Breaking things so you don't have to: risk assessment and failure prediction for cyber-physical AI

by

Charles Burke Dawson

B.S. Engineering, Harvey Mudd College, 2019
M.S. Aeronautics and Astronautics, MIT, 2021

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

Authored by:	Charles Burke Dawson Department of Aeronautics and Astronautics May 16, 2024
Certified by:	Chuchu Fan Assistant Professor of Aeronautics and Astronautics, Thesis Supervisor
Certified by:	Russ Tedrake Professor of Electrical Engineering and Computer Science, Committee Member
Certified by:	Sertaç Karaman Professor of Aeronautics and Astronautics, Committee Member
Accepted by:	Jonathan P. How Professor of Aeronautics and Astronautics Chair, Graduate Program Committee, Department of Aeronautics and Astronautics

Breaking things so you don't have to: risk assessment and failure prediction for cyber-physical AI

by

Charles Burke Dawson

Submitted to the Department of Aeronautics and Astronautics
on May 16, 2024 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

Before autonomous systems can be deployed in safety-critical environments, we must be able to verify that they will perform safely, ideally without the risk and expense of real-world testing. A wide variety of formal methods and simulation-driven techniques have been developed to solve this verification problem, but they typically rely on difficult-to-construct mathematical models or else use sample-inefficient black-box optimization methods. Moreover, existing verification methods provide little guidance on how to optimize the system's design to be more robust to the failures they uncover. In this thesis, I develop a suite of methods that accelerate verification and design automation of robots and other autonomous systems by using program analysis tools such as automatic differentiation and probabilistic programming to automatically construct mathematical models of the system under test. In particular, I make the following contributions. First, I use automatic differentiation to develop a flexible, general-purpose framework for end-to-end design automation and statistical safety verification for autonomous systems. Second, I improve the sample efficiency of end-to-end optimization using adversarial optimization to falsify differentiable formal specifications of desired robot behavior. Third, I provide a novel reformulation of the design and verification problem using Bayesian inference to predict a more diverse set of challenging adversarial failure modes. Finally, I present a data-driven method for root-cause failure diagnosis, allowing system designers to infer what factors may have contributed to failure based on noisy data from real-world deployments. I apply the methods developed in this thesis to a range of challenging problems in robotics and cyberphysical systems. I demonstrate the use of this design and verification framework to optimize spacecraft trajectory and control systems, multi-agent formation and communication strategies, vision-in-the-loop controllers for autonomous vehicles, and robust generation dispatch for electrical power systems, and I apply this failure diagnosis tool on real-world data from scheduling failures in a nationwide air transportation network.

Thesis supervisor: Chuchu Fan
Title: Assistant Professor of Aeronautics and Astronautics

Acknowledgments

I am lucky to have had an array of supporters throughout my PhD, each of whom deserves thanks for their role in this adventure. First, to my advisor, Professor Chuchu Fan: thank you for bringing me into your lab nearly four years ago, coaching me through the ups and downs of academic life, and giving me space to explore new research interests. Our weekly discussions provided a great source of perspective, support, and (when needed) challenge, and I am a better researcher for having had the privilege of working with you. In addition, thank you to my undergraduate advisors, Professors Kash Gokli and Philip Cha, for starting me on my academic journey with just the right combination of support and independence.

Next, to my thesis committee and research collaborators. Professors Russ Tedrake and Sertaç Karaman, thank you for your helpful discussions and feedback as I progressed from blurry vision to completed PhD. To Professors Sicun Gao and Max Li, thank you for the opportunity to work together at the start and end of my PhD journey, respectively. To Chelsea, Yang, Austin, and Falk: I'm grateful that we had the chance to collaborate and learn from each other's perspectives. To Chafik, Rachel, Jake, Fayez, Gee, Tyler, Jordyn, Britany, and the rest of the team at Marble, thanks for having me along for the ride!

I am also grateful for the countless conversations, both personal and research-related, with my labmates over the years: Kunal, Jake, Kwesi, Yue, Songyuan, Ji, Anjali, Oswin, Mingxin, Yongchao, Ruixiao, Yilun, Cheng, Allen, Laura, Chenning, Rujul, Sydney, Zengyi, and Kathleen. In addition, I am lucky to have had the opportunity to mentor a number of fantastic undergraduate researchers during my PhD: Dylan, Bethany, Mukun, Suchitha, Jenny, and Van. Hopefully you learned as much from working with me as I did from working with you. I know that you will all accomplish great things in the coming years.

Beyond MIT, thank you to the friends who have kept me sane over the last 5 years. To my friends from Harvey Mudd, Camille, Maggie, Alex, Sara, Katie, and Brenden: thank you for countless catch-up calls and apple-picking adventures. To my friends in Boston, Riley, Matt, Fran, James, Annie, Avi, and Leah: thank you for good times and board games, climbing, and canoeing. To Evan: thank you for your level-headed advice and good-natured humor.

Most importantly, thank you to my family. To Mom and Dad: I don't think a paragraph or even a whole thesis would be enough to say thank you for your boundless love and support. You've been there for me every step of the way, and I'm so grateful for you. To Grandma and Grandpa: you two are the best role models I could have asked for, both as educators and as loving, supportive human beings. To my brother, Henry: it has been such a privilege to grow up with you — I'm a better person for having you as a brother, and I can't wait to see what you do next on your own journey.

Finally, to the love of my life, Lydia: words are not enough, but I can try. Thank you for being there with me through the ups and downs of both life and my PhD. Your compassion, intelligence, and sheer zest for life inspire me every day, and there's no place I would rather be than with you. You are the best, I love you, and I'm looking forward to discovering what the next chapter holds.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Before robots can be deployed in safety-critical environments, we must be able to verify that they will perform safely. Unfortunately, as robots and other cyberphysical systems become more complex, they become harder for human engineers to test, verify, and debug. For example, autonomous vehicle (AV) operators rely on a combination of low-fidelity but inexpensive testing on massively-parallel simulation and more expensive but higher fidelity testing on physical hardware. Despite large amounts of time, money, and engineering effort invested in these systems, AVs have struggled to reach the required high degree of reliability and still encounter unforeseen, life-threatening corner cases in the wild. The challenge of testing and debugging safety-critical cyberphysical systems is not particular to AVs; for instance, as the electric power grid is increasingly automated to deal with variable sources of renewable energy, or as airspace is increasingly occupied by unmanned aerial vehicles, it will be crucial to ensure that the associated prediction and control algorithms are thoroughly tested before deployment.

In cases where real-world testing is too risky or expensive, engineers must instead rely on mathematical modeling and simulation to verify that a robot will perform as intended. Unfortunately, there are issues with each of these approaches. Although mathematical models are amenable to formal proofs, robots are often too complex to reduce to a set of equations.

On the other hand, although simulators can handle the full complexity of a robotic system, they are often treated as black-boxes, providing an incomplete view of a robot's performance. In order to safely deploy complex robots in the real world, we require new tools that blend the rigor of mathematical modeling with the scalability and generality of simulation.

Although a number of works have attempted to close this gap by using differentiable simulators to extract additional mathematical structure from simulation, there is a gap in the literature when it comes to applying these tools to verification and design for complex, safety critical autonomous systems. This thesis aims to close this gap by developing a suite of design and verification tools that achieve improved robustness of optimized designs, better coverage of diverse failures, and generalize across problem domains.

1.1 Thesis contributions

I develop tools to support the design and analysis process for robots and other safety-critical cyberphysical systems in four ways:

1. *Design optimization:* automatically search for design parameters that achieve good performance.

2. *Safety verification:* characterize the robustness of a design and predict corner cases where it is likely to fail (by either violating a constraint or incurring a high cost).

3. *Verification-guided design:* closing the feedback loop between verification and design; e.g., by using predicted corner cases to guide future design iterations.

4. *Anomaly explanation:* explain the root causes of unexpected behavior observed in deployed systems.

The goal of this thesis is to provide tools that will support engineers in developing increasingly complex robotic systems, enabling a more efficient design process and providing the ability to verify the safety of a design *before* deployment. The following sections provide a summary of the contributions of this thesis in each of these areas before concluding with an outline of the rest of this document.

1.1.1 End-to-end design optimization and robustness certification

Practical robotic and cyberphysical systems often have diverse subsystems that do not lend themselves to easy mathematical abstraction and formal analysis. For example, it would be difficult to develop a formal mathematical model for the interactions between planning, perception, and control subsystems. Further complicating matters, these systems must operate reliably in uncertainty environments. This combination of complexity and uncertainty makes it difficult to design and verify these systems, especially when some subsystems contain machine learning models with thousands of tunable parameters and difficult-to-interpret behaviors.

To address these challenges, in Chapter 3 I develop an automated tool that enables efficient optimization and statistical robustness certification of robot designs. This framework uses differentiable programming for end-to-end optimization of robotic systems, allowing users to flexibly model interactions between subsystems and the effect of environmental uncertainty. In addition, I develop a novel statistical framework for certifying the worst-case performance and sensitivity of optimized designs, and I apply this statistical framework to motivate a new heuristic for robust end-to-end design optimization. I apply these tools to optimize the design of two robotic systems in hardware, using statistical certification to verify robustness before transferring the optimized designs to hardware without fine-tuning.

1.1.2 Improving design robustness using counterexamples for formal specifications

Although end-to-end design optimization provides a flexible means of optimizing robot behaviors, prior approaches consider either a single environment [1]–[3] or domain randomization over a large number of environments to encourage robustness [4]–[6]. Although domain randomization is a natural way of encouraging robustness, it is computationally expensive and risks missing important corner cases if the set of training examples is too small.

In Chapter 4, I take inspiration from the body of work on adversarial optimization [7] and machine learning [8] to improve both the sample efficiency and robustness of end-to-end design optimization. In addition, I incorporate differentiable temporal logic to specify high-level safety and performance requirements that the adversary then seeks to falsify. I show that feeding these adversarial counterexamples back into the design optimization process yields more robust designs while requiring far fewer samples than standard domain randomization techniques and substantially less runtime than prior formal methods for control synthesis for temporal logic. Although this chapter demonstrates how ideas from adversarial optimization can be successfully applied to end-to-end design and verification, it also introduces a key challenge that motivates my later work in Chapter 5 — the lack of diversity in adversarial examples.

1.1.3 Predicting and repairing diverse failure modes

The end-to-end design and verification methods discussed so far have the advantage of using gradients from automatic differentiation to speed the search for high-performing designs and challenging failure modes, but there are a number of drawbacks to these gradient-based methods. In particular, because gradient-based methods quickly converge to local optima, they struggle to find a set of qualitatively different failure modes. Instead, nearby failure modes collapse into a single "most severe" mode. Moreover, high variance in the gradients found via automatic differentiation can create practical challenges for gradient-based optimization on systems with complex dynamics or visual feedback. Several prior works on failure mode discovery avoid this diversity-collapse issue by relying on gradient-free methods for optimization [9], inference [10], and rare-event simulation [11]. While these gradient-free methods yield more diverse counterexamples, they sacrifice sample efficiency.

In Chapter 5, I build off of the adversarial optimization method discussed in the previous section, resolving the issue of diversity by reformulating adversarial optimization as a sequential Bayesian inference problem that naturally admits a sampling-based solution method. Rather than using gradient ascent to find high-severity failure modes, this approach uses gradient-accelerated Markov chain Monte Carlo (MCMC) algorithms to sample environmental parameters that are likely to induce failures. My approach in this chapter builds on prior work using gradient-accelerated MCMC for verification problems [12], [13], but takes the additional step of developing a sequential inference framework to consider the interaction between failure prediction and repair.

I demonstrate my approach to predict and repair a diverse set of failure modes on a wide range of autonomous and cyberphysical systems, including power networks, multi-agent systems, and robotic system with visual feedback. Across these examples, this method yields designs with a lower failure rate than those found using existing gradient-free and gradient-based optimization techniques.

1.1.4 Explaining anomalies from limited data

Despite our best efforts at predicting and repairing possible failures in simulation, autonomous systems may still fail after deployment. In these cases, engineers must be able to diagnose the failure and understand its root causes before they can take corrective action. Because failures encountered in the wild are relatively rare, they can be challenging to diagnose due to the limited amount of available data. In the case of autonomous vehicles, there may be tens of thousands of miles of normal driving data for every mile of data where a failure occurred; in the case of critical infrastructure like power and transportation networks, there might be multiple years between severe failures. Although we can frame the anomaly explanation problem as one of Bayesian inference, existing inference methods struggle to handle this imbalanced dataset and will tend to either overfit to the limited anomaly data (which may be corrupted with noise or outliers) or else underfit the anomaly in favor of the much more abundant data from normal operations.

Existing methods require careful tuning of hyperparameters to avoid overfitting or underfitting; e.g., by adjusting the amount of regularization used when training on scarce anomaly data. In Chapter 6, I develop a self-regularized method, CalNF, for robust inference in this data-constrained setting. This method uses deep normalizing flows and takes inspiration from robust regression and statistical bootstrapping methods to learn an appropriately-regularized posterior distribution for the anomaly data in a self-supervised manner. I apply CalNF to a real-world case study into the causes of the 2022 Southwest Airlines scheduling crisis, where it is able to infer hidden changes in the distribution of aircraft within the Southwest network that may have contributed to the disruption and allowed local disturbances to cascade into system-wide failures.

Chapter 2

Background and Significance

This section provides an overview of relevant literature, with an eye towards framing the significance of the contributions of this thesis. Where relevant, later chapters will also include a focused review of literature specific to the problems studied in each chapter. To begin, I introduce the design and verification problems that motivate this thesis, then discuss the features that we desire from such tools (e.g. sample efficiency, applicability across domains, etc.). I then survey existing methods to identify technical gaps that this thesis aims to fill.

2.1 Desired features of design and verification tools

As mentioned in Chapter 1, there are four main design and verification functions that I explore in this work:

1. *Design optimization:* search for design parameters that achieve good performance.
2. *Safety verification:* characterize the robustness of a design and predict cases where it might fail.
3. *Verification-guided design:* use predicted corner cases to guide future design iterations.

4. *Anomaly explanation:* explain the root causes of unexpected behavior observed in deployed systems.

Formal problem statements for each of these functions will be introduced in the relevant chapters. When considering tools for solving these problems, there are a number of features that we would like to see:

1. *Scalability:* how computationally expensive is a method?
2. *Robustness:* for design tools, how robust are the optimized designs to environmental uncertainty?
3. *Coverage:* how well does the method explore the space of possible designs and failure?
4. *Flexibility:* does a method target a particular subsystem, or can it be used for end-to-end analysis?

As we will see shortly, existing design and verification tools make various trade-offs between these desired features.

2.2 Safety verification

Safety and robustness are critical concerns for any robotic system, and there is a correspondingly large body of work studying how to verify safety properties for autonomous systems. These approaches can be broadly categorized into *model-based* and *model-free* works. Model-based approaches rely on a mathematical model of the system to be verified, and use this model to prove that the system satisfies the desired safety properties. Model-free approaches, on the other hand, do not require a mathematical model of the system, instead using a large number of samples of the system's input-output behavior to characterize its safety.

Regardless of approach, verification methods typically aim to solve one of three problems [14]: falsification, likely failure analysis, or failure probability estimation. In this thesis, we focus primarily on falsification and likely failure analysis. Denote the disturbance space $y \in \mathcal{Y} \subseteq \mathbb{R}^n$, a simulator $f\colon \mathcal{Y} \to \Xi$ that maps disturbances to system traces ξ, and a specification $\psi\colon \Xi \to \mathbb{R}$ that determines whether a trace satisfies ($\psi(\xi) \geq 0$) or does not satisfy ($\psi(\xi) < 0$) the safety constraints. In this context, the falsification problem involves solving an optimization for a counterexample

$$\text{find } y \in \mathcal{Y} \text{ s.t. } \psi(f(y)) < 0 \tag{2.1}$$

which is commonly reparameterized as a minimization problem to find the worst counterexample

$$\min_{y \in \mathcal{Y}} \psi(f(y)) \tag{2.2}$$

The likely failure analysis problem adds information about the prior distribution of disturbances $p(y)$ and aims to find the most likely counterexample

$$\max_{y \in \mathcal{Y}} p(y) \text{ s.t. } \psi(f(y)) < 0 \tag{2.3}$$

2.2.1 Model-based safety verification

Early approaches to model-based verification and fault identification use symbolic logical models of the system under test to formally reason about failures using computationally expensive tools like satisfiability (SAT) solvers or search [15], [16]. More recent approaches to model-based failure identification use mathematical models of the system dynamics to frame the problem through the lens of reachability [17], [18], optimal control [19], or optimization (e.g. sum-of-squares [20], [21]).

The primary challenge facing all of these methods is that it may be difficult or impossible to construct a symbolic model for the system under test. For example, simulating the dynamics of a power transmission system or certain contact models requires solving an optimization problem and does not have a closed form. Even when it is theoretically possible to obtain a closed-form symbolic model, in practice the need to construct and manipulate large sets of equations introduces the possibility of error and requires a large amount of human effort. This challenge limits the flexibility of these methods, both across domains (since the effort to derive a symbolic model must be replicated across domains), and across subsystems (since even if a symbolic model is available to model a controller, a similar model may not be available for the planning subsystem). Moreover, when complete search and optimization algorithms are used as the underlying solution method (e.g. SAT, tabular Hamilton-Jacobi), model-based verification methods can struggle with scalability. Historically, these difficulties have motivated the development of model-free approaches to safety verification.

2.2.2 Black-box safety verification

In practice, although we may not have access to a mathematical or symbolic model of a system, we often have access to a simulator instead, motivating a set of so-called "black-box" methods. These methods are restricted to sampling input-output pairs from the simulator without side information such as gradients [14]. Since black-box methods are usually quite easy to integrate with an existing simulator (often relying on a standardized API such as the OpenAI Gym interface [22]), they have seen widespread use, particularly in verification for autonomous vehicles [11], [23]–[30]. The three most common types of black-box verification method involve black-box optimization, reinforcement learning, or black-box inference methods [14].

Black-box optimization These methods use the optimization formulations in Eq. (2.2) or Eq. (2.3) to search for a set of inputs that violate the safety property of interest, using methods like Bayesian optimization [26], REINFORCE [9], and ant colony optimization [17]. There are a large number of subtly-different black-box optimization schemes that could be applied to the verification problem [31], all making different trade-offs between exploration and exploitation.

Reinforcement learning (RL) RL breaks the monolithic optimization problem in Eq. (2.2) into a sequential decision making problem by simulating single steps rather than entire trajectories, then optimizing a policy that maximizes a reward that encourages violation of the safety property [25]. This reward is often hand-designed using domain expertise, e.g. by rewarding an adversarial vehicle for reducing the distance between it and the vehicle under test [9].

Black-box inference These methods take inspiration from algorithms for approximate Bayesian inference and are most often applied to the failure probability estimation [11] and likely failure mode problems [10]. To estimate failure probability, [11] uses adaptive importance sampling to guide exploration of the search space towards regions that are more likely to induce a failure (while adjusting the failure probability estimate to account for this biased exploration). To generate likely failure modes, [10] uses gradient-free Markov chain Monte Carlo (MCMC) to sample failure modes that are likely to induce a failure, while [11] uses adaptive importance sampling to accomplish a similar goal.

These black-box methods offer substantially improved flexibility over traditional model-based techniques, allowing users to "plug-and-play" with any existing simulator. Many of these tools also achieve asymptotic coverage guarantees through the use of a global stochastic optimization sub-routine (although these infinite-sample guarantees are not commonly realized in practice, as empirical results in Chapter 5 demonstrate). The primary challenge facing these methods is scalability: because they do not consider additional information on the mathematical structure of the problems at hand, black-box methods tend to require a large number of samples to reach a solution. Many black-box solution methods can scale via trivially parallelizable, allowing users to reduce wall-clock time but without reducing the overall amount of computation (and associated costs).

2.2.3 Gradient-based verification

A number of methods attempt to resolve the scalability challenges of black-box methods through the use of automatic differentiation to obtain gradients through end-to-end simulations (for completeness, a review of differentiable simulation literature is provided below). Inspired by the large body of work on identifying adversarial examples in deep neural networks [8], [32], a number of works use gradient-based optimization to find counterexamples for cyberphysical systems [33]–[36]. These methods achieve improved sample efficiency over black-box methods, but because these optimization tools (by design) induce a contraction on the solution space that reduces the diversity of counterexamples discovered using these methods. Recent works have attempted to improve diversity using guided diffusion models [28] and gradient-based sampling [12], [13].

2.3 Design optimization for robotics

Optimization is a fundamental tool for robotics, with a long history of use for motion planning and trajectory optimization [37], optimal control [38], localization and mapping [39], and formal verification [40], to name just a few applications. Without much risk of overstatement: for any given roboticist and any given robotics problem, the odds are good that he or she thinks about that problem through the lens of optimization. Given the huge variety of optimization problems found in robotics and control, it is important to narrow the scope of optimization problems considered in this thesis.

In particular, this thesis will focus on *end-to-end design optimization problems* in robotics and cyberphysical systems. By design optimization, we mean the problem of finding some optimal set of parameters specifying the design of a system; e.g. control gains, physical layout of the system, neural network parameters, or roles of different agents in a multi-agent system. By taking an end-to-end approach to design optimization, we consider the joint optimization problem over all of these design parameters (e.g. simultaneously optimizing the hardware design parameters, control gains, and perception module), where we seek to simultaneously optimize the parameters of all subsystems to achieve some desired behavior.

Automatic differentiation is a key enabling technology for end-to-end robot design optimization. Recent years have seen the development of a wide range of differentiable simulation environments for rigid contact [2], [41]–[43], articulated robots [44], soft and deformable bodies [1], [43], [45], hydrodynamics [3], [46], rendering [47]–[49], and more [50]–[52]. These differentiable simulators have enabled a number of end-to-end design optimization approaches for specific domains, including simple walking robots [53], quadrotors [54], soft robots [55], [56], and swimming robots [3], [57], as well as a number of studies on the use of gradients for related policy optimization [6], [52], [58] and system identification problems [59].

The body of evidence from these works show that, in cases where gradients are well-behaved, local gradient-based optimization requires fewer samples to converge than gradient-free methods (implying improved scalability). However, as discussed in Section 2.2.3, the contracting nature of local gradient-based optimization makes it prone to getting stuck in local optima in practice. Naturally, when gradients are not well-behaved (e.g. when the optimization landscape is flat or rapidly varying, or when the estimated gradients differ from the true gradients), gradient-based optimization methods will struggle [58], [60].

Although early work on end-to-end design optimization tends to treat the environment as fixed [1]–[3], [50], subsequent works explore several different methods for improving the robustness of the optimized designs.

Domain randomization: the natural first step for incorporating robustness into end-to-end design optimization is to consider a large set of random environments rather than a fixed environment, optimizing for good average performance across environments [4], [6]. This strategy is both easy to implement, easily parallelizable, and has been shown to yield good performance on a range of problems, but the need for a large training set of environments (to avoid missing important corner cases) leads to high computational expense.

Adversarial optimization: the computational expense of domain randomization motivates the development of algorithms that solve a verification sub-problem (either falsification or likely failure prediction) to generate counterexamples for use in further optimization. Adversarial optimization has been applied in both black-box [14] and gradient-based contexts [34]. By training on a smaller number of more challenging examples, these methods can reach similar levels of robustness as domain randomization with greatly reduced computational expense (particularly when using gradient-based solvers), but there are two drawbacks to adversarial methods. The first is that gradient-based adversarial optimization inherits the lack of diversity that affects many of the gradient-based verification tools discussed in Section 2.2.3. A related second challenge is that, even if we were able to solve optimization and verification problems to global optimality, pure equilibria may not exist for many end-to-end adversarial optimization problems, requiring solution methods capable of discovering diverse mixed strategies.

2.4 Summary of technical gap

So far, this section has reviewed prior work on end-to-end design optimization and verification, with an eye towards assessing the scalability, robustness, coverage, and flexibility of these methods. A theme that emerges from this review is the trade-off between scalability and coverage (for verification methods) and between scalability and robustness (for optimization); in fact, these trade-offs represent two sides of the same problem, since improved coverage of counterexamples is linked to improved robustness of optimized designs in an adversarial setting. Given this context, the primary aim of this thesis is to develop counterexample-guided tools for end-to-end design optimization and verification that not only enable sample-efficient exploration of diverse failure modes but also translate that improved diversity into improved robustness for the optimized designs. En-route to this goal, we address several technical sub-challenges, including issues of gradient quality and local equilibria, the link between robust optimization and the statistics of extreme events, practical considerations for adversarial optimization with non-pure equilibria, and empirical comparison of various gradient-free and gradient-based verification and optimization methods.

2.5 Review of relevant program analysis tools

In practice, the term "black-box" is often used to describe the setting where we have access to a computer program implementing a simulator of the system under test. The next two sections will show how we can obtain varying degrees of transparency into the structure of these programs using program analysis methods, effectively granting the ability to look inside the black box. In particular, we will discuss two program analysis techniques that are relevant to this thesis: automatic differentiation (which treats computer programs as mathematical functions that can be differentiated) and probabilistic programming (which treats stochastic programs as graphical models that can be used for Bayesian inference).

2.5.1 Automatic differentiation

Perhaps the most well-known (and widely-used) program analysis method in the machine learning and robotics communities is automatic differentiation (autodiff, or AD). AD achieved widespread use in the form of backpropagation for training neural networks [61]. The popularity of neural networks prompted the development of differentiable tensor math libraries such as PyTorch [62], TensorFlow [63], and JAX [64]. Following these general-purpose libraries, a number of specialized AD have also emerged in the form of differentiable optimization layers [65], physical simulators [50], renderers [49], [50], and even formal task specifications [66], to name but a few.[1]

At a high level, the aim of automatic differentiation is to allow the user to implement some function $y = f(x) : \mathbb{R}^n \to \mathbb{R}^m$ and then obtain the Jacobian $Df(x) \in \mathbb{R}^{m\times n}$ without

[1]The discussion in this document is focused on the Python ecosystem, but similar tools exist in Julia, C++, and other LLVM languages [67].

writing any additional code. This is typically done in one of two ways, referred to as forward- and reverse-mode AD, respectively. In the interest of space, this document will provide only a brief overview of these two methods; [68] provides a more thorough introduction.

