on deep reinforcement learning," *Journal of Robotics*, vol. 2018, 2018.

[44] G. Ishigami, A. Miwa, K. Nagatani, and K. Yoshida, "Terramechanics-based model for steering maneuver of planetary exploration rovers on loose soil," *Journal of Field robotics*, vol. 24, no. 3, pp. 233–250, 2007.

[45] R. He, C. Sandu, A. K. Khan, A. G. Guthrie, P. S. Els, and H. A. Hamersma, "Review of terramechanics models and their applicability to real-time applications," *Journal of Terramechanics*, vol. 81, pp. 3–22, 2019.

[46] S. Taheri, C. Sandu, S. Taheri, E. Pinto, and D. Gorsich, "A technical survey on terramechanics models for tire–terrain interaction used in modeling and simulation of wheeled vehicles," *Journal of Terramechanics*, vol. 57, pp. 1–22, 2015.

[47] A. Pazooki, S. Rakheja, and D. Cao, "Modeling and validation of off-road vehicle ride dynamics," *Mechanical systems and signal processing*, vol. 28, pp. 679–695, 2012.

[48] S. Triest, M. Sivaprakasam, S. J. Wang, W. Wang, A. M. Johnson, and S. Scherer, "Tartandrive: A large-scale dataset for learning off-road dynamics models," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 2546–2552.

[49] J. Gibson, B. Vlahov, D. Fan, P. Spieler, D. Pastor, A. akbar Agha-mohammadi, and E. A. Theodorou, "A multi-step dynamics modeling framework for autonomous driving in multiple environments," 2023.

[50] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International journal of robotics research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[51] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 2020, pp. 737–744.

[52] S. J. Wang, S. Triest, W. Wang, S. Scherer, and A. Johnson, "Rough terrain navigation using divergence constrained model-based reinforcement learning," in *Conference on Robot Learning*, 2021.

[53] S. J. Wang and A. M. Johnson, "Domain adaptation using system invariant dynamics models," in *Learning for Dynamics and Control*. PMLR, 2021, pp. 1130–1141.

[54] S. J. Wang, H. Zhu, and A. M. Johnson, "Pay attention to how you drive: Safe and adaptive model-based reinforcement learning for off-road driving," in *IEEE Intl. Conference on Robotics and Automation*, 2024, to appear. Also available as arXiv:2310.08674 [cs.RO].

[55] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[56] S. Wang, A. Bhatia, M. T. Mason, and A. M. Johnson, "Contact localization using velocity constraints," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 7351–7358.

[57] S. Wang, V. Nava, N. Jones, G. Lowry, and A. M. Johnson, "Ground-based robots for soil collection and analysis," in *American Geophysical Union (AGU) Fall Meeting*, December 2020, workshop poster.

[58] K. Nagatani, G. Ishigami, and Y. Okada, "Modeling and control of robots on rough terrain," in *Springer handbook of robotics*. Springer, 2016, pp. 1267–1284.

[59] A. Kelly and A. Stentz, "Rough terrain autonomous mobility–part 2: An active vision, predictive control approach," *Autonomous Robots*, vol. 5, no. 2, pp. 163–198, 1998.

[60] T. M. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *The International Journal of Robotics Research*, vol. 26, no. 2, pp. 141–166, 2007.

[61] 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*, vol. 21, no. 1, pp. 64–73, 2014.

[62] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1–2, pp. 99–134, 1998.

[63] A. T. Schwarm and M. Nikolaou, "Chance-constrained model predictive control," *AIChE Journal*, vol. 45, no. 8, pp. 1743–1752, 1999.

[64] 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*, 2009, pp. 1183–1189.

[65] A. M. Johnson, J. E. King, and S. Srinivasa, "Convergent planning," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 1044–1051, 2016.

[66] 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*, 2007, pp. 7–7.

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

[68] 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*, 2018, pp. 3044–3051.

[69] R. O. Chavez-Garcia, J. Guzzi, L. M. Gambardella, and A. Giusti, "Learning ground traversability from simulations," *IEEE Robotics and Automation letters*, vol. 3, no. 3, pp. 1695–1702, 2018.

[70] 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*, 2006, pp. 518–525.

[71] K. Zhang, F. Niroui, M. Ficocelli, and G. Nejat, "Robot navigation of environments with unknown rough terrain using deep reinforcement learning," in *IEEE International Symposium on Safety, Security, and Rescue Robotics*, 2018, pp. 1–7.

[72] S. Josef and A. Degani, "Deep reinforcement learning for safe local planning of a ground vehicle in unknown rough terrain," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6748–6755, 2020.