Forward-mode AD computes the product between a vector in the input tangent space δx and the Jacobian, "pushing forward" into the tangent space of the output $\delta x \to \delta y = Df(x)\delta x$. As a result, forward-mode AD is sometimes referred to as the Jacobian-vector product (JVP) or the pushforward map. It is typically implemented by operator overloading, in which primitive operations (e.g. $+$, $\times$, sin, etc.) are overloaded to operate on a new data type that carries both the primal x and tangent δx. As the function is computed and each primitive operation is carried out, the tangent value is updated using hand-derived derivative rules for each operation. The benefit of forward-mode AD is that its memory usage is constant with respect to the number of operations used to define f (roughly double the memory usage of evaluating the function value alone). The potential downside is that computing the Jacobian requires one primal and tangent evaluation of f for each column of the Jacobian; this is fine for functions with few inputs and many outputs, but does not scale to typical machine learning applications (where we often wish to differentiate with respect to thousands of model parameters).

Reverse-mode AD computes the product of a vector in the output tangent space δy and the transposed Jacobian, "pulling back" into the tangent space of the input $\delta y \to \delta x = Df(x)^T \delta y$, and it is referred to as the pullback map or vector-Jacobian product (VJP) accordingly. This mode is typically implemented as generalized backpropagation, constructing a computation graph containing the primitive operations applied while evaluating $y = f(x)$. To compute the derivative, we trace backwards through the computation graph, applying derivative rules for each computation. The benefit of this method is that computing the full Jacobian requires one forward and backward pass for each row of the Jacobian, which is much more efficient for systems with many input parameters and few outputs (including typical ML and optimization applications where there are many parameters or decision variables that yield a scalar objective output). The downside of this mode is that it is much more memory intensive, since the results of intermediate operations during the forward pass are typically cached and reused during the backwards pass, and the memory requirements scale linearly with the size of the computation graph.

Both of these modes require hand-derived derivative rules for the primitive operations used in computing f; however, in certain cases these rules may be overridden to provide more accurate or numerically stable gradients. An important case that commonly arises in robotics is the case where f involves solving either an optimization (e.g. $f(x) = \arg\min_y g(y, x)$) or root-finding problem (e.g. $f(x) = \text{find}_y \text{ s.t. } g(y, x) = 0$). In practice, these problems are solved iteratively, but naïvely differentiating the primitive operations applied in each iteration is both costly and inaccurate. Instead, these optimization and root-finding problems are treated as primitive operations for the purposes of AD, and the derivative rules are found automatically using the implicit function theorem [65]. This approach allows for accurate differentiation of implicit dynamics (including certain contact models [2]), rendering (since raytracing is a root-finding procedure), and optimization-based control.

It is important to note that although the gradients derived from AD are often referred to as exact or analytic, they are still only estimates of the true gradient. Most commonly used AD systems are not sound, admitting pathological inputs that can yield arbitrarily wrong gradients when differentiated (e.g. $f(x) = \{x \text{ if } x \neq 0;\ -x \text{ otherwise}\}$, which is identically equal to $y = x$ in both value and gradient but yields $df/dx(0) = -1$ when differentiated with most AD libraries). It is possible to detect some of these pathological cases, either at runtime or at compile-time, but even non-pathological functions can be stiff (with very large gradients), non-smooth, or even discontinuous. As a result, it is important to consider how these inaccuracies will affect downstream consumers of the gradient (e.g. for optimization). The downstream effects of these artifacts is an active area of research [58], [60], [69].

Significance of this thesis The last few years have seen a surge of applications of AD, including robotics problems such as control synthesis and system identification [6], [42], [57], [65], [70]. However, despite these successful applications, two questions remain to be answered in this thesis. First, since both modes of AD require more computation than a standard function evaluation, we must ask whether the derivative juice is worth the computational squeeze. That is, do AD-derived gradients provide enough of a performance increase on downstream tasks, relative to gradient-free optimizers or zero-order gradient estimates and evaluated on robotics-relevant benchmarks, to merit this additional computational expense? Second, given that AD can yield poorly-conditioned gradients on many problems of interest for robotics, can we design downstream algorithms that are robust to variance or inaccuracy in the gradients they receive? We will answer both of these questions in the affirmative in the following chapters.

2.5.2 Probabilistic programming

Probabilistic programming is a type of program analysis tool for programs that make random choices (e.g. by querying a random number generator). Probabilistic programming views the computation graph of such a program as a graphical probabilistic model (e.g. Bayesian network) that encodes a joint distribution over variables involved in the computation [71]. With this mindset, we can gain an additional level of transparency into the behavior of a program by automatically deriving this graphical model and applying approximate Bayesian inference techniques; for instance conditioning on certain variables and estimating the posterior distribution.

It is important to note that probabilistic programming is not an alternative to AD; in fact, most probabilistic programming frameworks rely on AD. AD treats programs as mathematical functions that can be differentiated, and probabilistic programming assigns a semantic meaning to those derivatives in the context of probabilistic inference problems. Moreover, while it is relatively easy to port existing code to use AD (e.g. replacing `numpy` with `jax.numpy` in Python), probabilistic programming requires additional changes the source code, often defining additional syntax to annotate random choices (so that those choices can be traced and referred to while solving inference problems).

Treating programs as probabilistic models offers a number of benefits. Both constrained and unconstrained optimization problems can be transcribed as posterior inference problems (this is the optimization-as-inference approach discussed in [72], [73] and explored in detail in Chapter 5), but inference also allows us to answer questions such as the likely failure analysis problem in (2.3).

Significance of this thesis Probabilistic programming has been applied to a large number of problems in statistical inference; in fact, many of its original applications involved automating traditional statistical inference tasks such as regression and hierarchical modeling with a convenient programming interface [74]. Although probabilistic graphical models such as factor graphs have been applied widely in robotics, with deep roots in the SLAM community [39], these methods have experienced some of the same hurdles as traditional model-based verification methods discussed above: it is difficult to derive these models by hand, particularly for complex autonomous systems. This thesis will close this gap by applying probabilistic programming to automatically generating these models for robotics safety verification and optimization problems.

Chapter 3

End-to-end Design Optimization with Statistical Robustness Certificates

Across disciplines, engineers use computer-aided design tools to boost their productivity and design increasingly complex systems. For example, mechanical engineers use a suite of 3D CAD (computer-aided design) and FEA (finite-element analysis) tools to design structures and understand their performance. Likewise, electrical engineers use electronic design automation tools, including hardware description languages like Verilog, to design and analyze large-scale, reliable, yet highly complex integrated circuits. In robotics, recent years have seen the development of highly capable modeling and simulation tools [75], [76], some of which include the ability to optimize particular subsystems, but the question of how best to leverage these modeling and simulation tools as computational design aids remains an open question (and the subject of substantial active research, as discussed in Chapter 2).

Two factors have made it difficult to develop automated design tools for robotics. The first is complexity: most robots are composed of many interacting subsystems. Although some tools may aid in designing certain subsystems (e.g. Simulink for controllers, SolidWorks or CATIA for hardware, custom software for training perception systems), these tools cover only a small part of the overall robotics design problem, which includes sensing, actuation, perception, navigation, control, and decision-making subsystems. In addition to being interconnected, these subsystems often have a large number of parameters that require tuning to achieve good performance (neural network-based perception is an extreme example); as a result, it is not enough that a tool be able to model all of these subsystems, it should help the user optimize the performance of the system end-to-end. Moreover, since few robotic systems are exactly alike, an effective design tool must allow the user to select an appropriate level of abstraction for the problem at hand. As a result, there is a need for flexible computational tools that can help designers optimize complex robotic systems.

The second difficulty is uncertainty. Robots operate in dynamic environments that cannot be fully specified *a priori*, and nonlinear interactions between the robot and its environment can make this uncertainty difficult to quantify. Nevertheless, we must account for this uncertainty during the design process and ensure that our designs perform robustly. The nature of this uncertainty can vary from problem to problem, reiterating the requirement that an automated design tool must be flexible enough to adapt to different robot design problems.

To be successful, an automated robot design tool must address these two challenges (complexity and uncertainty). In addition, just as mechanical and electrical engineers use automated tools to both *design* and *verify* their designs, a robot design tool must enable its user to both design autonomous systems and certify the robustness of those designs. The development of a tool meeting both of these requirements is the primary goal of this thesis, and this chapter presents the first component of that system: a general-purpose robot design optimization tool that is both flexible (using differentiable programming to model complex systems) and robust (avoiding "brittle" optima), along with a novel statistical approach to certifying a design's robustness to environmental uncertainty. In later chapters, we refine this approach to improve its robustness using adversarial optimization and probabilistic methods, and we improve the coverage and efficiency of the verification process using techniques from statistical inference and rare event simulation. This chapter is based on the author's published work [5].

Our goal is to develop a general-purpose robot design optimization tool that can be applied to a range of robot design problems with multiple subsystems. This goal is in contrast with other approaches that are restricted either to specific applications [3], [53]–[55], [57], [77] or subsystems [78]. To accomplish this goal, we make two novel contributions in this chapter. The first is algorithmic: our approach builds on recent developments in programming languages (i.e. automatic differentiation) to provide the flexibility to model complex systems while still allowing fast gradient-based optimization. The second concerns certification: to ensure that our optimized designs are robust in the face of uncertainty, we pair design optimization with a novel statistical approach to robustness analysis.

Our experiments show that our methods can (in our first case study) optimize a robotic system with five subsystems and six design variables in under five minutes, achieving an 8.4x performance improvement over the initial design. In our second case study, we optimize a system with three subsystems and 454 design variables in under an hour, achieving a 44% performance improvement over the initial design. Our use of differentiable programming allows us to complete this optimization 32% and 20x faster, respectively in each example, compared to approximate gradient methods. Both of these designs are certified using a statistical robustness analysis and successfully deployed in hardware. An open-source implementation of our framework, including repeatable code examples, is available at https://github.com/MIT-REALM/architect.

3.1 Preliminaries and assumptions

Key to the design of robotic systems is the tension between the factors a designer can control and those she cannot. For instance, a designer might be able to choose the locations of sensors and tune controller gains, but she cannot choose the sensor noise or disturbances (e.g. wind) encountered during operation. Robot design is therefore the process of choosing feasible values for the controllable factors (here referred to as *design parameters*) that achieve good performance despite the influence of uncontrollable factors (*exogenous parameters*).

Of course, this is a deliberately narrow view of engineering design, since it focuses on parameter optimization and ignores important steps like problem formulation and system architecture selection. Our focus on parameter optimization is intentional, as it allows the designer to focus her creative abilities and engineering judgment on the architecture problem, using computational aids as interactive tools in a larger design process [79], [80]. This focus is common in design optimization (e.g. aircraft design in [79] and 3D CAD optimization in [80]). To formalize the design optimization problem, we take a high-level view of the robot design problems (shown in Fig. 3.1), where a design problem has the following components:

Design parameters The system designer has the ability to tune certain continuous parameters $\theta \in \Theta \subseteq \mathbb{R}^n$; e.g., control gains or the positions of nodes in a sensor network.

Exogenous parameters Some factors are beyond the designer's control, such as wind speeds or sensor noise. We model these effects as random variables with some distribution $\phi \sim \Phi$ supported on a subset of $\mathbb{R}^m$. In this chapter, we assume no knowledge of Φ other than the ability to draw samples i.i.d..

Simulator Given particular choices for θ and ϕ, the system's state $s \in \mathcal{S}$ evolves in discrete time according to a known simulator $S : \Theta \times \Phi \to \mathcal{S}^T$. This simulator describes the system's behavior over a finite horizon T as a trace of states $s_1, \ldots, s_T$. S should be deterministic; randomness must be "imported" via the exogenous parameters.

Cost We assume access to a function $J : \mathcal{S}^T \to \mathbb{R}$ mapping system behaviors (i.e. a trace of states) to a scalar metric that we seek to minimize.

Constraints The choice of design parameters is governed by a set of constraints $c_i : \Theta \to \mathbb{R}$ with index set $i \in \mathcal{I}_c$. Design parameters θ are feasible if $c_i(\theta) \geq 0 \; \forall i \in \mathcal{I}_c$. Here, we consider

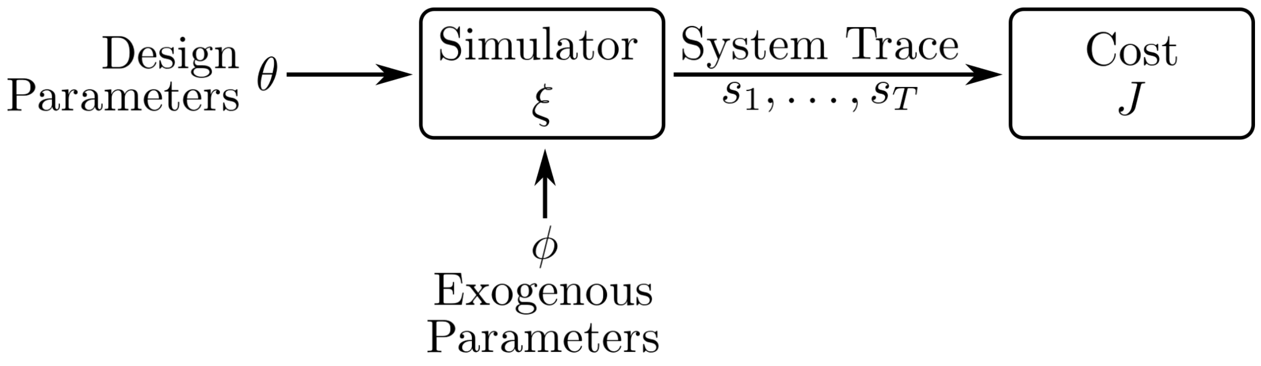


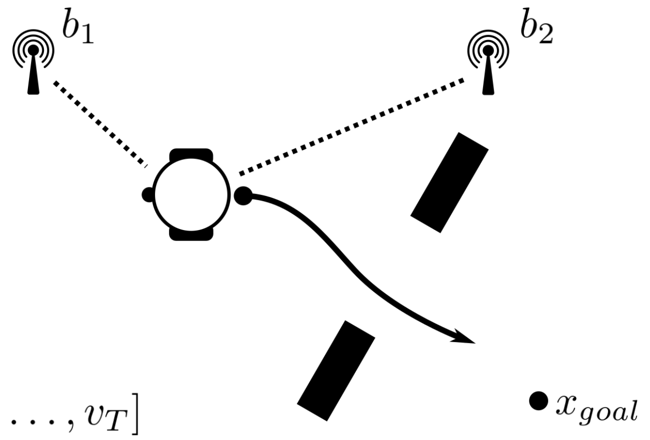
Figure 3.1: A glass-box model of a generic robotic system. Design optimization involves finding a set of design parameters so that the simulated cost is minimized, while robustness analysis involves quantifying how changes in the exogenous parameters affect the simulated cost.

constraints as functions of θ only; we leave the extension to robust constraints involving ϕ to future work.

We assume that the simulator S, cost J, and constraints c_i are automatically differentiable almost everywhere with respect to θ and ϕ. We will re-use this notation throughout this thesis, but various assumptions will be relaxed in later chapters (e.g. to consider non-deterministic simulators), and additional assumptions (e.g. smoothness) will be introduced as needed to provide theoretical guarantees.

We can make this discussion concrete with an example: consider the autonomous ground vehicle (AGV) design problem illustrated in Fig. 3.2. In this problem, our goal is to design a localization and navigation system that will allow the AGV to safely navigate between two obstacles. The AGV can estimate its position using an extended Kalman filter (EKF) with noisy measurements of its range from two nearby beacons and its heading from an IMU. The robot uses this estimate with a navigation function [81] and feedback controller to track a collision-free path between the obstacles.

In this problem, the design parameters θ include the (x, y) locations of the two range beacons $b_1, b_2 \in \mathbb{R}^2$ and the feedback controller gains $k \in \mathbb{R}^2$. The exogenous parameters ϕ are the actuation and sensor noises at each timestep $w_t \in \mathbb{R}^3$ and $v_t \in \mathbb{R}^3$, drawn i.i.d. from Gaussian distributions $\mathcal{N}(0, Q)$ and $\mathcal{N}(0, R)$, respectively, as well as the initial state (also



$$\begin{aligned}\theta &= [b_1, b_2, k] \in \mathbb{R}^6 \\ \phi &= [w_1, \ldots, w_T, v_1, \ldots, v_T] \\ &\quad \sim \mathcal{N}(0, Q)^T \times \mathcal{N}(0, R)^T\end{aligned}$$

Figure 3.2: A design optimization problem for an AGV localization and navigation system. The goal is to find placements for two range sensors along with parameters for the navigation system that allow the robot to safely pass through the narrow doorway.

Gaussian). The simulator ξ integrates the AGV's dynamics using a fixed timestep, updating the EKF and evaluating the navigation controller at each step. The cost function J assigns a penalty to collisions with the environment, estimation errors, and deviations from the goal location. We will return to this example in more detail in Section 3.4.1; first, we discuss our approach to design optimization and robustness analysis in Sections 3.2 and 3.3, respectively.

3.2 Design optimization using differentiable programming

Given the notation from Section 3.1, we can formally pose the robot design optimization problem. In formulating the optimization objective, it is important to consider the variance introduced by the exogenous parameters ϕ. Simply minimizing the expected value of the cost $\mathbb{E}_{\phi\sim\Phi}\left[J \circ S\left(\theta, \phi\right)\right]$ (where $\circ$ denotes composition) can lead to myopic behavior where exceptional performance for some values of ϕ compensates for poor performance on other values; this is related to the phenomenon of "reward hacking" in reinforcement learning [82].

Ideally, we would like our designs to be robust to variations in exogenous parameters: changing ϕ should not cause the performance to change much. We can include this requirement as a heuristic by penalizing the variance of J. Intuitively, this heuristic "smooths" the cost function with respect to the exogenous parameters: regions of high variance (containing sharp local minima) are penalized, while regions of low variance are rewarded. We return to justify this connection to robustness in Section 3.3.3. This heuristic leads us to the *variance-regularized robust design optimization problem*:

$$\min_{\theta \in \Theta} \quad \mathop{\mathbb{E}}_{\phi \sim \Phi} \Big[J \circ S\left(\theta, \phi\right) \Big] + \lambda \mathrm{Var}_{\phi \sim \Phi} \Big[\mathrm{J} \circ \mathrm{S}\left(\theta, \phi\right) \Big] \tag{3.1a}$$

$$\text{s.t.} \quad c_i(\theta) \geq 0 \quad \forall i \in \mathcal{I}_c \tag{3.1b}$$

Practically, we replace the expectation and variance with unbiased estimates over N samples $\phi_i \sim \Phi, i = 1, \dots, N$.

$$\min_{\theta \in \Theta} \quad \frac{1}{N} \sum_{i=1}^{N} \Big[J \circ S\left(\theta, \phi_i\right) \Big] \tag{3.2a}$$

$$+ \lambda \left[\frac{\sum_{i=1}^{N} \left(J \circ S\left(\theta, \phi_i\right)\right)^2}{N-1} - \frac{\left(\sum_{i=1}^{N} J \circ S\left(\theta, \phi_i\right)\right)^2}{(N-1)N} \right]$$

$$\text{s.t.} \quad c_i(\theta) \geq 0 \quad \forall i \in \mathcal{I}_c \tag{3.2b}$$

Of course, these Monte-Carlo estimates will require multiple evaluations of $J \circ S$ to evaluate (3.2a). Since S might itself be expensive to evaluate, approximating the gradients of (3.2a) and (3.2b) using finite differences will impose a large computational cost ($2nN$ additional evaluations of $J \circ S$ and c_i at each step). Instead, we can turn to automatic differentiation (AD) to directly compute these gradients with respect to θ, which we can use with any off-the-shelf gradient-based optimization algorithm. The precise choice of optimization algorithm is driven by the constraints and is not central to our framework. If the constraints are hyper-rectangle bounds on θ, then algorithms like L-BFGS-B may be used, but if the constraints are more complex then sequential quadratic programming or interior-point methods may be used. Our implementation provides an interface to a range of optimization back-ends through SciPy [83].

In this framework, the user need only implement the simulator and cost function for their specific problem using a differentiable programming framework like the JAX library for Python [64], and this implementation can be used automatically for efficient gradient-based optimization. By implementing a library of additional building blocks in this AD paradigm (e.g. estimation algorithms like the EKF), we provide an AD-based design optimization tool that strikes a productive balance between flexibility and ease of use. At the time of this writing, this library of building blocks includes an EKF for state estimation, the dynamics of various common robot platforms, formal specification languages like signal temporal logic, and differentiable RGB and depth image rendering.

3.3 Design certification via robustness analysis

Once we have found an optimal choice of design parameters, we need to verify that the design will be robust to uncertainty in the exogenous parameters. Similarly to 3D CAD and FEA packages for mechanical engineers, a successful design tool not only helps an engineer refine her design (i.e. using the design optimization framework in Section 3.2) but also helps her analyze and predict its performance. To certify the performance of an optimized design, we are interested in two distinct questions. First, what is the maximum cost we can expect given variation in the exogenous parameters? Second, how sensitive is the cost to external disturbances: by how much can a change in the exogenous parameters increase the cost?

Answering these questions is difficult because we must extrapolate worst-case performance from a finite number of simulations. To address this difficulty, we develop a probabilistic approach based on extreme value theory in statistics [84]–[86]. We begin by stating a relevant result:

Theorem 3.3.1 (Extremal Types Theorem; 3.1.1 in [86])**.** *Let $X_1, \dots, X_N$ be random variables drawn i.i.d. from an unknown distribution and $M_N = \max_i\{X_i\}$ be the sample maximum. If there exist sequences of normalizing constants $\{a_N > 0\}$ and b_N such that the* *limiting distribution of* $(M_N - b_N)/a_N$ *as* $N \to \infty$ *is non-degenerate, then*

$$\lim_{N\to\infty} \Pr\left[(M_N - b_N)/a_N \leq z\right] = G(z) \tag{3.3}$$

where $G(z)$ *is a Generalized Extreme Value Distribution (GEVD) with location* μ*, scale* σ*, and shape* ξ*,*

$$G(z) = \exp\left\{ -\left[1 + \xi\left(\frac{z-\mu}{\sigma}\right)\right]^{-1/\xi} \right\}, \tag{3.4}$$

supported on $\{z : 1 + \xi(z-\mu)/\sigma > 0\}$*.*

In the special case $\xi = 0$, this distribution has a slightly different form (known as a Gumbel distribution), but the result holds. In practice, a_n and b_n are not estimated directly (this merely changes the fit values of μ and σ) and the GEVD is fit directly to M_N by either maximizing the log likelihood [86] or estimating the posterior distribution of (μ, σ, ξ) using Markov Chain Monte Carlo methods [87]. A useful feature of the GEVD is that if our data suggest that $\xi < 0$, then the support of $G(z)$ is bounded above and we can estimate an upper bound on the maximum M_∞. If $\xi \geq 0$, then we cannot estimate a strict upper bound, but we can provide for a confidence interval for M_∞ instead. In the following sections, we apply this theorem to analyze the robustness of an optimized design.

3.3.1 Estimating the worst-case performance

Our first robustness question concerns the worst-case performance of our design: given variation in ϕ, what is the maximum cost we can expect for our choice of design parameters θ? Our insight is that the variation $\phi \sim \Phi$ induces an (unknown) distribution in $J \circ S(\theta, \phi)$, so $J \circ S(\theta, \phi)$ a random variable to which the extremal types theorem applies. Algorithm 3.3.1 provides a means for estimating the maximum of $J \circ S(\theta, \phi)$ by fitting a GEVD to observed maximums M_N. Generally speaking, the block size N and sample size M should be chosen to be as large as computationally feasible to reduce the variance of the GEVD estimate [86].

In practice, we use the automatic parallelization features of JAX to efficiently compute

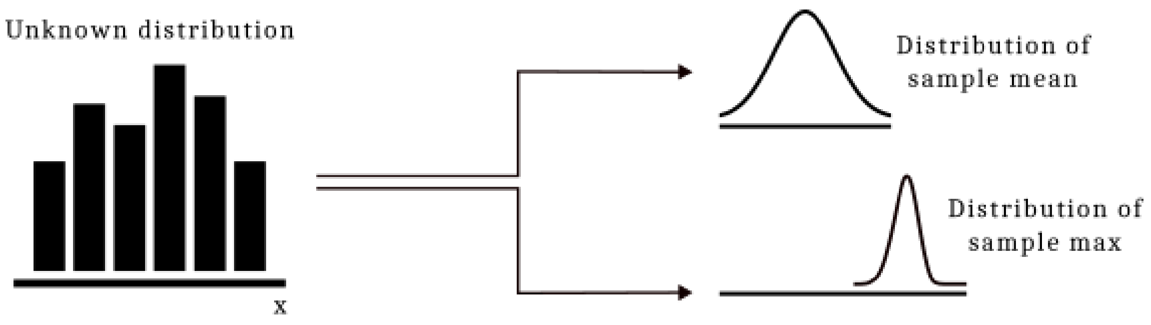


Figure 3.3: A visualization of Theorem 3.3.1, which provides conditions under which the distribution of sample maxima of samples drawn from an unknown distribution will be follow a GEVD. There is an intuitive connection with the central limit theorem, whereby the distribution of sample means will be Gaussian.

Algorithm 3.3.1: An algorithm for estimating the parameters of a GEVD governing the expected maximum cost $J \circ S$

```
Require: Block size $N > 0$ and sample size $M > 0$
  $X_j^i \leftarrow J \circ S(\theta, \phi_{ij})$; with $\phi_{ij} \sim \Phi$, $1 \leq j \leq N$, $1 \leq i \leq M$
  $M_N^i \leftarrow \max\{X_1^i, \dots, X_N^i\}$ for $i = 1, \dots, M$
  $(\mu, \sigma, \xi) \leftarrow$ posterior GEVD estimate given $\{M_N^i\}$
```

X_j^i and obtain the posterior distribution of μ, σ, and ξ using Markov Chain Monte Carlo sampling with the PyMC3 library [87]. From this posterior distribution, we take the 97% confidence level for each parameter (μ^*, σ^*, ξ^*). If $\xi^* < 0$, we have confidence that the corresponding GEVD has bounded support on the right and estimate the maximum cost $J_{max} \leq \mu - \sigma/\xi$. Otherwise, we can estimate the 97% confidence level for J_{max} using the GEVD described by (μ^*, σ^*, ξ^*).

3.3.2 Estimating sensitivity

In addition to the expected worst-case performance, it is also useful to know the sensitivity of that performance. That is, if the design performs well in one situation (i.e. for some value of ϕ), then how much can we expect its performance to degrade if ϕ changes? Formally, we define the sensitivity L as the least constant such that for any two $\phi_1, \phi_2 \sim \Phi$,

$$|J \circ S(\theta, \phi_1) - J \circ S(\theta, \phi_2)| \leq L||\phi_1 - \phi_2||$$

If $J \circ S$ is Lipschitz then L will be finite and equal the Lipschitz constant of $J \circ S$, but we do not require this assumption; if $J \circ S$ is not Lipschitz, then we can estimate a high-confidence upper bound on L.

In both cases, we can exploit the fact that L is an extreme value of the slope $|J \circ S(\theta, \phi_1) - J \circ S(\theta, \phi_2)|/||\phi_1 - \phi_2||$ and apply the extremal types theorem. Let $X = ||J \circ S(\theta, \phi_1) - J \circ S(\theta, \phi_2)||/||\phi_1 - \phi_2||$ be a random variable with $\phi_1, \phi_2 \sim \Phi$. The distribution of X is unknown, but the extremal types theorem lets us characterize the sample maximum $L_N = \max\{X_1, \dots, X_N\}$ using a GEVD. Algorithm 3.3.2 provides our method for fitting this distribution. This approach is similar to that in [84], [88] but removes the assumption that L is bounded by fitting a GEVD instead of a reverse Weibull distribution, allowing our approach to apply even when $J \circ S$ is not Lipschitz.

Algorithm 3.3.2: An algorithm for estimating the parameters of a GEVD governing the sensitivity of $J \circ S$

```
Require: Block size $N > 0$ and sample size $M > 0$
  $X_j^i \leftarrow |J \circ S(\theta, \phi_{ij,1}) - J \circ S(\theta, \phi_{ij,2})|/||\phi_{ij,1} - \phi_{ij,2}||$, with $\phi_{ij,1}, \phi_{ij,2} \sim \Phi$, $j = 1, \dots, N$, $i = 1, \dots, M$
  $L_N^i \leftarrow \max\{X_1^i, \dots, X_N^i\}$ for $i = 1, \dots, M$
  $(\mu, \sigma, \xi) \leftarrow$ posterior GEVD estimate given $\{L_N^i\}$
```

Algorithm 3.3.2 is similar to Algorithm 3.3.1, but the interpretation of the results differs in that Algorithm 3.3.2 allow us to understand the sensitivity of a design rather than its worst-case performance. In particular, if the 97% confidence level for the shape parameter ξ^* is negative, then $J \circ S$ is likely Lipschitz continuous with Lipschitz constant $L \leq \mu - \frac{\sigma}{\xi}$. If $\xi > 0$, then $J \circ S$ is likely not Lipschitz but we can estimate the 97% confidence level for L. As a result, this statistical approach allows us to avoid making prior assumptions about the continuity of our system.

3.3.3 Theoretical connections between verification by extremal types and variance-regularized optimization

Here, we will attempt to justify the variance regularization heuristic introduced in Section 3.2 with reference to the worst-case performance J_{max} and sensitivity L computed by Algorithms 3.3.1 and 3.3.2. First, let us examine the connection with expected worst-case performance J_{max}. If we take the probability of observing a cost $J = J \circ S(\theta, \phi)$ within α of J_{max}, with $0 < \alpha < 1$, and apply Cantelli's inequality [89], we see that

$$\mathrm{Pr}_{\phi \sim \Phi}(\mathrm{J} \geq \alpha \mathrm{J}_{\mathrm{max}}) \leq \frac{\mathrm{Var}_{\phi \sim \Phi}[\mathrm{J}]}{\mathrm{Var}_{\phi \sim \Phi}[\mathrm{J}] + (\alpha \mathrm{J}_{\mathrm{max}} - \mathbb{E}_{\phi \sim \Phi}[\mathrm{J}])^2}$$

Minimizing $\mathrm{Var}_{\phi \sim \Phi}[\mathrm{J}]$ in addition to $\mathbb{E}_{\phi \sim \Phi}[J]$ will correlate with decreasing this upper bound. As a result, we expect variance regularization to correlate with decreased probability of encountering near-worst-case performance.

We can also justify the connection between variance regularization and reducing sensitivity L by looking at the special case where $J \circ S$ is Lipschitz and the elements of ϕ are independent. The Bobkov-Houdré variance bound for Lipschitz functions [90] holds that $\mathrm{Var}_{\phi \sim \Phi}[\mathrm{J}] \leq \mathrm{L}^2 \sigma^2_{\Sigma \phi}$, where $\sigma^2_{\Sigma \phi}$ is the variance of the sum of elements in ϕ. This bound does not guarantee that minimizing $\mathrm{Var}_{\phi \sim \Phi}[\mathrm{J}]$ decreases L, but it suggests a correlation between these quantities.

3.4 Experimental results

So far, we have developed the theoretical and algorithmic basis for our robot design framework. It remains for us to empirically answer two questions: first, is our framework useful for solving practical robot design problems? Second, is our statistical method for robustness analysis sound?

In this section, we answer these questions through the lens of two case studies. The first involves finding optimal sensor placements for robot navigation, and the second involves optimizing a pushing strategy for multi-agent manipulation. We demonstrate the success of our optimization and robustness analysis framework on each example, and we provide results from hardware testing in both cases. Next, we include an ablation study justifying our use of automatic differentiation and variance regularization. We conclude by verifying the soundness of our statistical robustness analysis.

3.4.1 Case study: optimal sensor placement for navigation

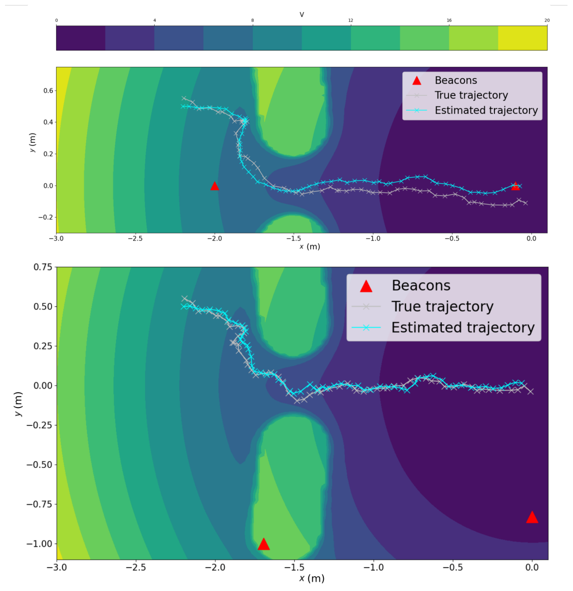
First, we return to the AGV localization and navigation example introduced in Fig. 3.2. This design problem requires finding an optimal placement for two ranging beacons to minimize estimation error and allow the robot to safely navigate between two obstacles. Range measurements from these beacons are integrated with IMU data via an EKF, and the resulting state estimate is used as input to a navigation function and tracking feedback controller to guide the robot to its goal. This design problem has two important features. First, it involves interactions between multiple subsystems: the output from the EKF is used by the navigation function, which feeds input to the controller, which in turn influences future EKF predictions. Second, the effect of uncertainty on the robot's performance is relatively strong.

The design parameters are the (x, y) locations of two range beacons and two feedback controller gains (6 total design parameters). The exogenous parameters include uncertainty in the robot's initial state along with actuation and sensing noise at each of T timesteps ($3+6T$ total exogenous parameters). The cost function has three components: one penalizing large estimation errors, one penalizing deviations from the goal, and one penalizing collisions with the environment. A formal definition of the design and exogenous parameters, simulator, cost, and constraints is given in Table A.1 in the appendix. The simulator and cost functions are implemented in Python using the JAX framework for automatic differentiation.

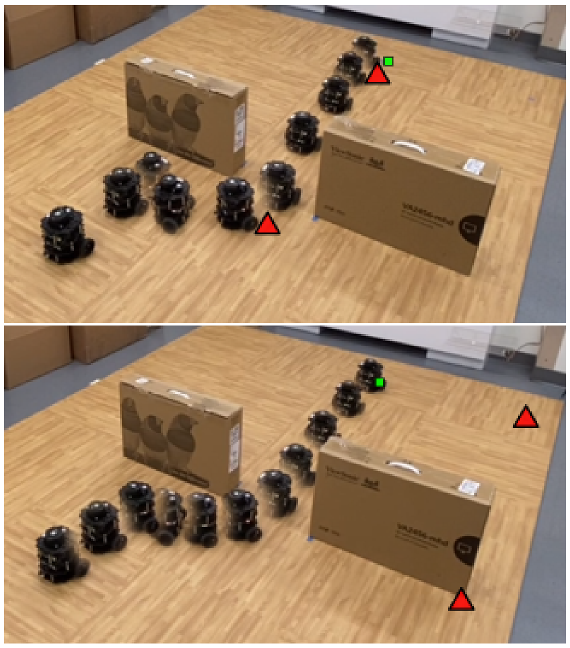
Fig. 3.4a compares simulated trajectories for the initial and optimized beacon placements and feedback gains, clearly showing the impact of design optimization. Initially, poor beacon placement causes the robot to accumulate estimation error and drift away from its goal. The optimized design moves the beacons off to the side to eliminate this drift. Optimization ($N = 512$, $\lambda = 0.1$, L-BFGS-B back-end) takes 3 minutes 34 s on a laptop computer (8 GB RAM, 1.8 GHz 8-core processor).

We test the initial and optimized design in hardware using the TurtleBot 3 platform. To emulate range beacon measurements in our lab, odometry and laser scan data are fused into a full state estimate from which range measurements are derived (the full state estimate is hidden from the robot, which only received the emulated range measurements). The control frequency is increased from 2 Hz in simulation to 10 Hz in hardware, and the obstacles are recreated in our laboratory. The hardware results, shown in Figs. 3.4b and 3.5a, confirm our simulation results: the initial design suffers from drift and ends approximately 10 cm from its target position, while the optimized design does not drift and ends within 5 cm of the goal. This difference can be seen most clearly in the posterior error covariance from the EKF; Fig. 3.5a shows how the optimized design greatly reduces uncertainty in the state estimate compared to the initial design. No parameter estimation or tuning on hardware was required.

We then apply the robustness analysis from Section 3.3 to certify the maximum absolute estimation error $\|x_t - \hat{x}_t\|$ in the optimized design (in meters, projected into the xy plane). Note that this error is different from the cost used during optimization, but we can still apply Algorithm 3.3.1 simply by changing the cost function for the duration of the analysis. Using block size $N = 1000$ and sample size $M = 1000$, we fit a GEVD using Algorithm 3.3.1 to the maximum estimation error for both the initial and optimized designs. These distributions are shown in Fig. 3.5b; the optimized design significantly reduces the expected maximum estimation error. We observe that the 97% confidence level for the shape parameter $\xi = 0.059$ is positive, so we cannot conclude that the worst-case estimation error is bounded, but we can derive a high-confidence bound of 0.21 m for our optimized design.



(a) Simulated trajectories for the initial (top) and optimized (bottom) AGV designs. Navigation function contours are shown in color.

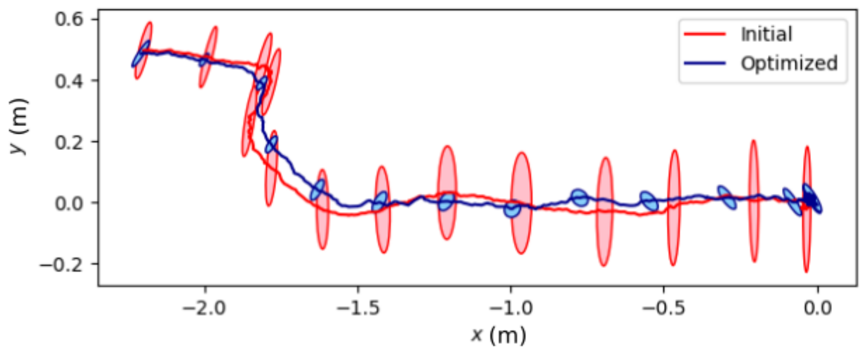


(b) Hardware performance of initial (top) and optimized (bottom) AGV designs. Square (green) shows the goal; triangles (red) show beacon locations.

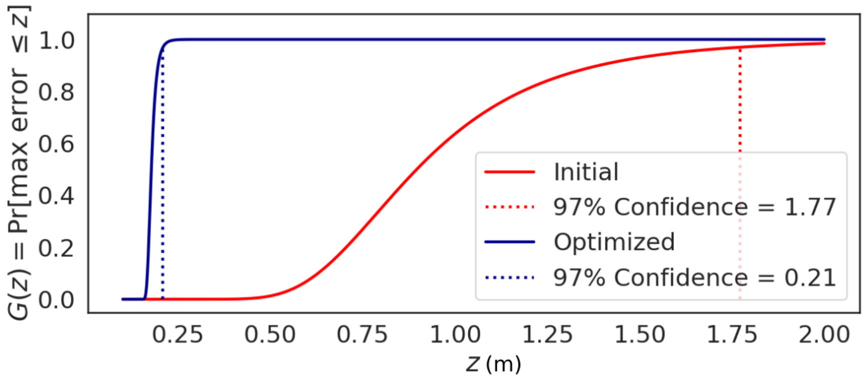
Figure 3.4: Simulation and hardware validation of optimized AGV designs, showing how the optimized design eliminates drift relative to goal.

3.4.2 Case study: collaborative multi-robot manipulation

Our second example involves finding a control strategy for multi-agent collaborative manipulation. In this setting, two ground robots must collaborate to push a box from its current location to a target pose (as in Fig. 3.6a). Given the desired box pose and the current location of each robot, a neural network plans a trajectory for each robot, which the robots then track using a feedback controller (θ includes both the neural network parameters and the tracking controller gains, with a total of 454 design parameters). The exogenous parameters include the coefficient of friction for each contact pair, the mass of the box, the desired pose of the box, and the initial pose for each robot (a total of 13 exogenous parameters; we vary the desired box pose and initial robot poses to prevent over-fitting during optimization). The cost function is simply the squared error between the desired box pose (including posi-



(a) Hardware results for EKF state estimates and posterior error covariance 3σ ellipse for initial and optimized AGV designs.

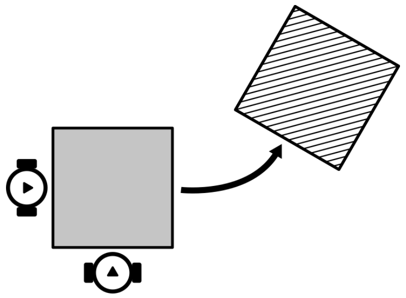


(b) GEVD CDF fit using Algorithm 3.3.1 for the maximum absolute estimation error in the xy-plane in both the initial and optimized AGV designs, with 97% confidence levels.

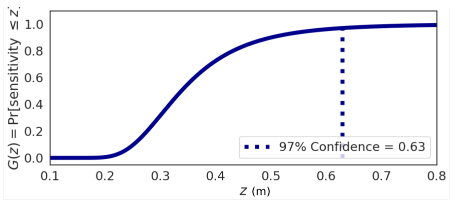
Figure 3.5: Additional validation of optimized AGV designs.

tion and orientation) and its true final pose after a 4 s simulation. A full definition of this design problem and contact dynamics model is included in Table A.2 in the appendix. We implement the contact dynamics simulator, trajectory planning neural network, and path tracking controller in Python using JAX.

Compared to the design problem in our first case study, this system has a simpler architecture (fewer subsystems) but more complicated dynamics and a much higher-dimensional design space. This example also showcases a different interpretation of the exogenous parameters: instead of representing true sources of randomness, these parameters represent quantities that are simply unknown at design-time. For example, the target position for the box is not random in the same way as sensor noise in the previous example, but since we



(a) Multi-agent manipulation design optimization problem. The goal is to find parameters for robot controllers and a neural network planner that push the box from an initial position (solid) to a desired position (striped).



(b) GEVD CDF fit using Algorithm 3.3.2 for the maximum sensitivity of the optimized collaborative manipulation strategy to variation in friction coefficient. z has units of meters per unit change in friction coefficient.

Figure 3.6: Setup and robustness analysis for the multi-agent manipulation problem.

cannot choose this value at design-time it must be included in ϕ. As a result, minimizing the expected cost with respect to variation in ϕ yields a solution that achieves good performance for many different target poses, enabling the user to select one at run-time and be confident that the design will perform well.

To solve this design problem, the neural network parameters are initialized i.i.d. according to a Gaussian distribution, and the tracking controller gains are set to nominal values. We then optimize the parameters using $N = 512$, $\lambda = 0.1$, and L-BFGS-B back-end. This optimization took 45 minutes 32 s on a laptop computer (8 GB of RAM and a 1.8 GHz 8-core processor). Fig. 3.7 shows a comparison between the initial and optimized strategies, and Fig. 3.8 in the appendix shows additional examples of the optimized behavior. The target pose is drawn uniformly $[x, y, \theta] \in [0, 0.5]^2 \times [-\pi/4, \pi/4]$, and the optimized design achieves a mean squared error of 0.0964.

We test the optimized design in hardware, again using the TurtleBot 3 platform. An overhead camera and AprilTag [91] markers are used to obtain the location of the box and each robot. At execution, each robot first moves to a designated starting location near the box, plans a trajectory using the neural network policy, and tracks that trajectory at 100 Hz until the box reaches its desired location or a time limit is reached. Results from this

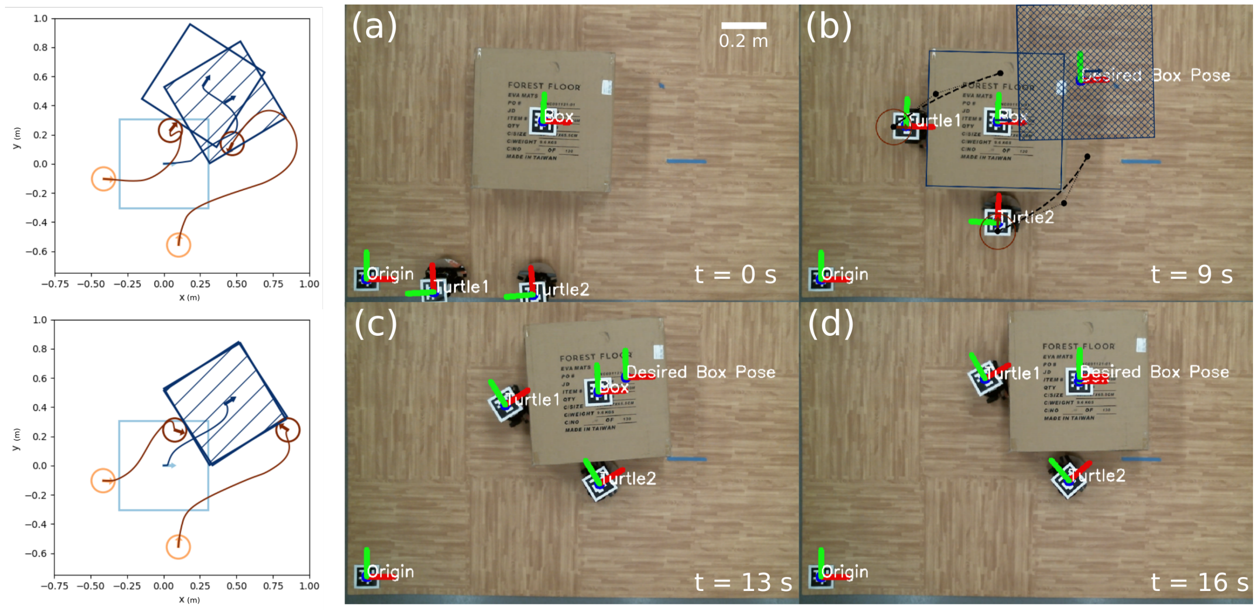


Figure 3.7: Left: Initial (top) and optimized (bottom) manipulation strategies in simulation (light/dark colors indicate initial/final positions, stripes indicate desired position). Right: Optimized manipulation strategy deployed in hardware. (a) The robots first move to positions around the box. (b) Using the optimized neural network, the robots plan a cubic spline trajectory pushing the box to its desired location. (c-d) The robots execute the plan by tracking that trajectory.

hardware experiment are shown in Fig. 3.7. Again, no parameter tuning or estimation on hardware was needed.

After successfully testing the optimized design in the laboratory, it is natural to ask how its performance might change as conditions (particularly the coefficients of friction) change. Using $M = 500$ blocks of size $N = 1000$ each, we use Algorithm 3.3.2 to fit a GEVD for the sensitivity constant L with respect to the coefficients of friction between each contact pair. We do this by allowing these coefficients to vary and freezing other elements of ϕ at nominal values (box mass 1 kg and target pose $[0.3, 0.3, 0.3]$). The fit distribution is shown in Fig. 3.6b. The 97% confidence level for the shape parameter is $\xi = 0.118 > 0$, so we cannot conclude that the performance of our design is Lipschitz with respect to the friction coefficients, but we can estimate the 97% confidence level for L as 0.63.

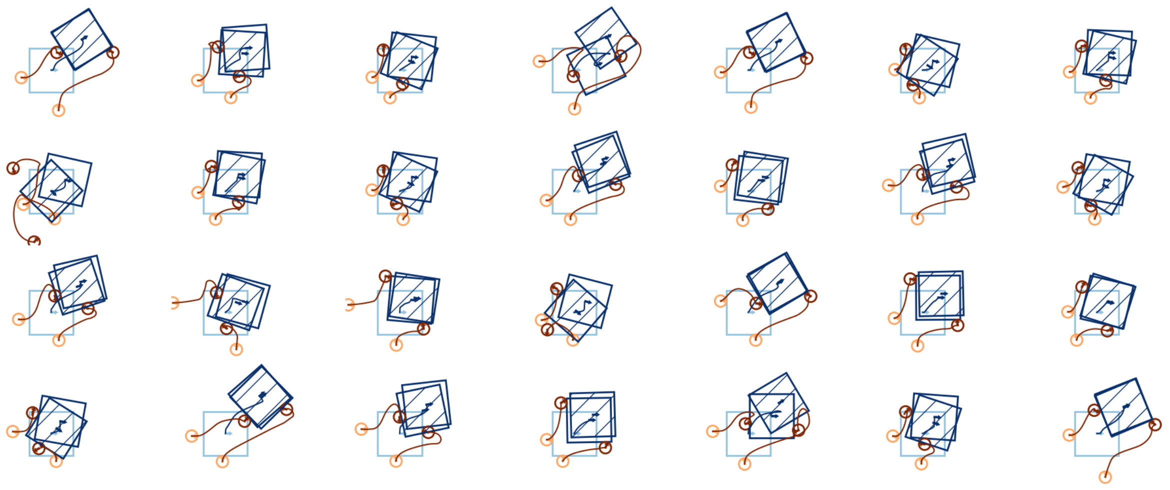
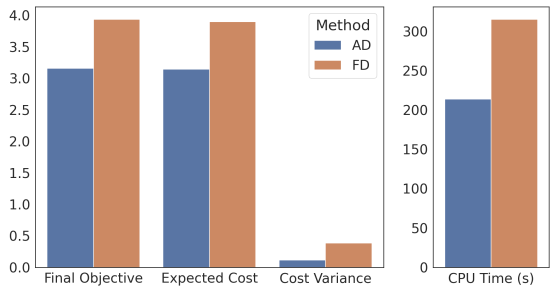


Figure 3.8: Additional examples of optimized multi-agent manipulation behavior in simulation, showing that the optimized strategy reaches the goal in most cases. Each example shows the results of executing the optimized pushing strategy for 4 s with a randomly selected set of friction coefficients, random target pose, and random initial robot poses. Light/dark colors indicate initial/final positions, respectively, and the striped box indicates the target pose.

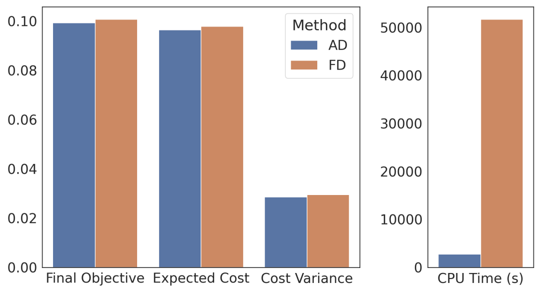
3.4.3 Design optimization ablation study

Our case studies in Sections 3.4.1 and 3.4.2 help demonstrate the utility of our framework for solving realistic robotics problems. However, it remains to justify the choices we made in designing this framework. For instance, how does automatic differentiation compare with other methods for estimating the gradient (e.g. finite differences)? What benefit does variance regularization in problem (3.2) bring? We answer these questions here using an ablation study where we attempt to isolate the impact of each of these features.

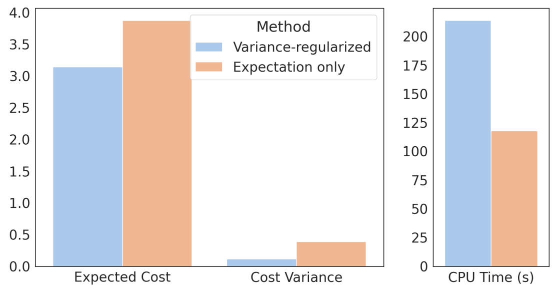
First, why use automatic differentiation? On the one hand, AD allows us to estimate the gradient with only a single evaluation of the objective function, while other methods (such as finite differences, or FD) require multiple evaluations. On the other hand, AD necessarily incurs some overhead at runtime, making each AD function call more expensive than those used in an FD scheme. Additionally, some arguments [69] suggest that exact gradients may be less useful than finite-difference or stochastic approximations when the objective is stiff



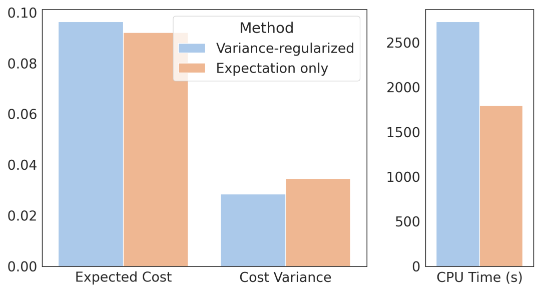
(a) AD vs. FD; sensor placement



(b) AD vs. FD; manipulation



(c) Effect of VR; sensor placement



(d) Effect of VR; manipulation

Figure 3.9: (a)-(b) Improvement of automatic differentiation (AD) over finite differences (FD) in both case studies. (c)-(d) Effect of variance regularization (VR) in both case studies.

or discontinuous. We compare AD with a 3-point finite-difference method by re-solving problem (3.2) for both case studies, keeping all parameters constant ($N = 512$, $\lambda = 0.1$, same random seed) and substituting the gradients obtained using AD for those computed using finite differences. Fig. 3.9 shows the results of this comparison. In the sensor placement example, AD achieves a lower expected cost and cost variance, and it runs in 32% less time. In the collaborative manipulation example, both methods achieve similar expected cost and variance, but the AD version runs nearly 19x faster. These results lead us to conclude that AD enables more effective optimization than finite differences and is an appropriate choice for our framework.