[73] T. Blum and K. Yoshida, "PPMC RL training algorithm: Rough terrain intelligent robots through reinforcement learning," *arXiv preprint arXiv:2003.02655*, 2020.

[74] G. Kahn, P. Abbeel, and S. Levine, "BADGR: An autonomous self-supervised learning-based navigation system," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1312–1319, 2021.

[75] 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.

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

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

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

[79] 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.

[80] D. F. Specht, "Probabilistic neural networks," *Neural networks*, vol. 3, no. 1, pp. 109–118, 1990.

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

[82] ——, "Model regularization for stable sample rollouts." in *UAI*, 2014, pp. 780–789.

[83] S. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, no. 4–5, pp. 185–196, 1993.

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

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

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

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

[88] P. Fankhauser, M. Bloesch, and M. Hutter, "Probabilistic terrain mapping for mobile robots with uncertain localization," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3019–3026, 2018.

[89] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[90] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[91] M. Müller, "Dynamic time warping," *Information retrieval for music and motion*, pp. 69–84, 2007.

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

[93] 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.

[94] E. V. Kumar and J. Jerome, "Robust lqr controller design for stabilizing and trajectory tracking of inverted pendulum," *Procedia Engineering*, vol. 64, pp. 169–178, 2013.

[95] C. Liu, J. Pan, and Y. Chang, "Pid and lqr trajectory tracking control for an unmanned quadrotor helicopter: Experimental studies," in *2016 35th Chinese Control Conference (CCC)*. IEEE, 2016, pp. 10 845–10 850.

[96] A. Nagariya and S. Saripalli, "An iterative lqr controller for off-road and on-road vehicles using a neural network dynamics model," in *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2020, pp. 1740–1745.

[97] 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*, 2018, pp. 5761–5768.

[98] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2555–2565.

[99] D. Ha and J. Schmidhuber, "World models," *arXiv preprint arXiv:1803.10122*, 2018.

[100] C. G. Atkeson and J. C. Santamaria, "A comparison of direct and model-based reinforcement learning," in *IEEE International Conference on Robotics and Automation*, vol. 4, 1997, pp. 3557–3564.

[101] J. Kocijan, R. Murray-Smith, C. E. Rasmussen, and A. Girard, "Gaussian process model based predictive control," in *Proceedings of the 2004 American control conference*, vol. 3. IEEE, 2004, pp. 2214–2219.

[102] Z. I. Botev, D. P. Kroese, R. Y. Rubinstein, and P. L'Ecuyer, "The cross-entropy method for optimization," in *Handbook of statistics*. Elsevier, 2013, vol. 31, pp. 35–59.

[103] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017, pp. 23–30.

[104] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," in *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.

[105] K. Bousmalis, A. Irpan, P. Wohlhart *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *IEEE International Conference on Robotics and Automation*, 2018, pp. 4243–4250.

[106] F. Golemo, A. A. Taiga, A. Courville, and P.-Y. Oudeyer, "Sim-to-real transfer with neural-augmented robot simulation," in *Conference on Robot Learning*, 2018, pp. 817–828.

[107] A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine, "Learning invariant feature spaces to transfer skills with reinforcement learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[108] N. H. Kim, Z. Xie, and M. Panne, "Learning to correspond dynamical systems," in *Learning for Dynamics and Control*. PMLR, 2020, pp. 105–117.

[109] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor fusion IV: control paradigms and data structures*, vol. 1611. International Society for Optics and Photonics, 1992, pp. 586–606.

[110] 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, 1999.

[111] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.

[112] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[113] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[114] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014.

[115] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart, "Robot-centric elevation mapping with uncertainty estimates," in *International Conference on Climbing and Walking Robots (CLAWAR)*, 2014.

[116] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, 2019, pp. 8026–8037.

[117] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[118] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2018.

[119] B. Zhou, J. Yi, and X. Zhang, "Learning to navigate on the rough terrain: A multi-modal deep reinforcement learning approach," in *IEEE International Conference on Power, Intelligent Computing and Systems*, 2022, pp. 189–194.

[120] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," in *Robotics: Science and Systems*, 2017.

[121] S. Zhu, A. Kimmel, K. E. Bekris, and A. Boularias, "Fast model identification via physics engines for data-efficient policy search," in *International Joint Conference on Artificial Intelligence*, 2018, pp. 3249–3256.

[122] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *Robotics: Science and Systems*, 2018.

[123] Y.-Y. Tsai, H. Xu, Z. Ding, C. Zhang, E. Johns, and B. Huang, "Droid: Minimizing the reality gap using single-shot human demonstration," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3168–3175, 2021.

[124] M. Kaspar, J. D. M. Osorio, and J. Bock, "Sim2real transfer for reinforcement learning without dynamics randomization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020, pp. 4383–4388.