The next question is whether variance regularization brings any benefit to the design optimization problem. To answer this question, we compare the results of re-solving both case studies with variance weight $\lambda = 0.1$ and $\lambda = 0$. These results are shown in Fig. 3.9; surprisingly, in the sensor placement example we see that the variance-regularized problem results in a lower expected cost, contrary to the intuition that regularization requires a trade off with increased expected cost. We expect that this lower expected cost may be a result of the regularization term smoothing the objective with respect to the exogenous parameters. However, these benefits are less pronounced than the benefits from automatic differentiation, and we do not see a distinct benefit from variance regularization in our second case study.

3.4.4 Accuracy of robustness analysis

To verify the soundness of our statistical robustness analysis methods, we need to determine whether the fit GEVD is likely to either under- or overestimate the worst-case performance of a design. To answer this question, we compare the cumulative distribution function (CDF) of the fit GEVD with an empirical CDF observed from data. Algorithms 3.3.1 and 3.3.2 both estimate a posterior distribution for μ, σ, and ξ, allowing us to construct an upper-bound and lower-bound GEVD using the 97% and 3% confidence level parameter estimates. Using these distributions, we can measure false optimism and conservatism using a one-sided Kolmogorov-Smirnov (KS) test [92].

Fig. 3.10 compares the estimated GEVDs and empirical data for worst-case performance in the sensor placement example (fit using Algorithm 3.3.1) and sensitivity in the manipulation example (fit using Algorithm 3.3.2). In the former case, we see that the empirical CDF lies between the upper- and lower-confidence limits for the fit distribution, indicating that the fit is neither falsely optimistic at the 97% level nor conservative at the 3% level (these conclusions are confirmed by the KS statistics provided in Table 3.1). In the second case study, even though the empirical CDF extends slightly beyond the estimated bounds in some regions, the statistical analysis in Table 3.2 indicates that the estimated GEVD is neither falsely optimistic at the 97% level nor conservative at the 3% level. In addition, we see that the gap between the 3% and 97% distributions is relatively small in both examples in Fig. 3.10.

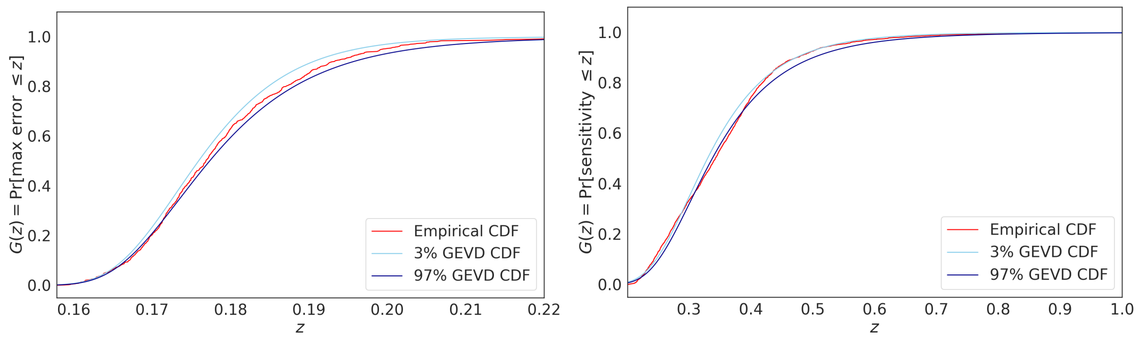


Figure 3.10: Comparison of fit GEVD CDFs and empirical CDF for worst-case estimation error in the sensor placement example (top) and sensitivity in the manipulation example (bottom).

	Null Hypothesis	KS Statistic	p-value	Conclusion ($p < 0.05$)
False Optimism	97% GEVD under-estimates worst-case performance	0.0410	0.0337	Reject; 97% GEVD *does not* under-estimate worst-case performance
Conservatism	3% GEVD over-estimates worst-case performance	0.0529	0.00354	Reject; 3% GEVD *does not* over-estimate worst-case performance

Table 3.1: Results of one-sided KS tests for the sensor placement case study. These results indicate that Algorithm 3.3.1 is sound in this case.

3.5 Summary

In this chapter, we introduce an automated design tool to improve the productivity of robot designers by a) enabling efficient optimization of robot designs and b) allowing users to certify the robustness of those designs. In developing this framework, we make two main contributions. First, we use differentiable programming for end-to-end optimization of robotic systems, creating a flexible software framework for design optimization. Second, we develop a novel statistical framework for certifying the worst-case performance and sensitivity of optimized designs. We apply these tools to optimize the design of two robotic systems in hardware. We use our statistical certification method to test the robustness of these designs, and we show that the optimized designs are robust enough to deploy in hardware. We hope

	Null Hypothesis	KS Statistic	p-value	Conclusion ($p < 0.05$)
False Optimism	97% GEVD under-estimates sensitivity	0.0399	6.75×10^{-5}	Reject; 97% GEVD *does not* under-estimate sensitivity
Conservatism	3% GEVD over-estimates sensitivity	0.0618	1.03×10^{-10}	Reject; 3% GEVD *does not* over-estimate sensitivity

Table 3.2: Results of one-sided KS tests for the collaborative manipulation case study. These results indicate that Algorithm 3.3.2 is sound in this case.

that by combining flexible design optimization with robustness certification, this framework makes the first steps towards developing the automated design aid for robotics engineers envisioned by this thesis.

That said, there are several limitations with the approach presented in this chapter. First, since our approach relies on sampling from Φ without any further information, it will require a large number of samples to accurately capture rare events. As we will see in the following chapters, we can close this gap when more information about Φ is available by using adversarial optimization and Bayesian inference techniques. Second, the statistical verification method yields an estimate of the worst-case cost, but it does not provide any insights into what environmental factors could lead to that worst-case performance. Finally, the optimization approach presented in this chapter is based on gradient descent, which is inherently local (i.e. it risks converging to a suboptimal solution) and prone to failing when the gradients of $J \circ S$ are not well behaved. We will address both of these limitations in the following chapters, starting with an adversarial optimization method that reduces the number of samples from Φ required for robust solutions, and then discussing a novel reformulation of the end-to-end design optimization problem that allows us to search for (approximately) globally optimal solutions and avoid issues due to poorly conditioned gradients.

Chapter 4

Counterexample-guided Optimization with Formal Specifications

In the previous chapter, we saw how automatic differentiation can enable end-to-end design optimization for a range of robotic systems. However, the optimization algorithm presented in Chapter 3 relies on a variance-regularization heuristic to encourage the robustness of the optimized designs. There are two issues with this approach, which we address in this chapter. First, estimating the cost variance requires a large number of samples of the exogenous parameters ϕ. Second, the methods presented in Chapter 3 provide a statistical estimate of the worst-case cost, but they do not provide concrete counterexamples; i.e., specific values of ϕ that lead to high cost.

In this chapter, we address both of these issues through the development of a *counterexample-guided* design optimization pipeline that replaces the variance regularization used in Chapter 3 with an adversarial optimization procedure that alternates between optimizing the design θ and a set of adversarial counterexamples ϕ. In addition, we use differentiable temporal logic to allow users to formally specify the desired behavior of the robot. This chapter is based on the author's published work [93].

There are different types of temporal logic that may be used to specify robot behaviors, but here we focus on *signal temporal logic* (STL), a formal language for specifying properties of real-values continuous-time signals [94]–[96]. Using STL, users can specify a variety of robot behaviors by combining logical and temporal operators to specify the desired order and dependencies between subtasks [95], [97], [98]. Although the syntax of STL can be opaque, there are tools for translating between STL and natural language [99].

A number of previous works have approached the problem of synthesizing robot behaviors from STL specifications, using either abstraction-based methods [97], mixed-integer optimization [95], [100], sampling-based planning [101], [102], or nonlinear optimization [66], [103], [104]. Abstraction-based methods first discretize the state space, then plan on this discrete domain; these methods have a long history but suffer from exponential dependence on problem dimension. Mixed-integer optimization-based methods encode the STL specification as linear constraints with integer variables, resulting in soundness and completeness guarantees, but the resulting mixed integer programs (MIPs) become intractable for problems with a large state space or long task horizon [100], [105], [106]. The size of the MIP can be reduced using a timed waypoint representation, but this requires restrictive assumptions (e.g. access to a tracking controller with bounded error [95]).

Some recent work uses nonlinear optimization to solve robot planning problems with STL specifications [66], [96], [98], [104], [107]. These methods achieve improved scalability through the use of smooth approximations of STL and local gradient-based optimization methods, but they sacrifice completeness and optimality guarantees and can result in solutions with poor robustness (as we demonstrate via our experiments later in this section).

The primary gap in the state-of-the-art for robot optimization with STL constraints is that existing methods do not explicitly consider robustness to environmental uncertainty or disturbances ([107] considers probability of satisfaction, but not robustness to worst-case uncertainty). Existing methods implicitly encourage robustness by maximizing the margin by which the STL specification is satisfied, but in practice we find that this is not sufficient to prevent failure when confronted with environmental uncertainty. Some methods do explicitly consider worst-case robustness [106], but our experiments indicate that these methods yield MIPs that are too computationally expensive to solve in practice.

In this chapter, we address this gap by extending the end-to-end optimization method presented in Chapter 3 to use counterexamples (i.e. a set of adversarially-chosen environmental parameters ϕ) to guide the optimization of the design parameters θ. This method uses an iterative adversarial optimization scheme inspired by solution methods for multi-player games, alternating between searching for a design that performs well against a set of known counterexamples and searching for new counterexamples to guide the design process. This approach relies on differentiable simulation, as discussed in Chapter 3, with the addition of differentiable temporal logic for formal task specifications. As our method relies on nonlinear gradient-based optimization, it will not recover the optimality or completeness of formal synthesis tools, but our primary aim is to improve the robustness of the resulting solutions.

This chapter is organized as follows: after briefly reviewing the syntax and semantics of STL, we extend the problem statement from Chapter 3 to include STL specifications. We then present our approach to counterexample-guided optimization, and conclude with numerical experiments that demonstrate the improved performance of our method relative to state-of-the-art mixed-integer and nonlinear optimization methods. We find that our approach not only yields designs that satisfy the STL specification despite worst-case environmental uncertainty, but also that it requires less than half of the optimization time as the next-most-successful method. Our counterexample-guided method scales to handle long-horizon tasks that are not tractable for MIP-based methods, and the designs found using our approach are consistently more robust than those generated by competing methods.

4.1 Background on signal temporal logic

Recall from Chapter 3 that we described the behavior of the system by the discrete trace of states over a fixed horizon T: $s_1, \dots, s_T$. In this chapter, we extend this view by considering behaviors in continuous time $s(t)$, which for convenience we represent as piecewise-linear interpolation between timed samples (t_i, s_i). Moving from a discrete sequence of states to a piecewise-linear continuous signal allows us to be more precise in specifying the desired robot behavior, since we can now be explicit about the state of the robot in between sampled times while maintaining a finite-dimensional representation.

Given this representation, we can specify the desired behavior of a robot using signal temporal logic (STL). STL is a formal language for specifying how a signal should evolve over time, and it allows us to express complex requirements for how a robot behaves over time. The rest of this section will cover formal STL syntax (how requirements are written) and semantics (how they are interpreted).

4.1.1 Syntax of signal temporal logic

STL requirements are expressed as *formulas*, which can be build out of three basic building blocks: predicates, logical operators, and temporal operators. Predicates are functions that map a continuous-time signal to a Boolean value, and they can be used to express properties of the robot's state or the environment (e.g. is a robot in a given state at a given time). Logical operators are used to combine predicates, and temporal operators are used to express how predicates should evolve over time [108]. We can define the syntax of an STL formula ψ recursively as:

$$\psi = \text{true} \mid \mu(x) \geq 0 \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \psi_1 \; \mathcal{U}_I \; \psi_2 \tag{4.1}$$

with closed (but potentially unbounded) time interval I, predicate $\mu : \mathcal{S} \to \mathbb{R}$ mapping states to real numbers, logical operators $\neg$ (negation) and $\wedge$ (conjunction), and temporal operator "until" $\mathcal{U}_I$, which can be read as "within interval I, ψ_1 must be true until ψ_2 becomes true". For convenience, we assume $I = [0, \infty)$ when not explicitly specified. Additional temporal operators can be defined in terms of these basic building blocks:

1. Eventually: $\Diamond_I \psi = \text{true}\, \mathcal{U}_I\, \psi$; read as "$\psi$ must be true at least once during I".

2. Always: $\Box_I = \neg \Diamond_I \neg\psi$; read as "$\psi$ must be true at all times during I".

Similarly, the usual suite of logical operators (e.g. or, implies) can be defined in terms of the basic logical operators. The syntax of STL can be opaque for unfamiliar readers; wherever possible in this thesis, STL specifications will be accompanied by natural language explanations.

4.1.2 Semantics of signal temporal logic

There are two related ways of interpreting any given STL formula: the Boolean and quantitative semantics. Given an STL formula ψ, the Boolean semantics assign a simple true/false value to a signal s at a particular time t indicating whether the STL formula is satisfied at that time [108]:

$$\begin{aligned}
&s, t \models \text{true} \\
&s, t \models \mu(x) \geq 0 && \text{iff } \mu(s(t)) \geq 0 \\
&s, t \models \neg\psi && \text{iff } s, t \not\models \psi \\
&s, t \models \psi_1 \wedge \psi_2 && \text{iff } s, t \models \psi_1 \text{ and } s, t \models \psi_2 \\
&s, t \models \psi_1\, \mathcal{U}_I\, \psi_2 && \text{iff } \exists\, t' \in t + I \text{ s.t. } w, t' \models \psi_2 \\
& && \qquad \text{and } w, t'' \models \psi_1\ \forall\, t'' \in [t, t']
\end{aligned}$$

A useful feature of STL is that it also admits a so-called *quantitative semantics*. Intuitively, if the Boolean semantics tell us whether an STL formula is satisfied by a given signal at a given time, the quantitative semantics tell us how well the signal satisfies the formula (i.e. by how much does it exceed the specified requirements, or by how much does it fall short). We denote the quantitative semantics as $\rho\colon \Psi \times \mathcal{S} \times \mathbb{R}^+ \to \mathbb{R}$, where $\rho(\psi, s, t)$ is the *robustness* of the signal s with respect to the formula ψ at time t. The robustness is a real number that quantifies how well the signal satisfies the formula at that time; the formula is satisfied precisely when $\rho \geq 0$. The robustness function is defined recursively as:

$$
\begin{aligned}
\rho(\text{true}, s, t) &= \top \\
\rho(\mu(x) \geq 0, s, t) &= \mu(s(t)) \\
\rho(\neg\psi, s, t) &= -\rho(\psi, s, t) \\
\rho(\psi_1 \wedge \psi_2, s, t) &= \min\{\rho(\psi_1, s, t), \rho(\psi_2, s, t)\} \\
\rho(\psi_1 \mathcal{U}_I\ \psi_2, s, t) &= \sup_{t_1 \in t+I} \min\{\rho(\psi_2, s, t_1), \inf_{t_2 \in [t, t_1]} \rho(\psi_1, s, t_2)\}
\end{aligned}
$$

where $\top$ is a constant defined to be greater than all other real values. In practice, linear-time algorithms exist to compute ρ from a given piecewise-affine signal s [108].

4.2 Problem statement

In this chapter, we extend the design optimization problem (3.2) from Chapter 3 in two ways: first, we incorporate STL into the cost function to specify the desired robot behavior. Second, we replace the variance regularization in (3.2a) with a robust adversarial formulation, which we will solve using counterexample-guided optimization. These two changes yield the robust design problem:

$$
\begin{aligned}
\min_{\theta}\ \max_{\phi}\quad & \lambda J(\theta, \phi) - \rho\left(\psi, S(\theta, \phi), 0\right) && \text{(4.2a)} \\
\text{s.t.}\quad & c_{\theta,i}(\theta) \geq 0 \quad \forall i \in \mathcal{I}_\theta && \text{(4.2b)} \\
& c_{\phi,i}(\phi) \geq 0 \quad \forall i \in \mathcal{I}_\phi && \text{(4.2c)}
\end{aligned}
$$

where $\rho\left(\psi, S(\theta, \phi), 0\right)$ is the robustness margin at the starting time $t = 0$ of the system trace simulated with parameters θ and ϕ with respect to STL specification ψ. We subtract ρ from a generic cost function J (weighted by $\lambda > 0$) to balance maximizing STL robustness against minimizing other costs (e.g. fuel use). The scaling factor λ is typically small to prioritize satisfying the STL specification. For convenience, we denote

$$J_\psi(\theta, \phi) = J(\theta, \phi) - \rho\left(\psi, S(\theta, \phi)\right). \tag{4.3}$$

In this adversarial formulation, we also include additional constraints (4.2c) to restrict the values of the exogenous parameters; often, unbounded exogenous parameters can lead to unbounded cost (e.g. large external disturbances that cause a robot to deviate arbitrarily far from a planned path), so bounding the exogenous parameters helps with the stability of this adversarial optimization problem.

4.3 Approach

Problem (4.2) is a nonlinear optimization that cannot generally be solved to global optimality. Instead, we take advantage of the two-player game structure of this problem to design an iterative algorithm to find a local *generalized Nash equilibrium*; i.e., the design parameters θ and corresponding adversarial exogenous parameters ϕ such that neither the designer nor the adversary have an incentive to change [7].

In order to solve this problem, we must address three challenges. First, in order to efficiently solve this nonlinear optimization problem, we must be able to compute gradients of ρ with respect to θ and ϕ; we address this problem using differentiable programming, as discussed in Section 4.3.1. Second, even when gradients are available, standard gradient-based adversarial optimization approaches may encounter stability problems in cases where no pure equilibria exist (i.e. the adversary and designer oscillate through a cycle of best-response solutions without converging). Finally, even if gradient-based solution methods converge, there is a risk that the resulting local Nash equilibrium will not be robust; i.e., overfitting the design θ to a particular adversarial ϕ so that the design performs poorly when ϕ changes. We address these last two problems by developing a meta-heuristic for counterexample-guided optimization, which we discuss in Section 4.3.2.

4.3.1 Differentiable signal temporal logic

Although nonlinear problems like (4.2) can be solved without derivatives of ρ, for example using zero-order gradient estimators [69] or black-box optimizers [14], it is often much more efficient to make use of gradients when they are available. Despite their usefulness, exact gradients are difficult to derive symbolically for complex problems; instead, we extend our work on differentiable simulation in Chapter 3 to include differentiable approximations of STL robustness.

As defined above, ρ is a continuous but non-smooth function of s, t, and all parameters of the formula ψ. Smooth approximations to ρ can be derived by replacing the min and max operators with smooth approximations:

$$\min(a,b) \approx \widetilde{\min}_\gamma(a,b) = \frac{1}{\gamma}\log\left(e^{\gamma a} + e^{\gamma b}\right) \tag{4.4}$$

$$\max(a,b) \approx \widetilde{\max}_\gamma(a,b) = -\widetilde{\min}_\gamma(-a,-b) \tag{4.5}$$

where $\gamma > 0$ controls the degree of smoothing; $\lim_{\gamma\to\infty}\widetilde{\min}_\gamma(a,b) = \min_\gamma(a,b)$. This approximation was introduced in [96] and later used in [103], [104]; other works [66] use a related but slightly different approximation.

Using these smooth approximations, we implement a fast, linear-time algorithm for computing the differentiable robustness margin, based on [108]. We use the JAX framework [64] for efficient automatic differentiation and just-in-time compilation. In contrast to the implementation in [66], where the time complexity of evaluating the $\mathcal{U}$ operator scales quadratically with signal length, our method achieves linear time complexity. In the next section, we discuss how combining this differentiable robustness margin with gradients from differentiable simulation allows us to solve (4.2) using an iterative algorithm for counterexample-guided optimization.

4.3.2 Counterexample-guided optimization

As mentioned above, to solve the robust optimization problem in (4.2), we must find a generalized Nash equilibrium between the designer and the adversary. Moreover, this equilibrium should ideally be robust in that the design θ should perform well against a wide range of adversarial exogenous parameters ϕ. A common approach to finding Nash equilibria in two-player games like (4.2) is the family of nonlinear Gauss-Seidel-type methods [7], which solve min-max problems alternating between optimizing θ and ϕ. These methods tune each set of parameters in turn while holding the other constant; i.e. solving a sequence of optimization problems:

$$\theta^*_{i+1} = \arg\min_{\theta} J_\psi(\theta, \phi^*_i) \tag{4.6a}$$

$$\phi^*_{i+1} = \arg\max_{\phi} J_\psi(\theta^*_{i+1}, \phi) \tag{4.6b}$$

If this sequence converges (which is not guaranteed), then the stationary point (θ^*, ϕ^*) will be a local Nash equilibrium [7].

The first risk in using this simple iterative algorithm is that the optimization for both θ and ϕ can get stuck in local minima. These local minima not only lead to suboptimal performance of the design θ^*, but also "overconfidence" when θ^* is overfit to a particular value of ϕ^*, which might cause the designer to believe that the design is robust because it performs well against a single adversarial example when it may fail for nearby values of ϕ. The second risk is that this iterative algorithm may converge to a limit cycle rather than an equilibrium. In cases where the adversary's best response is highly sensitive to the current design of the system, a pure Nash equilibrium may not exist, or if it does exist it may not be found by iterative solution algorithms.

To mitigate these issues and improve the robustness of the optimized design, we propose an extended Gauss-Seidel method that takes inspiration from two ideas from the machine learning and optimization literature: domain randomization and counterexample-guided search [4], [109].

The core idea is to allow the adversary to adopt a mixed strategy; rather than committing to the most recent best response, we allow the adversary to propose a mixture of all best responses found during previous iterations. To do this, instead of optimizing θ against a single ϕ, we can maintain a dataset $\mathcal{D}_N = \{\phi_i\}_{i=1}^N$ of exogenous parameters, and optimize θ against all of them simultaneously.

$$\theta^* = \arg\min_\theta \mathbb{E}_{\mathcal{D}_N}\left[J(\theta, \phi_i)\right] \tag{4.7a}$$

$$\phi^* = \arg\max_\phi J(\theta^*, \phi) \tag{4.7b}$$

In addition to effectively allowing mixed strategies for the adversary, this approach also encourages the design to be robust to a wide range of adversarial exogenous parameters, rather than overfitting to a single value of ϕ. To populate this dataset, we use ϕ_i^* from successive iterations of the alternating Gauss-Seidel process as counterexamples to guide the optimization of θ.

Pseudo-code for this counterexample-guided iterative Gauss-Seidel optimization method is given in Algorithm 4.3.1. This algorithm begins by initializing a dataset with N_0 i.i.d. ϕ sampled randomly from Φ, then alternates between solving the two optimization problems in (4.7a). At each step, the adversary's current best counterexample ϕ^* is added to the dataset, and the process continues until either reaching a fixed point where the best counterexample does not change between iterations or a maximum number of steps is reached. As we demonstrate empirically in Section 4.4, this counterexample-guided optimization achieves substantially better sample efficiency than simple domain randomization, in that it finds designs θ^* that are more robust to disturbances despite using fewer samples of ϕ.

An important note is that Algorithm 4.3.1 is enabled by the automatic differentiation approach described in Section 4.3.1. Without access to the gradients of J_ψ, solving the subproblems on lines 4 and 5 would be much more difficult. Although previous works have used gradients of STL robustness with respect to θ, we are not aware of any prior work using gradients with respect to disturbance parameters ϕ for robust optimization. Some prior works have used counterexamples to guide mixed-integer solvers to plan from STL specifications [106], but as our experiments in Section 4.4 show, mixed-integer methods are not practical for long-horizon tasks. We find that solving even a single mixed-integer problem can take more than an hour, making it impractical to solve multiple such problems in an iterative robust optimization context.

Algorithm 4.3.1: Counterexample-guided Gauss-Seidel method for solving robust planning problems

```
   Input: Starting dataset size $N_0$, maximum number of iterations $M$
   Output: Optimized design parameters $\theta^*$, dataset of counterexamples $\mathcal{D}_N$
1  $\mathcal{D}_N \leftarrow N_0$ examples $\phi_i \in \mathcal{D}$ sampled uniformly i.i.d.
2  $\phi^*_{prev} \leftarrow \varnothing$
3  for $i \in \{1, \dots, M\}$ do
4      $\theta^* = \arg\min_\theta \mathbb{E}_{\mathcal{D}_N}\left[J(\theta, \phi_i)\right]$
5      $\phi^* = \arg\max_\phi J(\theta^*, \phi)$
6      if $\phi^* = \phi^*_{prev}$ then
7          break
8      $\phi^*_{prev} \leftarrow \phi^*$
9      Append $\phi^*$ to $\mathcal{D}_N$
10 return $\theta^*$, $\mathcal{D}_N$
```

4.4 Experiments

We validate our robust optimization method on two case studies involving a satellite rendezvous problem, which was originally proposed as a benchmark in [110]. We compare our method against state-of-the-art methods for planning and optimization from STL specifications, showing the robustness and scalability of our approach.

4.4.1 Benchmark problems

In the satellite rendering problem proposed in [110], a chaser satellite must maneuver to catch a target satellite. We can model the dynamics of the chaser satellite in the Clohessy-Wiltshire-Hill (CHW) coordinate system, which assumes a circular orbit and places the target satellite at the origin, with the x-axis pointing away from the Earth, the y-axis pointing along the target's orbit, and the z-axis pointing out of the orbital plane. In the CHW frame, the chaser satellite's dynamics are approximately linear, with state defined as positions p_x, p_y, p_z and velocities v_x, v_y, v_z. The control inputs are the thrusts u_x, u_y, u_z in the x, y, z directions, leading to dynamics:

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \\ \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ 3n^2 p_x + 2n v_y + u_x/m \\ -2n v_x + u_y/m \\ -n^2 p_z + u_z/m \end{bmatrix}$$

where $n = \sqrt{\mu/a^3}$ is the mean-motion parameter of the target, $\mu = 3.986 \times 10^{14}\,\mathrm{m}^3/\mathrm{s}^2$ is the Earth's gravitational constant, and $a = 353\,\mathrm{km}$ is the altitude of the target in low Earth orbit. $m = 500\,\mathrm{kg}$ is the mass of the chaser [110].

We define STL specifications for two tasks of increasing difficulty: a simple low-speed rendezvous and a more complex loiter-then-rendezvous mission, shown in Fig. 4.1. We define

STL specifications for each mission, ψ_1 and ψ_2:

$$\begin{aligned}
\psi_1 &= \psi_{\text{reach target}} \wedge \psi_{\text{speed limit}} \\
\psi_2 &= \psi_{\text{reach target}} \wedge \psi_{\text{speed limit}} \wedge \psi_{\text{loiter}} \\
\psi_{\text{reach target}} &= \Diamond\, (r \leq 0.1) \\
\psi_{\text{speed limit}} &= (r \geq 2.0)\, \mathcal{U}\, \Box\, (v \leq 0.1) \\
\psi_{\text{loiter}} &= \Diamond \Box_{[0, T_{obs}]}\, (2.0 \leq r \wedge r \leq 3.0)
\end{aligned}$$

where $r = \sqrt{p_x^2 + p_y^2 + p_z^2}$ and $v = \sqrt{v_x^2 + v_y^2 + v_z^2}$.

In natural language, these formulae read as follows: $\psi_{\text{reach target}}$ says that "the chaser eventually comes within 0.1 m of the target", $\psi_{\text{speed limit}}$ says that "once the chaser is within 2.0 m of the target, its speed cannot exceed 0.1 m/s", and ψ_{loiter} says that "the chaser should spend T_{obs} seconds between 2–3 m away from the target at some point during the mission". We use $T_{obs} = 10\,\text{s}$ in our experiments. The two missions are build from these building blocks: mission 1 includes the "reach target" and "speed limit" requirement, while mission 2 includes all three requirements.

For each mission, the design parameters θ define the planned trajectory for the chaser (represented as state and input waypoints) as well as the feedback control gains used to track that trajectory, while the exogenous parameters ϕ represent bounded uncertainty in the initial state of the chaser ($p_x(0), p_y(0) \in [10, 13]$, $p_z(0) \in [-3, 3]$, $v_x(0), v_y(0), v_z(0) \in [-1, 1]$). We simulate both missions for 200 s with a 2 s timestep. For each mission, we include a penalty on the total impulse I (in Newton-seconds) required for the maneuver, with $J_\psi = \rho(\psi, S(\theta, \phi), 0) + \lambda I$ and $\lambda = 5 \times 10^{-5}$.

4.4.2 Baselines

We compare our approach against two baselines: an MIP planner based on that in [106] and [105] and the nonlinear optimization approach from [96], [104].

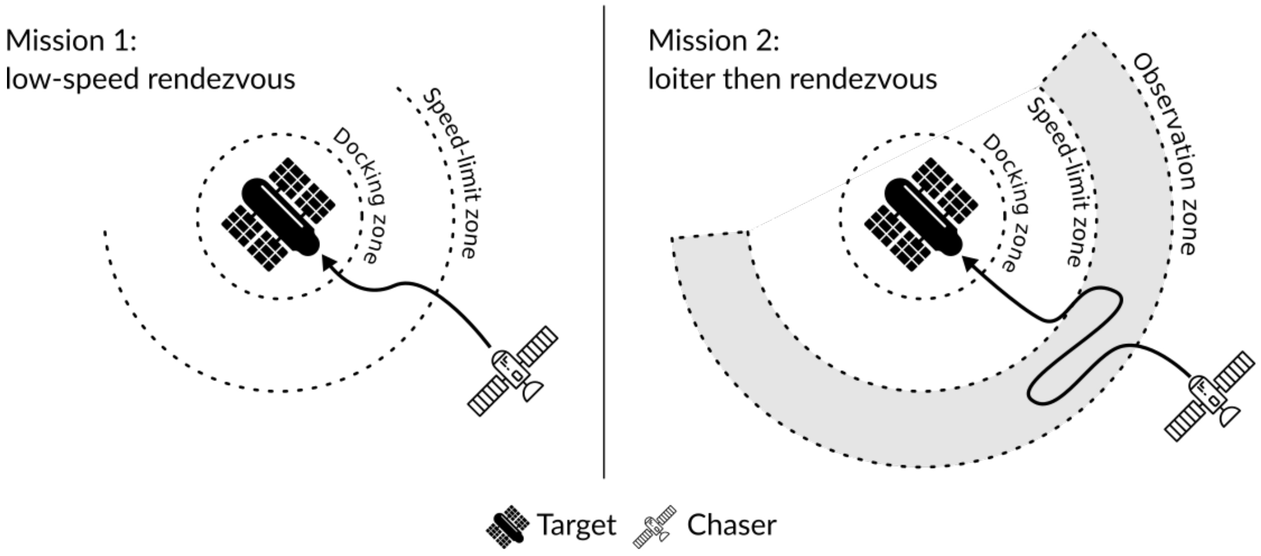


Figure 4.1: Two satellite rendezvous missions used in our experiments. In the first mission, the chaser satellite must eventually reach the target while respecting a maximum speed constraint in the region immediately around the target. In the second mission, the chaser must still reach the target and obey the speed limit, but it must also loiter in an observation region for some minimum time before approaching. The first mission requires an STL formula with three predicates and three temporal operators, while the second mission requires five predicates and five temporal operators. Figure © IEEE 2022; used with permission.

The MIP approach uses a model-predictive control formulation, and [106] proposes to add counterexamples after solving each instance of the problem. Due to the large size of the resulting MIPs (2800-4500 integer variables for these case studies), we could not find an optimal solution for either satellite problem within 1 hour. Because we were not able to solve even a single instance of the MIP planning problem, we were not able to solve it multiple times to generate any MIP-generated counterexamples. For the comparisons below, we take the best feasible solution found after 500 s for the first mission and 1000 s for the second mission.

We also compare with extensions of the nonlinear optimization method from [96], [104] that include domain randomization with either 32 or 64 samples of ϕ. These methods are similar to those proposed in [66], but we re-implement the method to ensure a fair comparison (our implementation uses just-in-time compilation to speed up gradient computation, resulting in a speed increase over the original implementation). We use the same cost function as in our method, with the same penalty on total impulse.

In the following, we denote our method as CG (counterexample-guided), the nonlinear optimization methods as NLopt (with additional notation for the number of domain randomization examples), and the mixed integer programming method as MIP.

4.4.3 Results

Figs. 4.2 and 4.3 show the results of solving the first and second missions, respectively, with each method. In all cases, we compare the results starting the optimization process from 50 random seeds, reporting the time required and the robustness of the optimized plan under an adversarial disturbance computed via local gradient ascent against the optimal solution. Experiments are run on a laptop computer with 8 GB of RAM and a 1.8 GHz 8-core CPU.

Across both missions, we find that our method consistently yields more robust plans than prior methods. In the first mission, our method finds designs that satisfy the STL specifications in all but 3 trials despite the worst-case adversarial disturbance (an example trajectory is shown in Fig. 4.4a), while the next-best method (NLopt with 64 domain randomization samples) fails on 14 out of 50 trials and takes more than twice as long to run (114.3 s vs. 53.7 s for our method). These results demonstrate the importance of using high-quality counterexamples during optimization: instead of 64 random samples, our method uses 8 initial random samples and 1–4 (median 2) additional counterexamples found using Algorithm 4.3.1. Because the MIP cannot practically be solved multiple times to consider variation in ϕ, the solutions found using the MIP method tend to be less robust; moreover, MIP failed to find a feasible solution for the first mission within 500 s in 16 out of 50 trials.

Our method also performs well on the second mission, satisfying the STL requirement on 46 out of 50 random trials; a representative trajectory is shown in Fig. 4.4b. Despite an additional 500 s of solving time, the MIP method fails to find a feasible solution in 16 out of 50 trials (the MIP encoding requires 2806 binary variables). The second-most-robust

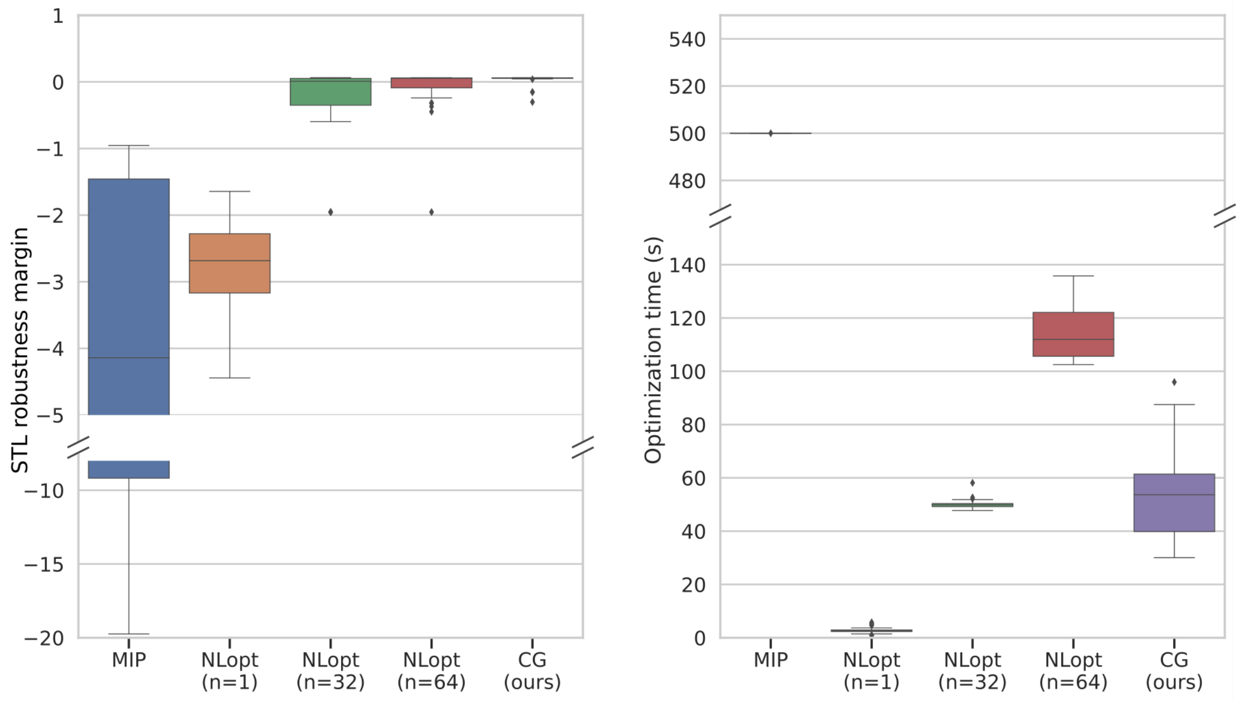


Figure 4.2: Comparison of different STL planning methods on the first example mission, averaged over 50 random seeds. Left: the robustness margin $\rho(\psi_1)$ computed for the optimized design parameters and worst-case exogenous parameters. Right: the planning time required by each method. Our method (CG) achieves much higher robustness than all other methods (satisfying the STL specification despite adversarial perturbations in all but 3 instances) and runs twice as fast as the next-most-robust method. Figure © IEEE 2022; used with permission.

method, after ours, is NLOpt with 64 random examples; this method fails to find a design that satisfies the STL requirement on 17 out of 50 trials despite taking twice as long to run as our method (which required a median of 2 iterations of Algorithm 4.3.1).

4.5 Summary

In this chapter, we describe a counterexample-guided approach to end-to-end robot design optimization with formal behavior specifications. This chapter closes two gaps that were left open in Chapter 3. First, by using STL to directly specify the desired behavior, we avoid the need for manual reward design. Second, we consider robustness to worst-case disturbances at

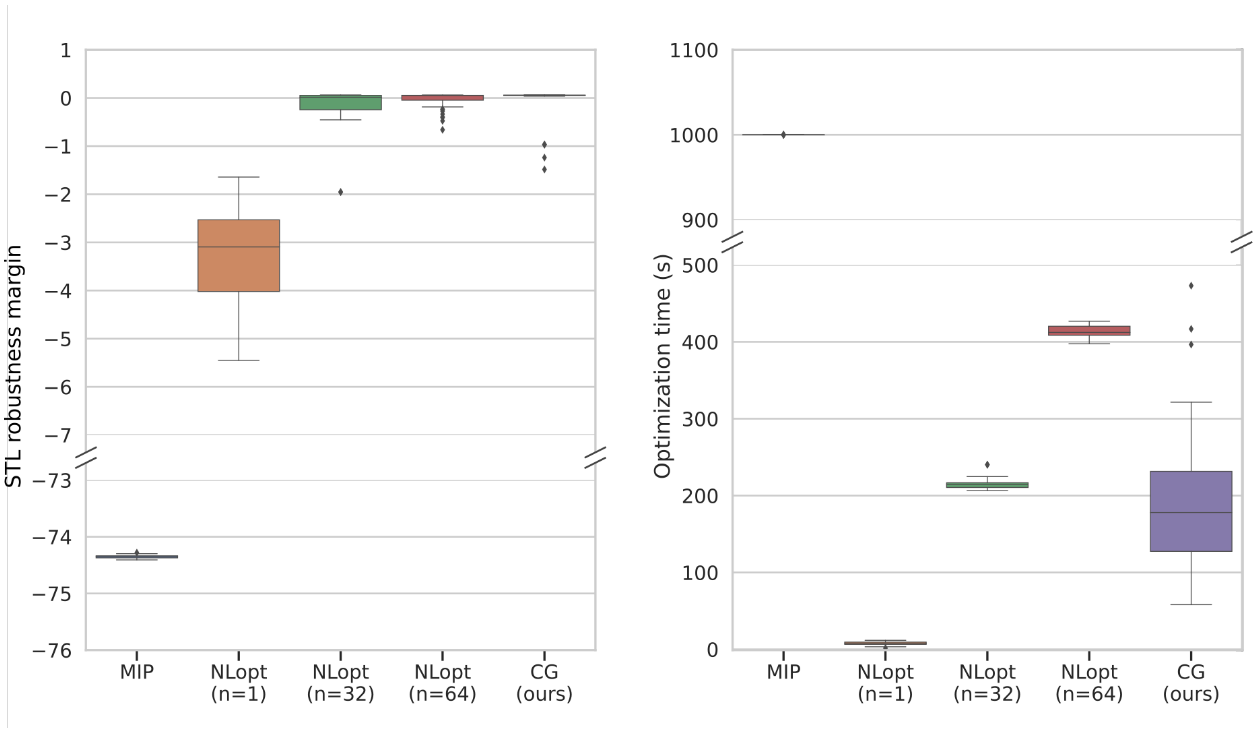
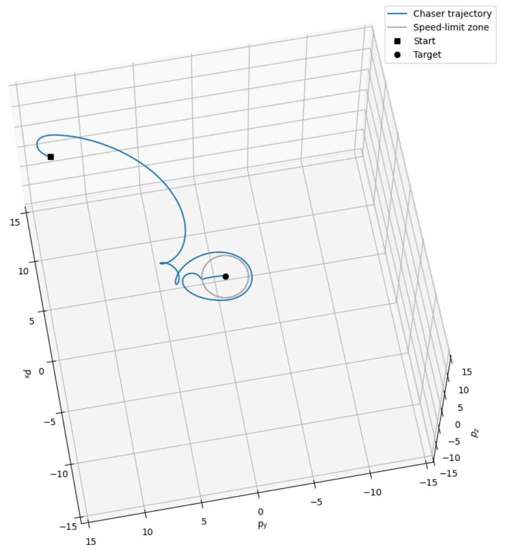


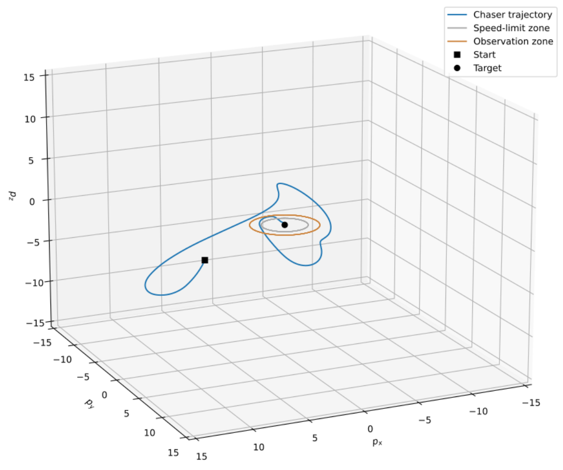
Figure 4.3: Comparison of different STL planning methods on the second example mission, averaged over 50 random seeds. Left: the robustness margin $\rho(\psi_2)$ computed for the optimized design parameters and worst-case exogenous parameters. Right: time required by each method to find a plan. Our method (CG) finds much more robust plans, satisfying the specification in all but 4 instances compared to 17 failures for the next-best method (NLopt with 64 examples). Our method also runs more than twice as fast as the next-most-robust method. Figure © IEEE 2022; used with permission.

design time using counterexample-guided adversarial optimization (rather than the post-hoc statistical method from Chapter 3).

Our empirical results show that our adversarial optimization method yields optimized designs that are more robust than those found by competing methods, despite requiring substantially less computational effort than the next-best method. That said, all methods described in this chapter (aside from intractable MIP-based methods) are inherently local; they rely on local optimization to find both promising designs and adversarial counterexamples. As a result, there is a risk that our method, and any other method that relies on local gradient-based optimization, will get stuck in a local optimum. At first glance, this sub-optimality might not seem like a large issue, since local optimization methods have been



(a) Mission 1.



(b) Mission 2.

Figure 4.4: Optimized trajectories found using our method for the two satellite rendezvous missions. Figures © IEEE 2022; used with permission.

successful in many robotics and learning applications [5]. Unfortunately, for safety-critical applications, a sub-optimal counterexample is potentially worse than no counterexample at all. This is because a sub-optimal counterexample — i.e. an artificially easy test case — can lead to false negatives where we erroneously believe that we have mitigated the worst-case disturbance. We can see this issue manifest in Figs. 4.2 and 4.3 in the few failures that remain even after Algorithm 4.3.1 converges. In these cases, Algorithm 4.3.1 has converged to a local Nash equilibrium and has not correctly identified the true worst-case disturbance.

In the next chapter, we will introduce a novel reformulation of the counterexample-guided optimization problem that addresses this issue. By combining (approximately) global probabilistic inference methods with gradients from differentiable simulation, we can find more challenging counterexamples that will lead to correspondingly more robust designs.

Chapter 5

Gradient-accelerated Inference for Diverse Counterexamples

In Chapters 3 and 4, we introduce local gradient-based optimization methods for design, using standard optimization and adversarial verification-guided optimization. As discussed at the end of Chapter 4, there are a few drawbacks inherent to local methods.

1. **Risk of local minima:** Local gradient-based optimizers are greedy and prone to getting stuck in local minima. This risk is particularly important in safety-critical use cases, since converging to a counterexample that is far from the true worst-case might lead us to falsely believe that a design is safe.

2. **Lack of diversity in adversarial examples:** When verifying a system, it is important to consider a diverse set of failure modes. However, optimization-based approaches have difficulty balancing exploration with exploitation; nearby solutions will typically converge to the nearest local optimum.

3. **Sensitivity to gradient quality:** Many robotic systems, particularly those involving contact or vision, have dynamics that are not differentiable everywhere, as we assumed in previous chapters. Even when these dynamics are smoothed, there are still regions

where the gradients can be poorly conditioned (i.e. arbitrarily large) or flat, which will cause a gradient-based optimizer to diverge or get stuck.

These drawbacks have prompted much discussion in the literature on the limitations of differentiable simulation as a design tool, particularly since certain robotics applications with contact-rich dynamics [58], [60] or vision in the loop [47] can yield poorly conditioned gradients. This has lead to prior work on safety verification for systems with vision in the loop to learn an approximate proxy model for the vision component [12] (where it can be difficult to gather enough training data for rare failure events) or skip the rendering step entirely during backpropagation [33] (which makes it difficult to repair visual-feedback policies).

In this chapter, we address these drawbacks by re-framing the counterexample-guided optimization problem from Chapter 4 as a Bayesian inference problems, which we then solve using gradient-accelerated Markov Chain Monte Carlo (MCMC) methods. We demonstrate empirically how this Bayesian inference framework provides improved performance (in both convergence speed and solution quality) over both gradient-based and gradient-free optimization methods, and we provide a theoretical analysis to support these empirical observations. We also provide a gradient-free variant of our method for use in cases when a differentiable simulator is not available. This chapter is based in part on the author's published work in [111].

The probabilistic approach presented in this chapter builds off of prior work on inference as a verification tool. [11] propose an end-to-end verification tool for autonomous vehicles based on gradient-free adaptive importance sampling. [10] use gradient-free Markov Chain Monte Carlo (MCMC) to find counterexamples. [12] and [13] use gradient-based MCMC to estimate the risk of failure and find counterexamples, respectively. These prior works are focused only on predicting failure modes; they do not consider how to improve the system (e.g. by re-optimizing the controller) to fix these failures once they have been discovered. Although one could run these existing methods repeatedly to update the design parameters, doing so would overlook two important questions. First, since updating the design will cause the distribution of likely failures to shift (invalidating the initial set of predicted failures), it will be necessary to continuously re-predict new failures. Rather than re-running existing methods from scratch, we must ask how much computation can be shared across subsequent failure prediction and design update steps? Second, how can we characterize the equilibrium (if it exists) between the failure prediction and design repair processes? Answering these questions requires both new algorithmic design and theoretical analysis.

The work in this chapter fills this gap in this by developing and analyzing a unified method for failure mode prediction and repair, exploiting the duality between these problems to efficiently search for both a diverse set of counterexamples and updates to the system's policy or design that reduces the severity of those failures.

5.1 From optimization to inference

To motivate the switch from optimization to inference, consider the toy example of optimizing the cost landscape shown on the left in Fig. 5.1a. This cost function has local minima at $x = -0.544$ (marked with a square) and $x = 0.919$ (the true global optimum, marked with a circle). If initialized with $x < 0$, a local gradient-based optimizer will naturally converge to the sub-optimal local minimum at $x = -0.544$, missing the global minimum. We can see this behavior in Fig. 5.1b, which plots the convergence of gradient-based optimization with 5 different random starting positions $x \sim \mathcal{N}(-1.5, 0.1)$. All 5 runs converge to the suboptimal local minimum at $x = -0.544$.

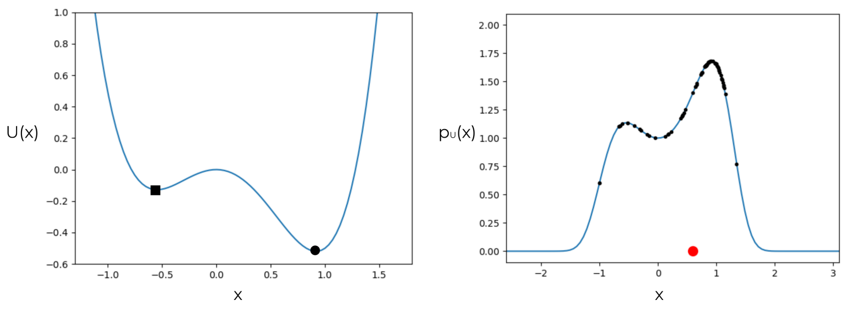
To avoid getting stuck in this local minimum, we can convert this optimization problem into an inference problem; i.e. sampling from the unnormalized probability density:

$$\arg\min_x U(x) \quad \implies \quad x \sim p(x) \propto e^{-U(x)}$$

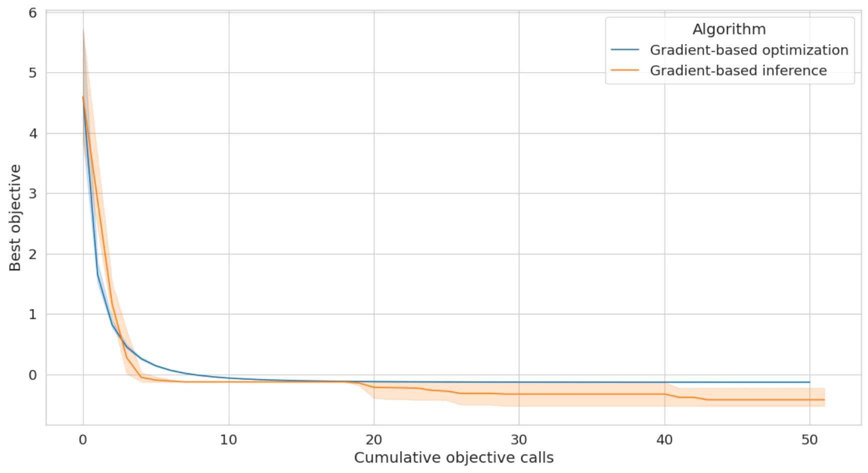
This unnormalized distribution is shown on the right in Fig. 5.1a. We can see that the distribution is bimodal, with a peak at the suboptimal local minimum and a larger peak at the global minimum. By sampling from this distribution, we can find the global minimum on this problem with high probability (this is the so-called *Optimization as Inference* perspective [72], [73]). In practice, we can approximately sample from this distribution using Markov Chain Monte Carlo (MCMC) methods. Like certain stochastic or particle swarm optimization methods, MCMC algorithms balance exploring different regions of the distribution with exploiting high-likelihood areas. Unlike these optimization methods, MCMC offers a number of practical and theoretical benefits that we will discuss shortly (e.g. polynomial time complexity with respect to problem dimension, as opposed to exponential for large families of optimization algorithm [72]). Fig. 5.1b shows the convergence of Metropolis-adjusted Langevin (MALA) MCMC, a gradient-based inference method, on the same cost landscape over 5 different random seeds. MALA is able to find the global minimum with high probability, while gradient-based optimization gets stuck in the local minimum.

A wide variety of MCMC algorithms exist to sample from arbitrary non-normalized probability distributions [112]; common variants include Random-walk Metropolis-Hastings (RMH), Hamiltonian Monte Carlo (HMC), unadjusted Langevin (ULA), and MALA. RMH is gradient-free, while HMC, ULA, and MALA are gradient-based. MALA, which can be seen as a single-step version of HMC, is outlined in Algorithm 5.1.1. Like all MCMC algorithms, it defines a Markov chain with a transition rule designed to preserve the so-called detailed balance condition, which ensures that the stationary distribution of the Markov chain is the same as the target density (see [81], [112] for a more complete introduction). MALA defines a transition rule that begins by proposing a candidate state in line 3 using a gradient-driven drift term and a Gaussian diffusion term. The candidate state is accepted with probability P_{accept} in line 5, which is proportional to the ratio of the target density at the candidate and current states. If the candidate state is accepted, it becomes the new current state; otherwise, the current state is repeated. The algorithm is run for K steps, and the final state is returned as the sample from the target density.