[125] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "RMA: Rapid motor adaptation for legged robots," in *Robotics: Science and Systems*, 2021.

[126] J. Yin, Z. Zhang, and P. Tsiotras, "Risk-aware model predictive path integral control using conditional value-at-risk," in *IEEE International Conference on Robotics and Automation*, 2023, pp. 7937–7943.

[127] Z. Xie, X. Da, M. Van de Panne, B. Babich, and A. Garg, "Dynamics randomization revisited: A case study for quadrupedal locomotion," in *IEEE International Conference on Robotics and Automation*, 2021, pp. 4955–4961.

[128] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *IEEE International Conference on Robotics and Automation*, 2018, pp. 3803–3810.

[129] W. Yu, J. Tan, Y. Bai, E. Coumans, and S. Ha, "Learning fast adaptation with meta strategy optimization," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2950–2957, 2020.

[130] W. Yu, C. K. Liu, and G. Turk, "Policy transfer with strategy optimization," in *International Conference on Learning Representations*, 2019.

[131] W. Yu, V. C. Kumar, G. Turk, and C. K. Liu, "Sim-to-real transfer for biped locomotion," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019, pp. 3503–3510.

[132] N. Hansen, A. Ostermeier, and A. Gawelczyk, "On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation," in *International Conference on Genetic Algorithms*, 1995, pp. 57–64.

[133] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[134] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[135] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *Advances in NIPS 2016 Deep Learning Symposium, arXiv:1607.06450*, 2016.

[136] M. S. Gandhi, B. Vlahov, J. Gibson, G. Williams, and E. A. Theodorou, "Robust model predictive path integral control: Analysis and performance guarantees," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1423–1430, 2021.

[137] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Information-theoretic model predictive control: Theory and applications to autonomous driving," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1603–1622, 2018.

[138] C. Feller and C. Ebenbauer, "Relaxed logarithmic barrier function based model predictive control of linear systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 3, pp. 1223–1238, 2016.

[139] E. G. Birgin and J. M. Martínez, *Practical augmented Lagrangian methods for constrained optimization*. SIAM, 2014.

[140] L. Petit and A. L. Desbiens, "RRT-Rope: A deterministic shortening approach for fast near-optimal path planning in large-scale uncluttered 3d environments," in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2021, pp. 1111–1118.

[141] E. Heiden, L. Palmieri, S. Koenig, K. O. Arras, and G. S. Sukhatme, "Gradient-informed path smoothing for wheeled mobile robots," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 1710–1717.

[142] B. Hu, Z. Cao, and M. Zhou, "An efficient RRT-based framework for planning short and smooth wheeled robot motion under kinodynamic constraints," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 4, pp. 3292–3302, 2021.

[143] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011. [Online]. Available: https://doi.org/10.1177/0278364911406761

[144] R. C. Coulter *et al.*, *Implementation of the pure pursuit path tracking algorithm*. Carnegie Mellon University, The Robotics Institute, 1992.

[145] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[146] T. Han, A. Liu, A. Li, A. Spitzer, G. Shi, and B. Boots, "Model predictive control for aggressive driving over uneven terrain," 2023.

[147] R. Y. Rubinstein and D. P. Kroese, *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*. Springer, 2004, vol. 133.

[148] D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek, "Scalable global optimization via local bayesian optimization," *Advances in neural information processing systems*, vol. 32, 2019.

[149] N. Hansen, "The cma evolution strategy: a comparing review," *Towards a new evolutionary computation: Advances in the estimation of distribution algorithms*, pp. 75–102, 2006.

[150] J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[151] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 995–1001 vol.2.

[152] A. Datar, C. Pan, M. Nazeri, and X. Xiao, "Toward wheeled mobility on vertically challenging terrain: Platforms, datasets, and algorithms," *arXiv preprint arXiv:2303.00998*, 2023.

[153] Z. Zhuang, Z. Fu, J. Wang, C. Atkeson, S. Schwertfeger, C. Finn, and H. Zhao, "Robot parkour learning," *arXiv preprint arXiv:2309.05665*, 2023.

[154] X. Cheng, K. Shi, A. Agarwal, and D. Pathak, "Extreme parkour with legged robots," *arXiv preprint arXiv:2309.14341*, 2023.

[155] T. Li, R. Calandra, D. Pathak, Y. Tian, F. Meier, and A. Rai, "Planning in learned latent action spaces for generalizable legged locomotion," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2682–2689, 2021.

[156] W. Xiao, T. He, J. Dolan, and G. Shi, "Safe deep policy adaptation," *arXiv preprint arXiv:2310.08602*, 2023.