We have shown how transitioning from gradient-based optimization to inference (in par-



(a) Left: a bimodal cost landscape $U(x) = x^4 - 0.5x^3 - x^2$ with a local optimum at $x \approx -0.544$ and a global optimum at $x \approx 0.919$. Right: the corresponding unnormalized likelihood $p(x) \propto e^{-U(x)}$, samples drawn from this distribution (black dots), and the average of 100 samples (red dot).



(b) Convergence of gradient-based optimization and inference (Metropolis-adjusted Langevin) on the double-well cost landscape from Fig. 5.1a, showing how the optimization method gets stuck in a local minimum while the inference method is able to escape and find the global minimum.

Figure 5.1: A simple example demonstrating the use of MCMC on multimodal optimization problems.

ticular, gradient-based inference using MALA) addresses the first drawback identified at the start of this chapter (the risk of getting stuck in local minima). Next, we will discuss how gradient-based inference also addresses the other two drawbacks (the difficulty of sampling diverse failure modes and sensitivity to gradient quality).

5.1.1 Sampling diverse solutions

MCMC sampling methods provide a principled means for balancing exploration and exploitation, and so they are well-suited to problems with multiple modes (particularly when

Algorithm 5.1.1: Metropolis-adjusted Langevin algorithm (MALA, [72], [113])

```
  Input: Initial $x_0$, steps $K$, stepsize $\epsilon$, density $p(x)$.
  Output: A sample drawn from $p(x)$.
1 for $i = 1, \dots, K$ do
2     Sample $\eta \sim \mathcal{N}(0, 2\epsilon I)$                        ▷ Gaussian noise
3     $x_{i+1} \leftarrow x_i + \epsilon \nabla \log p(x_i) + \eta$            ▷ Propose next state
4     $P_{accept} \leftarrow \frac{p(x_{i+1}) e^{-||x_i - x_{i+1} - \epsilon \nabla \log p(x_{i+1})||^2 / (4\epsilon)}}{p(x_i) e^{-||x_{i+1} - x_i - \epsilon \nabla \log p(x_i)||^2 / (4\tau)}}$
5     With probability $1 - \min(1, P_{accept})$:    $x_{i+1} \leftarrow x_i$    ▷ Accept/reject proposal
6 return $x_K$
```

the modes are close together in the search space). However, although MCMC methods enjoy good asymptotic guarantees, including ergodicity and convergence to the target distribution in the infinite-sample limit, any finite-sample implementation will struggle to explore a highly multi-modal target distribution if the modes are well separated (any finite MCMC run will be biased towards the modes near its starting point). A family of algorithms known as sequential Monte Carlo (SMC) algorithms exist to solve this problem by smoothly interpolating between a sequence of distributions, starting from a simple distribution (e.g., a Gaussian) and ending at the target distribution [114]. SMC algorithms are particularly useful for inference problems with multiple modes, since they can be used to sample from the target distribution in a way that is less biased by the initialization. In the following section, we will discuss how SMC can be applied to sample diverse solutions (both failure modes and designs) for practical autonomous system design and verification problems.

5.1.2 Sensitivity to gradient quality

As discussed at the start of this chapter and in prior work [58], [69], many practical robotics problems involve non-smooth dynamics that may not be differentiable, or even continuous, at all points. For example, dynamics involving rigid-body contact are non-smooth at the point when contact is made or broken [58], and many image rendering algorithms are not directly differentiable [47]. Even when these functions can be smoothed, they may still have poorly behaved gradients (e.g. gradients that are either flat or very large). Prior work [58] identifies the ballistic optimization problem shown in Fig. 5.2 as a simple problem that illustrates the effect of both flat gradients and discontinuities.

Fig. 5.3 shows the cost of the best solution found by local optimization using gradient descent, inference using MALA (a gradient-based method), and inference using RMH (a gradient-free method), all starting from a point left of the flat region. As expected, we see that gradient descent gets stuck in the flat region, but both inference methods are able to explore the flat region and eventually discover the global optimum. A natural next question is whether and to what extent gradients help the inference methods explore this distribution; to answer this question, we define a high-dimensional version of the ballistic problem that simply runs N instances in parallel (using an N-dimensional decision vector with one element for each instance) and returns the total cost across all problems. In low dimensions, there is no discernible difference between the gradient-free and gradient-based inference methods. In higher dimensions $N > 10$, gradient-free inference (RMH) converges very slowly, while gradient-based inference (MALA) converges even in $N = 1000$ dimensions.

We attribute this improved performance to three factors. First, the Gaussian diffusion term in the MALA proposal allows it to explore the flat region of the cost landscape and eventually find the optimal solution. Second, the accept/reject step provides some robustness to stiff or inaccurate gradients; if a bad gradient causes MALA to propose a state with much higher cost (lower likelihood), then it will likely reject that proposal and try again. Third, the gradient-based drift term in the proposal provides a valuable heuristic for exploring high-dimensional space, where the exponentially decreased volume of the region around the optimal solution makes it difficult to explore using gradient-free methods.

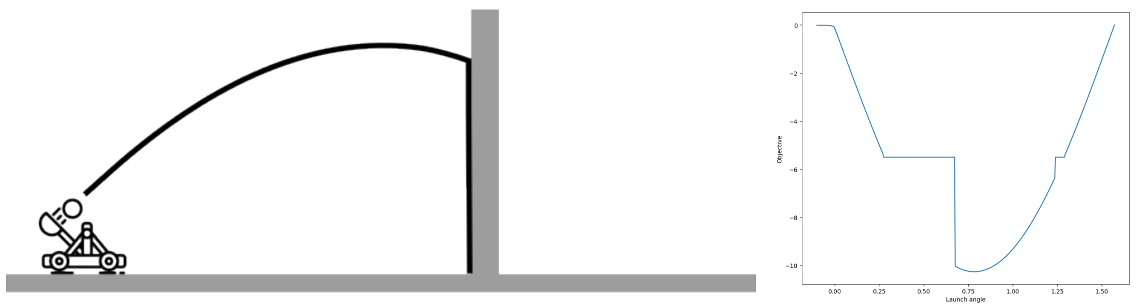
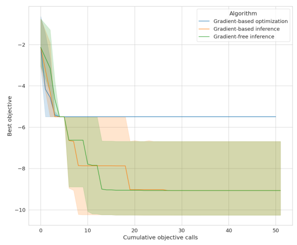
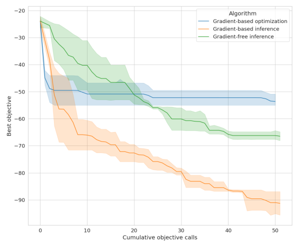


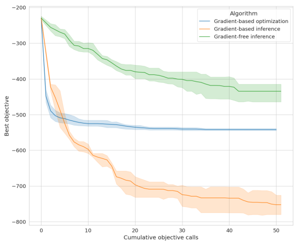
Figure 5.2: Left: the ballistic optimization problem from [58]. Right: the corresponding cost landscape.



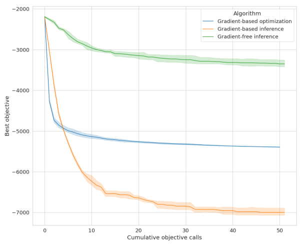
(a) $N = 1$



(b) $N = 10$



(c) $N = 100$



(d) $N = 1000$

Figure 5.3: Convergence of gradient-based optimization (gradient descent), gradient-based inference (MALA), and gradient-free inference (RMH) on the ballistic cost landscape from Fig. 5.1a.

5.2 Sampling and repairing diverse failure modes

Now that we have motivated the switch from optimization to inference, we will reframe the adversarial optimization from Chapter 4 as a sequential inference problem, then demonstrate how we can used gradient-accelerated MCMC sampling (i.e. MALA) to efficiently sample diverse failure modes and find more robust optimized designs.

5.2.1 Preliminaries

Let us define a "failure mode" as a set of environmental parameters ϕ that induce high cost for given policy parameters θ, i.e. finding multiple solutions $\phi^*(\theta) = \text{find}_\phi J(\theta, \phi) \geq J^*$ for failure threshold J^*. In this context, *failure prediction* is the search for such ϕ^* given a design θ, while *failure repair* is the search for updated design parameters θ^* that achieve low costs despite possible variation in ϕ. Note that this is a slightly different framing than used in the adversarial setting in Chapter 4 due to the use of a cost threshold, which is intended to encourage diversity in the predicted failure modes and makes our theoretical analysis later in this chapter more straightforward.

In the following, we will frame the failure prediction and repair in the language of Bayesian inference. We assume that ϕ are distributed according to some prior distribution $p_{\phi,0}$; unlike in Chapter 3, we assume the ability to evaluate and automatically differentiate $\log p_{\phi,0}$. Similarly, we assume knowledge of a prior over design parameters $p_{\theta,0}$, which is used to regularize the design parameters during the repair process.

Given this context, *failure prediction* entails sampling from a pseudo-posterior distribution that balances the prior likelihood of a disturbance with the severity of the induced failure:

$$p_{\text{failure}}(\phi;\theta) \propto p_{\phi,0}(\phi)e^{-[J^*-J(\theta,\phi)]_+} \tag{5.1}$$

where J^* is the cost threshold for a failure event and $[\cdot]_+$ is the exponential linear unit. Intuitively, we can interpret this likelihood as a posterior over environmental parameters ϕ conditioned on a failure occurring [10], [12], [72], [111]. By framing the search for failures prediction as a sampling problem, rather than the traditional adversarial optimization, we gain advantages discussed in the previous section, including the ability to sample diverse failure modes and robustness to poorly conditioned gradients.

This is not the first paper to take a sampling-based approach to failure prediction; for example, [11], [12], and [10] also approach failure prediction using this lens. Our contribution is showing how this sampling framework can be extended to not only predict failures but also repair the underlying policy, thus mitigating the impact of the failure. Given initial design parameters θ_0 and a population of anticipated failure modes $\phi_1,\ldots,\phi_n$, we can increase the robustness of our policy by sampling from a corresponding repair pseudo-posterior, similar to Eq. (5.1),

$$p_{\text{repair}}(\theta;\phi_1,\ldots,\phi_n) \propto p_{\theta,0}(\theta;\theta_0)e^{-\sum_{\phi_i}[J(\theta,\phi_i)-J^*]_+/n} \tag{5.2}$$

where the prior likelihood $p_{\theta,0}$ regularizes the search for repaired policies that are close to the original policy.

Intuitively, this distribution of repaired policies can be seen as a posterior over policies conditioned on the event that a failure *does not* occur in the given scenarios. Sampling from this posterior can be seen as a form of regularized re-training on the set of predicted failures, since maximizing the log of (5.2) is equivalent to minimizing the empirical risk $\sum_{\phi_i}[J(\theta, \phi_i) - J^*]_+/n$ with regularization $||\theta - \theta_0||_2^2$ (assuming a Gaussian prior). This connection helps motivate our use of (5.2), but we find empirically in Section 5.3 that the increased diversity from sampling rather than straightforward gradient optimization yields better solutions in practice.

5.2.2 Approach

Previous works have shown that sampling from a failure distribution like Eq. (5.1) can generate novel failures [10], [12], [13], but several challenges have prevented these works from considering end-to-end policy repair as well. Our main contribution is a framework for resolving these challenges and enabling simultaneous failure prediction and repair, which we call *RADIUM* (Robustness via Adversarial Diversity using MCMC, illustrated in Fig. 5.4). We have designed this framework to take advantage of problem structure (e.g. differentiability) when possible, but we provide the ability to swap gradient-based subsolvers for gradient-free ones when needed, and we include a discussion of the associated trade-offs.

Challenge 1: Distribution shift during retraining Previous methods have proposed generating failure examples for use in retraining, but there is an inherent risk of distribution shift when doing so. Once we repair the policy, previously-predicted failures become stale and are no longer useful for verification (i.e. the distribution of likely failures has shifted). In the worst case, this can lead to overconfidence if we claim to have repaired all previously-discovered failures while remaining vulnerable to other failures. To address this issue, we

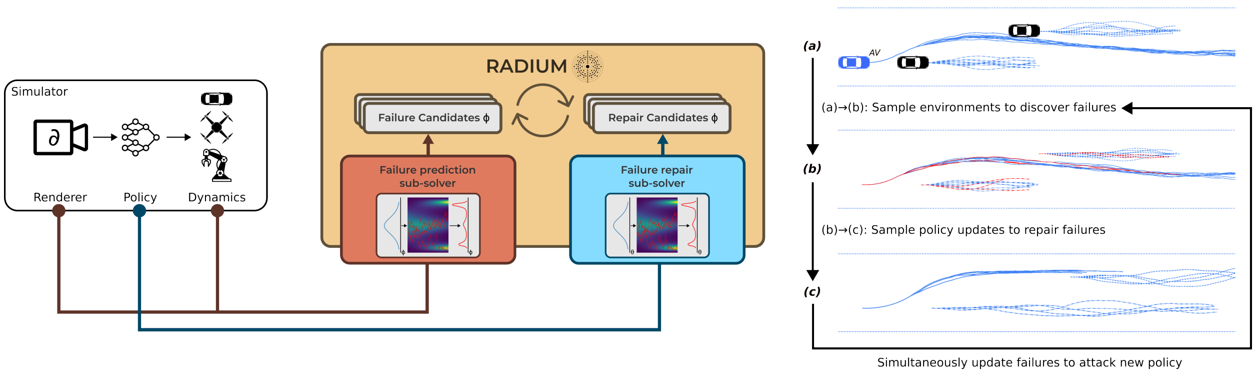

Simulator
Renderer
Policy
Dynamics
RADIUM
Failure Candidates ϕ
Repair Candidates ϕ
Failure prediction sub-solver
Failure repair sub-solver
(a)
AV
(a)→(b): Sample environments to discover failures
(b)
(b)→(c): Sample policy updates to repair failures
(c)
Simultaneously update failures to attack new policy

Figure 5.4: An overview of our approach for closed-loop rare-event prediction, which efficiently predicts and repairs failures in autonomous systems. Our framework alternates between failure prediction and repair sub-solvers, which use a simulated environment to efficiently sample from the distributions (5.1) and (5.2). We use differentiable rendering and simulation to accelerate our method with end-to-end gradients, but we also propose a gradient-free implementation.

interleave failure and repair steps to continuously update the set of predicted failures as we repair the policy, creating an adversarial sampling process that generates a robust repaired policy along with a set of salient failure cases.

Challenge 2: Exploring diverse failure modes Traditional methods like Markov chain Monte Carlo (MCMC) are able to sample from non-normalized likelihoods like (5.1) and (5.2), but they struggle to fully explore the search space when the likelihood is highly multi-modal. To mitigate this issue, we take inspiration from the recent success of diffusion processes [28], [115] and sequential Monte Carlo algorithms [116] that interpolate between an easy-to-sample prior distribution and a multi-modal target distribution. Instead of sampling directly from the posterior, we begin by sampling from the unimodal, easy-to-sample prior and then smoothly interpolate to the posterior distributions (5.1)-(5.2). This process yields the tempered likelihood functions:

$$\tilde{p}_{\text{failure}} \propto p_{\phi,0}(\phi) e^{-\tau[J^* - J(\theta,\phi)]_+} \tag{5.3}$$

$$\tilde{p}_{\text{repair}} \propto p_{\theta,0}(\theta,\theta_0) e^{-\frac{\tau}{n}\sum_{\phi_i}[J(\theta,\phi_i) - J^*]_+} \tag{5.4}$$

where the tempering parameter τ is smoothly varied from 0 to 1. When $\tau = 0$, this is equivalent to sampling from the prior distributions, and when $\tau \to 1$ we recover the full posteriors (5.1)-(5.2). This tempering process reduces the risk of overfitting to one particular mode of the failure distribution and encourages even exploration of the failure space.

Challenge 3: Efficiently sampling in high dimension Previous works have proposed a wide variety of sampling algorithms that might be used as sub-solvers in our framework, including MCMC methods like random-walk Metropolis-Hastings (RMH; [117]), Hamiltonian Monte Carlo (HMC; [118]), and the Metropolis-adjusted Langevin algorithm (MALA; [119]), variational inference methods like Stein Variational Gradient Descent (SVGD; [120]), and other black-box methods like adaptive importance sampling [11]. RADIUM is able to use any of these sampling methods as sub-solvers for either prediction or repair. Generally, these sampling methods can be classified as either gradient-free or gradient-based. Theoretical and empirical evidence suggests that gradient-based methods can enjoy faster mixing time in high dimensions on certain classes of sufficiently smooth non-convex problems [72], but autonomous systems with visual feedback have historically been treated as black-boxes due to an inability to backpropagate through the rendering step [10]–[12]. To enable the use of gradient-based samplers in RADIUM, we draw upon recent advances in differentiable simulation and rendering [49], [50] provide end-to-end gradients. In Sections 5.2.4 and 5.3, we provide theoretical and empirical evidence of a performance advantage for gradient-based samplers, but in order to make RADIUM compatible with existing non-differentiable simulators we also conduct experiments using gradient-free sampling subroutines.

5.2.3 RADIUM

Pseudocode for RADIUM is provided in Algorithm 5.2.1. The algorithm maintains a population of candidate repaired policies $[\theta_1, \dots, \theta_n]$ and failures $[\phi_1, \dots, \phi_n]$ that are updated over N sampling rounds. In each round, we sample a set of new candidate policies from the repair pseudo-posterior (5.4), then sample a new set of failures that attack the current population of policies. In practice, we average the tempered failure log probability (5.3) over the population of candidate designs, which results in a smoother distribution.

RADIUM supports a wide range of subroutines for sampling candidate failures and repaired policies. In our experiments, we include RMH and MALA (gradient-free and gradient-based MCMC algorithms, respectively). We choose these particular methods to provide a direct comparison between similar algorithms with and without gradients, as MALA reduces to RMH when the gradient is zero. Pseudocode for MALA is included in Algorithm 5.1.1.

A final practical consideration is that although the stochasticity in our sampling-based approach can help us explore the design and failure spaces, we incur a degree of sub-optimality as a result. When using gradient-based sampling, we have the option to reduce this sub-optimality by "quenching" the solution: switching to simple gradient descent (e.g. using MALA for the first 90 rounds and then gradient descent on the last 10 rounds). In practice, we find that quenching can noticeably improve the final cost without compromising the diversity of predicted failure modes. Quenching is particularly important on problems where the cost function includes penalty terms for constraint satisfaction, where a few rounds of gradient descent can help to satisfy these constraints.

Algorithm 5.2.1: RADIUM: Robustness via Adversarial Diversity Using MCMC

```
Input: $N$ rounds, $K$ steps per round, stepsize $\lambda$, population size $n$, tempering rate
       $\alpha$, sampling algorithm (e.g. MALA as in Alg. 5.1.1)
1 Sample initial failures and policies using priors:
    $[\phi_1, \dots, \phi_n]_0 \overset{\text{iid}}{\sim} p_{\phi,0}$,  $[\theta_1, \dots, \theta_n]_0 \overset{\text{iid}}{\sim} p_{\theta,0}$;
2 for $i = 1, \dots, N$ do
3     $\tau \leftarrow 1 - e^{-\alpha i/N}$;                                  // Tempering schedule
4     Sample $[\theta_1, \dots, \theta_n]_i \overset{\text{iid}}{\sim}$ (5.4);                    // Sample repaired policies
5     Sample $[\phi_1, \dots, \phi_n]_i \overset{\text{iid}}{\sim}$ (5.3);             // Generate failures attacking $\theta_i^*$
6 end
Return: Repaired policy $\theta_N^* = \arg\max_i$ (5.4) and failures $[\phi_1, \dots, \phi_n]_N$ attacking
        that policy.
```

5.2.4 Theoretical analysis

The iterative adversarial sampling process defined in Alg. 5.2.1 raises a few theoretical questions. First, when can we expect the individual sampling steps on lines 4 and 5 to converge, and under what conditions might we expect a gradient-based sampling sub-routine to converge faster than a gradient-free one? Second, assuming that these individual samplers converge, what sort of policies will result from the adversarial sampling loop in Alg. 5.2.1?

Convergence and gradient acceleration RADIUM inherits the asymptotic convergence guarantees of the particular subsolvers used for each sampling step. For example, when using an MCMC sampler, so long as that sampler can propose arbitrarily large steps with non-zero probability and satisfies the detailed balance condition (e.g. through the use of a Metropolis adjustment), then the sampler will produce samples asymptotically close to the target sampling distribution. Since the conditions for asymptotic convergence of MCMC samplers are relatively weak [117], it is more interesting to ask about finite-sample convergence rates; in particular, under what conditions can we expect gradient-based samplers like MALA to accelerate convergence to the target distribution?

In many robotics problems, even when analytical gradients are available, it is unclear whether these gradients are useful for optimization (i.e. low empirical bias and variance; [58]). Here, we build on recent theoretical results by [72] to provide sufficient conditions for fast, polynomial-time convergence of gradient-based samplers in our setting. With slight abuse of notation, we use $J(\theta, \phi)$ to denote the composition of the simulator and cost function.

Theorem 5.2.1. *Let $J(\theta, \phi)$ be a L-Lipschitz smooth cost function (i.e. ∇J is L-Lipschitz continuous), let the log prior distributions $\log p_{\phi,0}$ and $\log p_{\theta,0}$ be Lipschitz smooth everywhere and m-strongly convex outside a ball of finite radius R, and let $d = \max(\dim\theta, \dim\phi)$ be the dimension of the search space. If $m > L$, then MALA with appropriate step size will yield samples within ϵ total variation distance of the target distributions (5.3) and (5.4) with total number of sampling steps $\leq \widetilde{\mathcal{O}}\left(d^2 \ln \frac{1}{\epsilon}\right)$.*

The key idea of the proof is to rely on the log-concavity of the prior distributions to dominate the non-convexity of the cost function sufficiently far from the central modes.

Proof. We will show the proof for sampling from the failure generating process with likelihood given by Eq. (5.3); the proof for the repair generating process follows similarly. The log-likelihood for the failure generating process is

$$\log p_{\phi,0}(\phi) - \tau[J^* - J(\theta, \phi)]_+ \tag{5.5}$$

[72] show that MALA sampling enjoys the convergence guarantees of Theorem 5.2.1 so long as the target log likelihood is strongly convex outside of a ball of finite radius R (see Theorem 1 in [72]). Since $\log p_{\phi,0}(\phi)$ is assumed to be strongly m-convex, it is sufficient to show that as $||\phi|| \to \infty$, the strong convexity of the log-prior dominates the non-convexity in $\tau[J^* - J(\theta, \phi)]_+$.

For convenience, denote $f(\phi) = -\tau[J^* - J(\theta, \phi)]_+$ and $g(\phi) = \log p_{\phi,0}(\phi)$. We must first show that $f(\phi) + g(\phi)$ is $(m - L)$-strongly convex, for which it suffices to show that $f(\phi) + g(\phi) - (m - L)/2||\phi||^2$ is convex. Note that

$$f(\phi) + g(\phi) - \frac{m - L}{2}||\phi||^2 = f(\phi) + \frac{L}{2}||\phi||^2 + g(\phi) - \frac{m}{2}||\phi||^2 \tag{5.6}$$

$g(\phi) - \frac{m}{2}||\phi||^2$ is convex by m-strong convexity of g, so we must show that the remaining term, $f(\phi) + L/2||\phi||^2$, is convex. Note that the Hessian of this term is $\nabla^2 f(\phi) + LI$. Since we have assumed that J is L-Lipschitz smooth (i.e. its gradients are L-Lipschitz continuous), it follows that the magnitudes of the eigenvalues of $\nabla^2 f$ are bounded by L, which is sufficient for $\nabla^2 f(\phi) + LI$ to be positive semi-definite, completing the proof. $\square$

Theorem 5.2.1 requires smoothness assumptions on the cost; we recognize that this assumption is difficult to verify in practice and does not hold in certain domains (notably when rigid-body contact is involved). However, in the problems we consider it is possible to smooth the simulator (e.g. using smoothed dynamics and a blurred renderer and scene representation), thus smoothing the gradients of J. The smoothness and convexity conditions hold for many common prior distributions, such as Gaussian and smoothed uniform distributions.

Adversarial Joint Distribution Even if the samplers for both policy and environmental parameters converge within each round of Alg. 5.2.1, it is not clear what will be the effect of running these samplers repeatedly in an adversarial manner. Our next theoretical result defines the joint distribution of θ and ϕ as a result of this adversarial sampling loop. To simplify the theoretical analysis, we consider the case when population size $n = 1$, and we replace the smooth ELU with a ReLU in (5.1) and (5.2).

Theorem 5.2.2. *The iterative adversarial sampling procedure in Alg. 5.2.1 yields policies drawn from a marginal distribution with unnormalized density function*

$$f_\theta(\theta^*) = p_{\theta,0}(\theta^*) \left(\frac{\mathbb{E}_{\phi\sim p_{\phi,0}}\left[e^{J(\theta^*,\phi)-J^*} | J(\theta^*,\phi) \leq J^*\right]}{\mathbb{E}_{\phi\sim p_{\phi,0}}\left[e^{J(\theta^*,\phi)-J^*}\right]} + \frac{\mathbb{P}[J(\theta^*,\phi) > J^*]}{\mathbb{E}_{\phi\sim p_{\phi,0}}\left[e^{J(\theta^*,\phi)-J^*}\right]} \right) \tag{5.7}$$

where $\mathbb{P}(J(\theta^*,\phi) > J^* = \mathbb{E}_{\phi\sim p_{\phi,0}}[\mathbb{1}(J(\theta^*,\phi) \geq J^*)]$ is the probability of failure when ϕ is sampled from the prior distribution.

The first term in the parenthesis in (5.7) is bounded above by 1 and maximized when the policy does not experience failure (in which case the conditional and unconditional expectations will be equal). The numerator of the second term is bounded on $[0, 1]$, while the denominator grows exponentially large when a failure occurs, assigning higher probability (relative to the prior) for policies that avoid failure.

Proof. We can treat Alg. 5.2.1 as a two-stage Gibbs sampling procedure and apply the Hammersley-Clifford Theorem [121] to get the joint distribution

$$f_{\theta,\phi}(\theta^*,\phi^*) = p_{\theta,0}(\theta^*) p_{\phi,0}(\phi^*) \frac{e^{-[J^*-J(\theta^*,\phi^*)]_+}}{\mathbb{E}_{\phi\sim p_{\phi,0}}\left[e^{J(\theta^*,\phi)-J^*}\right]} \tag{5.8}$$

Integrating over ϕ yields the marginal distribution of θ, completing the proof.

$$f_{\theta^*} = \int_\phi f_{\theta,\phi}(\theta^*, \phi) d\phi = \frac{p_{\theta,0}(\theta^*)}{\mathbb{E}_{\phi\sim p_{\phi,0}}\left[e^{J(\theta^*,\phi)-J^*}\right]} \int_\phi p_{\phi,0}(\phi) e^{-[J^*-J(\theta^*,\phi^*)]_+} d\phi \tag{5.9}$$

$$= p_{\theta,0}(\theta^*) \frac{\mathbb{E}_{\phi\sim p_{\phi,0}}\left[e^{-[J^*-J(\theta^*,\phi)]_+}\right]}{\mathbb{E}_{\phi\sim p_{\phi,0}}\left[e^{J(\theta^*,\phi)-J^*}\right]} \tag{5.10}$$

$$= p_{\theta,0}(\theta^*) \frac{\mathbb{E}_{\phi\sim p_{\phi,0}}\left[e^{-(J^*-J(\theta^*,\phi))} | J^* - J(\theta^*,\phi) \geq 0\right] + \mathbb{E}_{\phi\sim p_{\phi,0}}\left[1 | J^* - J(\theta^*,\phi) < 0\right]}{\mathbb{E}_{\phi\sim p_{\phi,0}}\left[e^{J(\theta^*,\phi)-J^*}\right]} \tag{5.11}$$

$$= p_{\theta,0}(\theta^*) \frac{\mathbb{E}_{\phi\sim p_{\phi,0}}\left[e^{-(J^*-J(\theta^*,\phi))} | J^* \geq J(\theta^*,\phi)\right] + \mathbb{E}_{\phi\sim p_{\phi,0}}\left[1 | J^* < J(\theta^*,\phi)\right]}{\mathbb{E}_{\phi\sim p_{\phi,0}}\left[e^{J(\theta^*,\phi)-J^*}\right]} \tag{5.12}$$

$$= p_{\theta,0}(\theta^*) \frac{\mathbb{E}_{\phi\sim p_{\phi,0}}\left[e^{-(J^*-J(\theta^*,\phi))} | J^* \geq J(\theta^*,\phi)\right] + \mathbb{P}[J(\theta^*,\phi) > J^*]}{\mathbb{E}_{\phi\sim p_{\phi,0}}\left[e^{J(\theta^*,\phi)-J^*}\right]} \tag{5.13}$$

$$\tag{5.14}$$

□

5.3 Simulation experiments

In this section, we provide empirical comparisons of RADIUM with existing methods for adversarial optimization and policy repair in a range of simulated environments. We have two main goals in this section. First, we seek to understand whether re-framing the failure repair problem from optimization to inference leads to better solutions (i.e. more robust designs and better coverage by the predicted failures). Second, we study whether the gradient-based version of our method yields any benefits over the gradient-free version. After the empirical comparisons in this section, Section 5.4 then provides three case studies with a more in-depth discussion of how RADIUM can be applied to practical verification and design problems, including two case studies demonstrating how repaired designs can be transferred from simulation to hardware.

We conduct simulation studies on a range of problems from the robotics and cyberphysical systems literature, comparing against previously-published adversarial optimization methods. More detail on each benchmark, baseline, and implementation is provided in the appendix.

5.3.1 Experimental setup

Baselines

We compare with three baselines taken from the adversarial optimization and testing literature. *Gradient descent with randomized counterexamples (GD_r)* optimizes the design using a fixed set of random counterexamples, representing a generic policy optimization with domain randomization approach. *Gradient descent with adversarial counterexamples (GD_a)* alternates between optimizing the design and optimizing for maximally adversarial failure modes, as in [33], [93]. *Learning to collide (L2C)* uses black-box optimization (REINFORCE) to search for failure cases [9]. We denote the gradient-free and gradient-based variants of RADIUM as R_0 and R_1, respectively. All methods were run on the same GPU model with the same number of rounds for each task. Hyperparameters for all experiments are given in the appendix.

Benchmark problems

We rely on two classes of benchmark problem in this work: three problems without vision in the loop, and four problems with vision in the loop. Some of these problem domains include multiple environments of varying complexity, for a total of 13 different environments. A summary of these environments is given in Fig 5.5. More details on the parameters and cost functions used for each environment are given in the appendix.

Non-vision benchmarks *Search:* a set of seeker robots must cover a region to detect a set of hiders. θ and ϕ define trajectories for the seekers and hiders, respectively. Failure occurs if any hider escapes detection by the seekers (which have fixed sensing radius). This

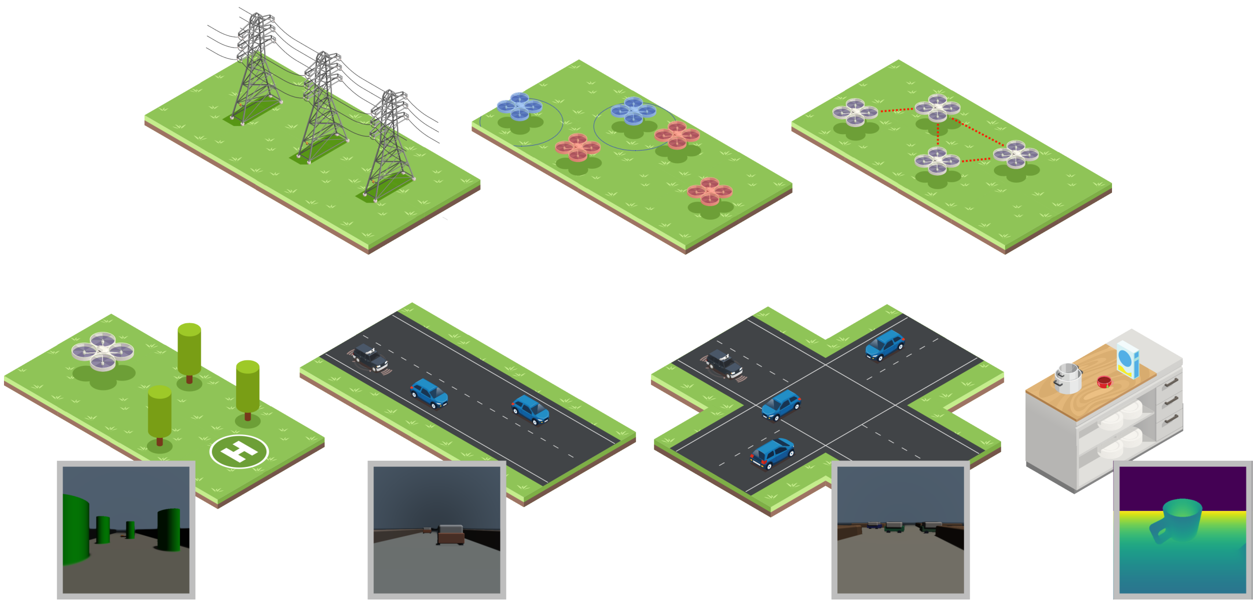


Figure 5.5: The different environments used in our simulation studies, including 3 domains without visual feedback (power grid, search, and formation control) and 4 domains with vision in the loop (drone, AV highway, AV intersection, and grasping). The inset shows the robot's perspective in the vision-in-the-loop tasks.

environment has two variants: small (6 seeker vs. 10 hider, $\dim\theta = 60$, $\dim\phi = 100$) and large (12 seeker vs. 20 hider, $\dim\theta = 120$, $\dim\phi = 200$). *Formation control:* a swarm of drones fly to a goal while maintaining full connectivity with a limited communication radius. θ defines trajectories for each robot in the swarm, while ϕ defines an uncertain wind velocity field. Failure occurs when the second eigenvalue of the graph Laplacian is close to zero. This environment has small (5 agent, $\dim\theta = 30$, $\dim\phi = 4$) and large (10 agent, $\dim\theta = 100$, $\dim\phi = 4$) variants. *Power grid dispatch:* electric generators must be scheduled to ensure that the network satisfies voltage and maximum power constraints in the event of transmission line outages. θ specifies generator setpoints and ϕ specifies line admittances; failures occur when any of the voltage or power constraints are violated. This environment has small (14-bus, $\dim\theta = 32$, $\dim\phi = 20$) and large (57-bus, $\dim\theta = 98$, $\dim\phi = 80$) versions.

Vision-in-the-loop benchmarks *AV (highway):* An autonomous vehicle must overtake two other vehicles. *AV (intersection):* the autonomous vehicle must navigate an uncontrolled intersection with crossing traffic. In both AV tasks, the actions of the non-ego vehicles are uncertain, and the AV observes RGBd images from a front-facing camera as well as its own speed. *Drone:* A drone must safely navigate through a cluttered environment in windy conditions. There is uncertainty in the wind speed and location of all obstacles. Initial convolutional neural network (CNN) policies θ_0 for drone and intersection environments were pre-trained using behavior cloning, and CNN-based policies for the highway environment were pre-trained using PPO [122]. *Grasp (box/mug):* a robot must locate and grasp an object using a depth image of the scene. There is uncertainty in the location of the objects and in the location of a nearby distractor object. The grasp detector is trained with labels of ground-truth grasp affordances.

The dimension of the failure space is 20 for the highway task, 30 for the intersection task, 12 for the drone task, and 4 for the grasping tasks. The dimension of the policy space is 64k for the highway and intersection tasks, 84k for the drone task, and 266k for the grasping tasks.

Implementation

Since we require a differentiable renderer and simulation engine for our work, we were not able to use an off-the-shelf simulator like CARLA [76]. Instead, we write our own simulator and basic differentiable renderer using JAX, which is available at github.com/MIT-REALM/architect_corl_23. Likewise, our method and all baselines were implemented in JAX and compiled using JAX's just-in-time (JIT) compilation. Each metric reports the mean and standard deviation across four independent random seeds. All methods are given the same total sample budget for both prediction and repair (except for GD_r, which does not update the predicted failure modes).

The non-vision benchmarks were all initialized with random θ_0, and the vision benchmarks were initialized using θ_0 trained using reinforcement learning or behavior cloning with domain randomization. We include comparisons with GD_r, GD_a, and *L2C* on all non-vision benchmarks. Since θ_0 on the vision benchmarks was trained using domain randomization, GD_r is not able to improve the initial parameters, and so we include comparisons with θ_0, GD_a, and *L2C* for the vision benchmarks.

Metrics

To measure the robustness of the optimized policies, we report the failure rate (FR) on a test set of 1,000 i.i.d. samples of ϕ from the prior $p_{\phi,0}$. We also report the mean cost on this test set as well as the maximum cost on the vision-in-the-loop benchmarks (where cost is bounded by construction) and the 99^{th}-percentile cost for non-vision benchmarks (some of which have unbounded cost, making the 99^{th} percentile more representative). Costs are normalized by the maximum cost observed across any method. Finally, for each task, we report the time required to run a simulation both with and without reverse-mode differentiation.

5.3.2 Results

Fig. 5.6 shows the results from benchmark problems without vision in the loop, while Fig. 5.7 shows the results from problems with vision in the loop. For ease of comparison, each plot groups the gradient-free methods (*L2C* and R_0) and the gradient-based methods (GD_r, GD_a, and R_1). Since the initial parameters for the vision-in-the-loop benchmarks in Fig. 5.7 were trained using RL or behavior cloning (which do not require differentiable simulation), we group θ_0 with the gradient-free results in Fig. 5.7. We also compare the convergence rates of each method in Fig. 5.8. Table 5.1 shows the time required for simulating with and without automatic differentiation.

Fig. 5.9 shows examples of failure cases and repaired policies generated using R_1 on three vision-in-the-loop tasks: AV (highway), AV (intersection), and drone. The left of Fig. 5.9 shows the initial policy θ_0 and failure modes discovered using our method (sampling

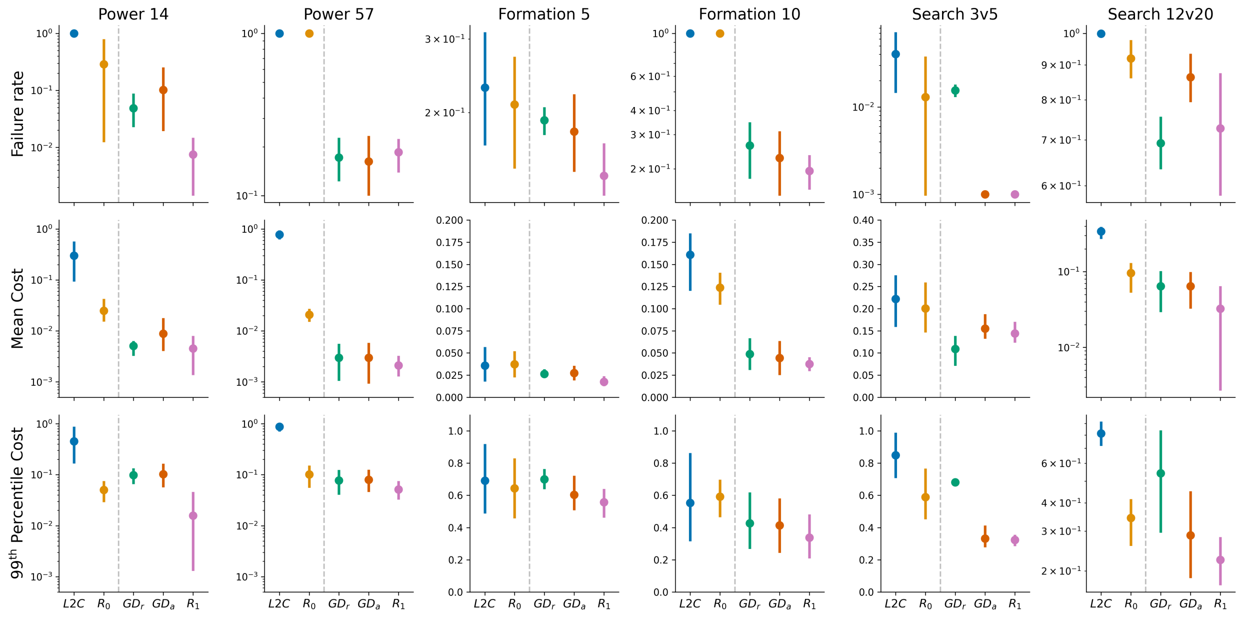


Figure 5.6: Comparison of our method (gradient-free and gradient-based variants R_0 and R_1, respectively) and baseline methods on benchmark problems without vision in the loop, showing failure rate, mean cost, and 99th percentile cost on a test set of 1,000 randomly sampled ϕ. The dashed gray lines separate gradient-free and gradient-based methods.

ϕ while holding θ_0 fixed), while the right shows the repaired policy and updated challenging counterexamples. Since the distribution of failure modes shifts as we repair the policy, we continuously re-sample the failure modes to be relevant to the updated policy. In all cases, we see that the repaired policy found using our method experiences fewer failures, despite the updated adversarial failure modes. In certain cases, the repaired policy exhibits a qualitatively different behavior; for example, in the vision-in-the-loop highway control task, the repaired policy is less aggressive than the original policy, avoiding the risky overtake maneuver (top row of Fig. 5.9).

5.3.3 Discussion

In our results on problems without vision in the loop (Fig. 5.6), we see several high-level trends. First, we see that gradient-based techniques (GD_r, GD_a, and R_1) achieve lower failure rates, mean cost, and 99th percentile costs relative to gradient-free methods ($L2C$ and R_0)

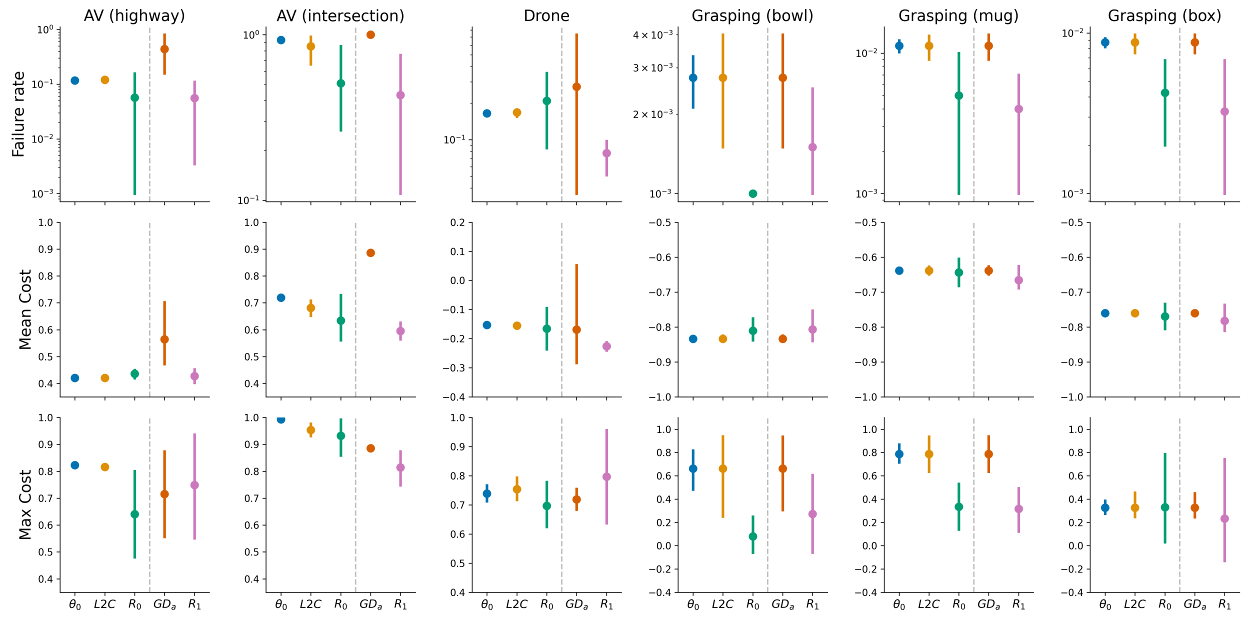
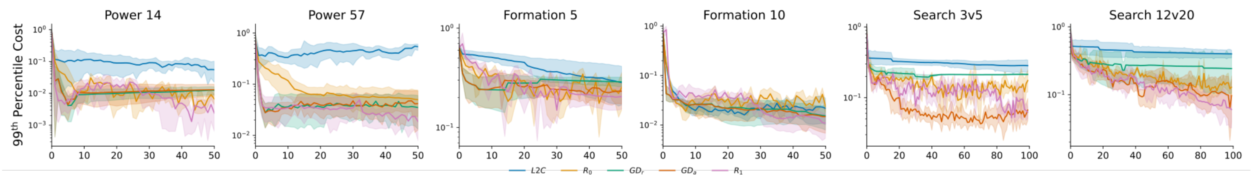


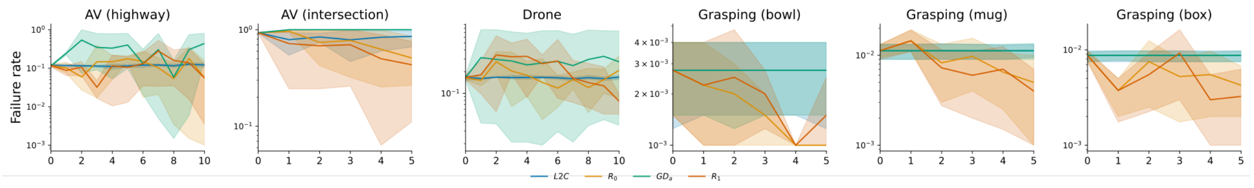
Figure 5.7: Comparison of our method (gradient-free and gradient-based variants R_0 and R_1, respectively) and baseline methods on benchmark problems with vision in the loop, showing failure rate, mean cost, and max cost on a test set of 1,000 randomly sampled ϕ. The dashed gray lines separate gradient-free and gradient-based methods.

on the test set, likely because gradient information helps the former methods explore the high-dimensional search space (as seen in the faster convergence of gradient-based methods in Fig. 5.8a). Moreover, we find that our methods (R_0 and R_1) outperform other methods within each of their respective categories; i.e. R_0 yields repaired solutions with lower costs and failure rates than *L2C*, and R_1 likewise outperforms GD_r and GD_a.

We see a slightly different pattern in our results for problems with vision in the loop. On these problems, we find that existing gradient-based methods like GD_a do not achieve lower failure rates than gradient-free methods like *L2C*, possibly due to poor gradient quality from the differentiable renderer (where occlusions can lead to large variance in the automatically-derived gradients). In contrast, both variants of our method achieve low failure rates for repaired policies in the vision-in-the-loop tasks, and R_1 in particular is able to achieve better performance on some tasks because the Metropolis-Hastings adjustment on line 5 of Algorithm 5.1.1 allows it to reject large steps caused by poorly conditioned gradients.



(a) Non-vision benchmarks



(b) Vision-in-the-loop benchmarks

Figure 5.8: Convergence rates of our method and baselines on tasks with (top) and without (bottom) vision in the loop.

Table 5.1: Time required for simulating a rollout with and without autodiff (AD) for each task (in seconds). Average and standard deviation (subscript) reported across 100 trials on AMD Ryzen Threadripper 3990X 64-Core Processor (non-vision tasks) and an NVIDIA RTX A4000 (vision-in-the-loop tasks).

	Non-vision tasks					
	Power (14)	Power (57)	Formation (5)	Formation (10)	Search (3v5)	Search (12v20)
w/o AD	$0.00122_{0.00413}$	$0.0107_{0.00893}$	$0.0326_{0.0173}$	$0.628_{0.296}$	$0.00147_{0.00461}$	$0.00461_{0.00747}$
w/ AD	$0.00165_{0.00488}$	$0.0136_{0.0111}$	$0.0543_{0.0212}$	$0.714_{0.306}$	$0.00358_{0.00704}$	$0.0107_{0.00851}$
	Vision-in-the-loop					
	AV (hw.)	AV (int.)	Drone	Grasp (all)		
w/o AD	$0.70_{0.003}$	$2.22_{0.01}$	$0.39_{0.002}$	$0.0045_{5.1\times10^{-5}}$		
w/ AD	$1.72_{0.003}$	$6.65_{0.14}$	$1.77_{0.06}$	$0.0049_{3.8\times10^{-5}}$		

5.4 Case studies

In this next section, we present three case studies to illustrate the practical use of our method. We first demonstrate how RADIUM can be used to solve a challenging optimization problem arising in the control of electrical power systems. We then provide two case studies showing how RADIUM can be applied to robotics problems and transferred to hardware, including one case study demonstrating sim2real transfer of vision-in-the-loop robot control policies.

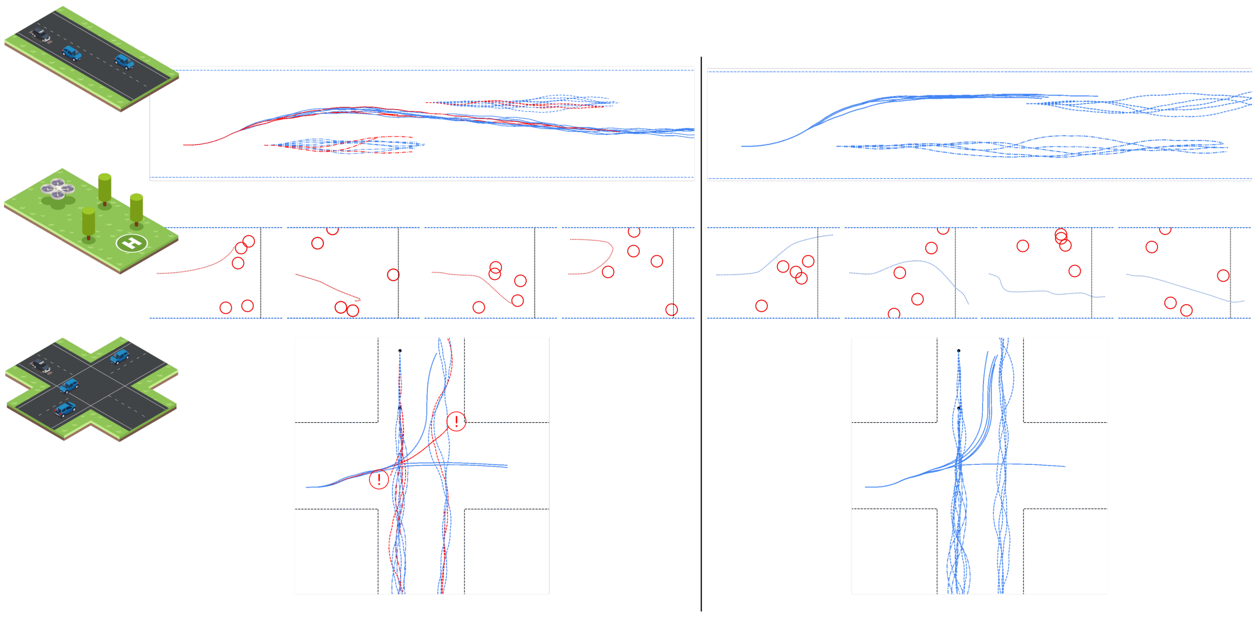


Figure 5.9: Examples of failure cases (left) and repaired policies (right) generated using our method. Failed trajectories are shown in red.

5.4.1 Robust generation dispatch for secure power networks

For our first case study, we consider the problem of controlling an electric power grid subject to failures in transmission lines. Two simple networks (the IEEE 14- and 57-node test system) are shown in Fig. 5.10. The goal in this problem is to find control inputs (power injection and voltage at each generator, and power demand at each load) that ensure that the voltage seen by each load is stable, even in the event of transmission outages (which we model using a bimodal distribution for the admittance of each line). The simulator models the AC power flow through this network, and the cost function penalizes excessively high or low voltages or any violation of rated generator capacities. More details on this motivating example is provided in the appendix.

Given a transmission network, the so-called *security-constrained optimal power flow problem* (or SCOPF [123]) is the problem of scheduling generator setpoints and power demand from loads to minimize the economic cost of generation and ensure that the network operates safely (satisfying voltage and maximum power constraints) in the event of transmission line outages. The design parameters $x = (P_g, |V|_g, P_l, Q_l)$ include the real power injection P_g and AC voltage amplitude $|V|_g$ at each generator in the network and the real and reactive power draws at each load P_l, Q_l; all of these parameters are subject to minimum and maximum bounds that we model using a uniform prior distribution $p_{x,0}$. The exogenous parameters are the state $y_i \in \mathbb{R}$ of each transmission line in the network; the admittance of each line is given by $\sigma(y_i)Y_{i,nom}$ where σ is the sigmoid function and $Y_{i,nom}$ is the nominal admittance of the line. The prior distribution $p_{y,0}$ is an independent Gaussian for each line with a mean chosen so that $\int_{-\infty}^{0} p_{y_i,0}(y_i)dy_i$ is equal to the likelihood of any individual line failing (e.g. as specified by the manufacturer; we use 0.05 in our experiments). The simulator S solves the nonlinear AC power flow equations [124] to determine the state of the network, and the cost function combines the economic cost of generation c_g (a quadratic function of P_g, P_l, Q_l) with the total violation of constraints on generator capacities, load requirements, and voltage amplitudes:

$$J = c_g + v(P_g, P_{g,min}, P_{g,max}) + v(Q_g, Q_{g,min}, Q_{g,max}) \tag{5.15}$$

$$+ v(P_l, P_{l,min}, P_{l,max}) + v(Q_l, Q_{l,min}, Q_{l,max}) \tag{5.16}$$

$$+ v(|V|, |V|_{min}, |V|_{max}) \tag{5.17}$$

where $v(x, x_{min}, x_{max}) = L\left([x - x_{max}]_+ + [x_{min} - x]_+\right)$, L is a penalty coefficient ($L = 100$ in our experiments), and $[\circ]_+ = \max(\circ, 0)$ is a hinge loss.

Efficient solutions to SCOPF are the subject of active research [123], [125]. In addition to its potential economic and environmental impact [124], SCOPF is also a useful benchmark problem for 3 reasons: 1) it is highly non-convex, 2) it has a large space of possible failures, and 3) it can be applied to networks of different sizes to test an algorithm's scalability. In our case, the 14-bus network has 32 design parameters and 20 exogenous parameters, while the 57-bus network has 98 design parameters and 80 exogenous parameters.

A high-level comparison of RADIUM with existing methods on these SCOPF problems is

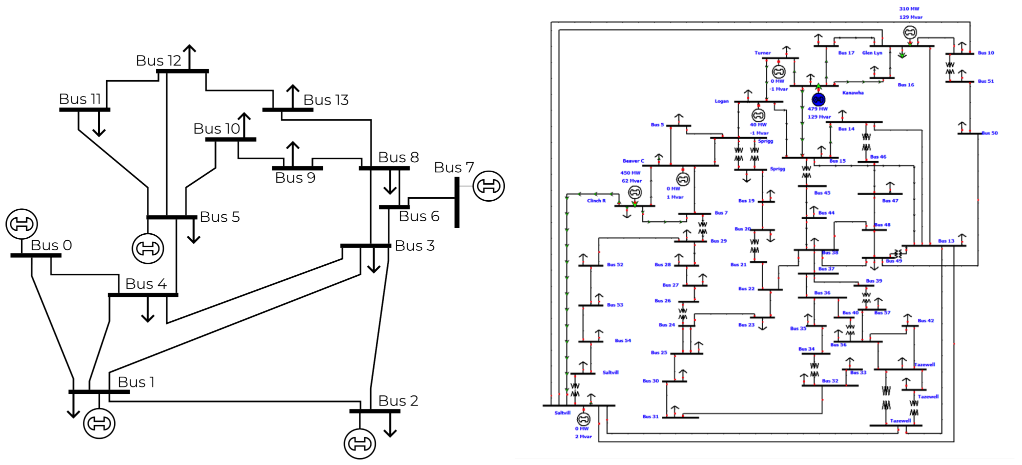
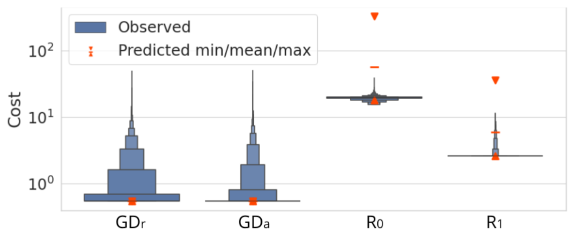


Figure 5.10: Example 14- and 57-bus electricity transmission networks [126].

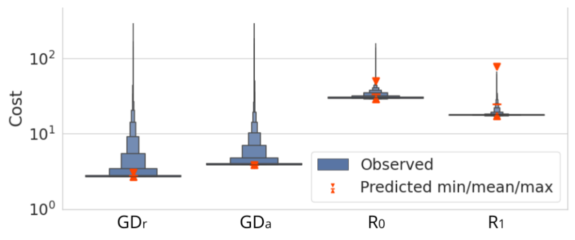
shown in Figs. 5.6 and 5.8a; in this section, we provide a more detailed comparison between RADIUM and optimization-based methods that are the state-of-the-art for this SCOPF problem [34] (the comparison with *L2C* is not shown in this section, since it is not able to solve the SCOPF problem).

Solution quality To compare the quality of these methods' solutions, we use each method to optimize 10 candidate designs and predict 10 failure modes using Algorithm 5.2.1. We then select the design that achieves the highest likelihood according to Eq. (5.4), then use one additional round to sample new failure modes that attack the chosen design. The maximum cost across these final predicted failure modes provides a measure of each algorithm's confidence in its solution. We then compare the performance on these predicted failure modes to the maximum cost observed on a test set of 10^6 exogenous parameters sampled randomly from the prior $p_{y,0}$. Fig. 5.11a shows the predicted and observed costs for each method on the IEEE 14-bus test case. The prediction-and-mitigation process takes 30.5 s for GD_r, 61.5 s for GD_a, 111.5 s for R_0, and 141.7 s for R_1 (including the cost of JAX just-in-time compilation).

We can assess these methods in two ways: by the quality of the optimized design and by the quality of the predicted failure cases. The two optimization-based methods, GD_a and GD_r, find solutions with the lowest best-case cost, but their solutions are not robust. Not



(a) 14-bus network



(b) 57-bus network

Figure 5.11: Comparison of our method with baselines for failure prediction and mitigation on power transmission networks. Red markers show the maximum, mean, and minimum-cost failure modes predicted by each method after optimizing the design, while the box plot shows the distribution of costs on a test set of 10^6 random failures.

only do these methods find solutions that are susceptible to a heavy tail of failures, but they are overconfident and fail to predict those failures (instead predicting that all 10 candidate failures will be successfully mitigated). In contrast, R_0 is not overconfident (it successfully predicts failure modes that match the range of the empirical failure distribution), but it finds a solution that is 10 times costlier than those found by R_1. Only R_1 is able to find a robust, low-cost solution without being overconfident.

Once we have a robust design optimized using our method, we can examine the predicted failure modes to understand the remaining ways in which our design might fail. Of the ten failure modes predicted by R_1, four include attacks on the transmission line connecting the generator at bus 7 to the rest of the network (this was the most commonly attacked line). Interestingly, of these four attacks, only one (shown in Fig. 5.12) is able to cause a violation of the voltage stability constraints. It is only by fully disconnecting the generator at bus 7 (dotted red line) and partially impairing the line between buses 1 and 4 (solid red; 15% impairment) that we see the voltage drop at several buses (shown in orange). This information about potential failure modes can be very useful to system designers; in this example, the designer may choose to focus monitoring and infrastructure hardening efforts on the two affected lines.

To understand the scalability of our approach, we repeat this experiment on a larger 57-bus network with 80 transmission lines. All hyperparameters were the same except for

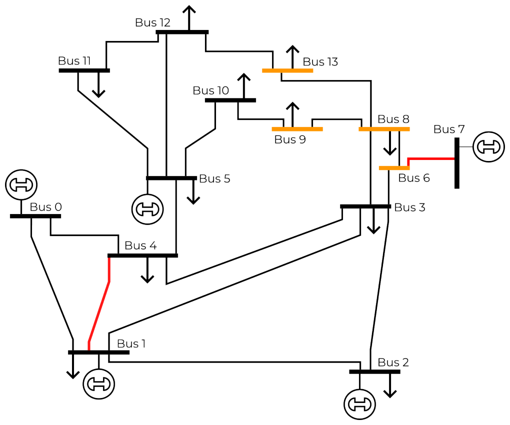


Figure 5.12: The only predicted failure modes (of 10 candidates) that causes violation of voltage constraints on the 14-bus transmission network, using the optimized design found using R_1. Fully disconnecting the generator at bus 7 (dotted red line) is not enough; other predicted failures include this outage but do not cause a constraint violation. It is only by additionally impairing the line between buses 1 and 4 that the voltage constraint is violated at the buses shown in orange.

the step size for exogenous parameters, which was reduced to 10^{-3}. The results are shown in the bottom panel of Fig. 5.11a; we see that R_1 continues to not only find a robust solutions (with a relatively light tail of failures) but also accurately predicts the range of possible failure modes (R_0 does not explore the full failure space in this case). Running GD_r on this example takes 406.2 s, GD_a takes 803.4 s, R_0 takes 1002.3 s, and R_1 takes 1438.5 s.

Convergence rate From comparing solution quality, there is a clear separation between the sampling- and optimization-based methods, with sampling able to find more robust designs and more accurately cover the range of possible failures (although gradient-based sampling finds higher-quality solutions than gradient-free sampling in both cases). A natural next question is how quickly these methods converge to a solution.

To measure the relative convergence rate of GD_r, GD_a, R_0, and R_1, we measure the 99th percentile cost J of the candidate design $[x]_i$ with the highest log likelihood (5.4) on a test set of 1000 exogenous parameters sampled randomly from the prior. The convergence of this test-set performance as a function of the number of sampling rounds is shown in Fig. 5.13 for both the 14- and 57-bus networks.

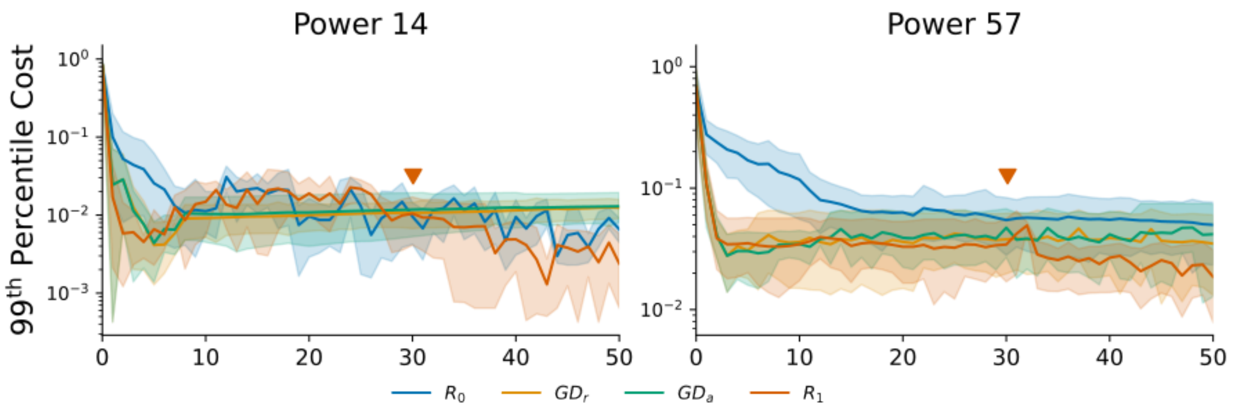


Figure 5.13: Comparison of convergence rates of different methods on 14- (left) and 57-bus (right) power networks, showing the 99th percentile cost of the best design after each round. The ▼ symbol indicates the start of quenching for R_1.

There are two important conclusions to be drawn from comparing the convergence rates in Fig. 5.13. The first is that the gradient-based methods, GD_r, GD_a, and R_1, converge faster than gradient-free R_0. Although the SCOPF problem (as well as ACOPF, the non-adversarial version) are non-convex optimization problems, there are known convex relaxations [127], and gradient descent-based methods have been shown to work well for finding local optima [34], [128], so the good performance of gradient-based methods is not surprising.

The second important conclusion from Fig. 5.13 is the importance of quenching R_1 on this problem (i.e. disabling the stochastic part of the sampling algorithm and taking a few gradient descent steps during the final rounds). Prior to quenching, R_1 converges to a solution with similar 99th percentile cost as GD_r and GD_a, but after quenching (in the last 20 rounds), R_1 is able to find a solution with a much lower cost. This is likely because of the constraints on P, Q, and $|V|$ in the SCOPF problem; sampling based methods may struggle to exactly satisfy constraints like these, as $MALA$ and RMH are constantly injecting noise into the solution, and so running a few steps of gradient descent on θ towards the end of the repair process likely helps drive constraint violations to zero by converging to the local minima nearest to the solutions explored by the sampling process. The fact that R_1 converges to a lower-cost solution than GD_r or GD_a, despite running the same gradient-based optimization process during the quenching phase, suggests that the improved exploration of designs and failures due to sampling provides an advantage over pure optimization-based methods on this problem.

5.4.2 Multi-agent control and path planning

We return to the search problem used as a baseline in Section 5.3, where a team of seeker robots must cover a region to detect a set of hiders. The simulation environment includes n_{seek} seeker robots and n_{hide} hider robots. Each robot is modeled using single-integrator dynamics and tracks a pre-planned trajectory using a proportional controller with saturation at a maximum speed chosen to match that of the Robotarium platform [129]. The trajectory $\mathbf{x}_i(t)$ for each robot is represented as a Bezier curve with 5 control points $\mathbf{x}_{i,j}$,

$$\mathbf{x}_i(t) = \sum_{j=0}^{4} \binom{4}{j} (1-t)^{4-j} t^j \mathbf{x}_{i,j}$$

The design parameters are the 2D position of the control points for the trajectories of the seeker robots, while the exogenous parameters are the control points for the hider robots. The prior distribution for each set of parameters is uniform over the width and height of the Robotarium arena ($3.2\,\mathrm{m} \times 2\,\mathrm{m}$).

We simulate the behavior of the robots tracking these trajectories for $100\,\mathrm{s}$ with a discrete time step of $0.1\,\mathrm{s}$ (including the effects of velocity saturation that are observed on the physical platform), and the cost function is

$$J = \sum_{i=1}^{n_{hide}} \left(\widetilde{\min_{t=t_0,\ldots,t_n}} \left(\widetilde{\min_{j=1,\ldots,n_{seek}}} \left\| \mathbf{p}_{hide,i}(t) - \mathbf{p}_{seek,j}(t) \right\| - r \right) \right)$$

where r is the sensing range of the seekers ($0.5\,\mathrm{m}$ for the $n_{seek} = 2$ case and $0.25\,\mathrm{m}$ for the $n_{seek} = 3$ case); $\widetilde{\min}(\cdot) = -\frac{1}{b}\mathrm{logsumexp}(-b\ \cdot)$ is a smooth relaxation of the element-wise minimum function where b controls the degree of smoothing ($b = 100$ in our experiments); $t_0, \ldots, t_n$ are the discrete time steps of the simulation; and $\mathbf{p}_{hide,i}(t)$ and $\mathbf{p}_{seek,j}(t)$ are the

(x, y) position of the i-th hider and j-th seeker robot at time t, respectively. In plain language, this cost is equal to the sum of the minimum distance observed between each hider and the closest seeker over the course of the simulation, adjusted for each seeker's search radius.

We deploy the optimized hider and seeker trajectories in hardware using the Robotarium multi-robot platform [129] (we use 3 seekers and 5 hiders, since we had difficulty testing with more agents in the limited space). We first hold the search pattern (design parameters) constant and optimize evasion patterns against this fixed search pattern, yielding the results shown on the left in Fig. 5.14 where the hiders easily evade the seekers. We then optimize the search patterns using our approach (with $K = 100$ rounds and $M = 10$ substeps per round, taking 41 s), yielding the results on the left where the hiders are not able to evade the seekers. Trajectories for the hiders and seekers were planned offline and then tracked online using linear trajectory-tracking controllers.

Although the gap between simulation and reality is not particularly large in this case, there are effects present in the hardware system that we did not model in our simulator (e.g. the collision-avoidance safety filter used by the Robotarium). Despite this small gap, this case study serves as a useful proof-of-concept for transferring optimized designs from simulation to hardware.

In the next case study, we will demonstrate the transfer of failure modes and repaired designs in a much more challenging environment with vision in the loop.

5.4.3 Vision-in-the-loop control of a 1/10-scale race car

In our simulation studies in Section 5.3, we introduced a highway driving task where an autonomous vehicle must overtake two slower cars using visual feedback to avoid collision with other agents. In this case study, we return to this example, developing a hardware test environment where the autonomous agent controls a $1/10^{\text{th}}$-scale race car and the two non-ego agents are represented by TurtleBots.

We pre-train a policy for the highway task that has three components: a tracking con-

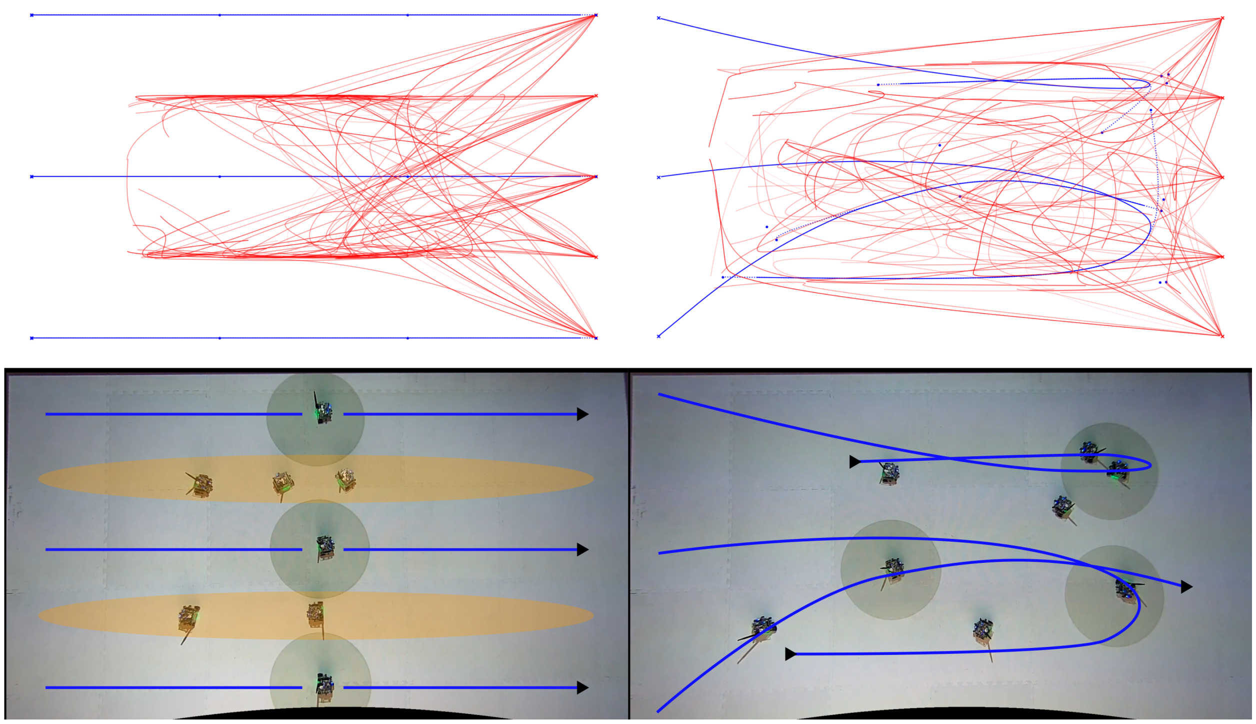


Figure 5.14: (Left) HW results for search-evasion with 5 hiders and 3 seekers, showing an initial search pattern (seeker trajectories; blue) and predicted failure modes (hider trajectories; red). (Right) HW results for an optimized search pattern leaves fewer hiding places. The top row shows the predicted failure modes (i.e. hider trajectories), while the bottom row shows a snapshot from hardware executions of these trajectories.

troller that follows a pre-planned trajectory, a model-based collision avoidance controller that attempts to avoid rear-ending cars in front of the ego vehicle (using the depth camera to measure the distance to the next car), and a neural network controller that accelerates and steers based on the depth image received from a forward facing camera. The parameters of the neural network and the pre-planned trajectory are optimized via vanilla gradient descent during pre-training. The cost function is the negative minimum reward observed during each rollout:

$$J = -\widetilde{\min}_t \left(r_t\right) \tag{5.18}$$

$$r_t = 0.1(x_t - x_{t-1}) - 5\sigma(-5d_t) \tag{5.19}$$

where $\widetilde{\min}_t$ is a soft minimum (as defined in Chapter 4) over time, with sharpness parameter

10. r_t is the reward at each timestep, defined as the weighted sum of the distance traveled along the highway and a term that penalizes collisions. σ is the sigmoid function and d_t is the minimum distance between the ego vehicle and the nearest obstacle at time t.

We then use R_1 to predict failure modes for this pre-trained, vision-in-the-loop policy in simulation, then transfer both the pre-trained policy and predicted failure modes to hardware. As shown in Fig. 5.15a, the failure modes predicted in simulation correspond to real failures on hardware. We then repair the policy using R_1, which results in an updated policy and new predicted worst-case failure mode, shown in Fig. 5.15b. Since R_1 predicts different failure modes for the nominal and repaired policies, Table 5.2 compares the failure rates of both policies on 20 independent samples of ϕ from the prior distribution, showing that the repaired policy is 5x safer than the original policy.

These results demonstrate that both the predicted failure modes and the repaired policy can successfully transfer from simulation to hardware, despite the gap between the simulated dynamics and rendering system and reality. A benefit of the sampling-based approach proposed in R_0 and R_1 is that the noise added during the sampling step helps us avoid converging to narrow local minima, avoiding failures that occur only due to quirks in the simulation environment.

Table 5.2: Failure rate on 1000 simulated and 20 hardware trials of nominal and repaired policies with exogenous parameters sampled i.i.d. from the prior $p_{\phi,0}$.

Policy	Failure rate (simulation)	Failure rate (hardware)
Nominal	4.4 %	25%
Repaired	0.6 %	5%

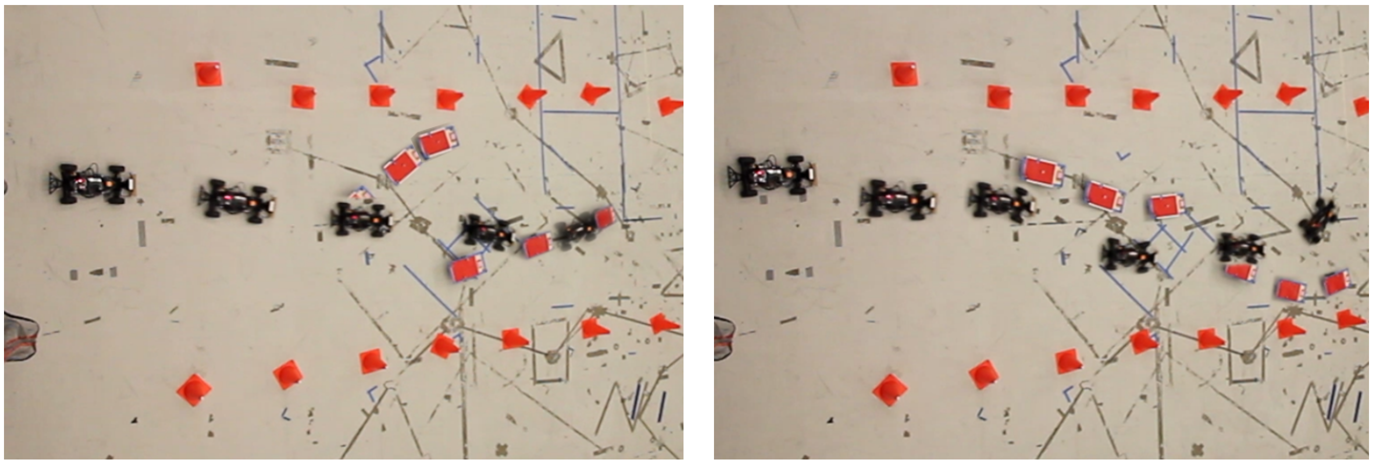

(a) Nominal policy (b) Repaired policy

Figure 5.15: Composite images from hardware experiments with vision-in-the-loop controllers. (Left) the nominal policy and the worst adversarial example found using our method, where a crash occurs. (Right) the repaired policy and new worst adversarial example (both found using our method); the repaired policy avoids crashing in this case.

5.5 Summary

In this chapter, we close several key gaps in the prior work on adversarial design optimization. In particular, we reframe the adversarial optimization problem studied in Chapter 4 as a inference problem solved by sampling from specially constructed pseudo-posterior distributions. This reformulation brings three key advantages. First, it provides practical advantages through the use of gradient-based sampling algorithms that are robust to high-variance or poorly conditioned gradients and are able to avoid getting stuck in local minima. Second, this approach allows us to not only sample a diverse range of high-severity failures but also simultaneously re-sample policy updates to repair those failures, sharing computation between subsequent rounds of predictions and updates to improve efficiency. Finally, our use of well-established MCMC algorithms allows us to draw on a rich set of theoretical tools to analyze the properties of our proposed algorithm, deriving conditions on polynomial-time convergence (with respect to dimension) and characterizing the equilibrium distribution of robust designs.

We apply our approach to a range of robotics and cyberphysical control problems, demonstrating how the use of gradients from differentiable simulation and rendering can help accelerate convergence. However, we acknowledge that a differentiable simulation environment is not always available, and that substantial engineering effort can be required to develop such a simulator. Because of this difficulty, and because simulators of certain phenomena (e.g., contact [50] and rendering with occlusion [47]) can yield inaccurate gradients if care is not taken, we build graceful degradation into our approach, providing a gradient-free version of our algorithm that can be used when gradients are not available. When gradient quality is poor, we find empirically that the acceptance rate of R_1 tends to zero, providing feedback to the user to either improve their differentiable simulator or switch to R_0 (which we find outperforms gradient-free baselines on the benchmark problems studied in this chapter).

Although the RADIUM framework developed in this chapter closes a number of important technical gaps from previous chapters, there are several limitations that remain to be addressed. First, these methods rely on a manually-specified cost function to define failures. For some complex systems, it may be difficult to specify an appropriate cost function *a priori*, and doing so may lead our method to focus only on those failures defined by the cost function, preventing the discovery of other undesirable behaviors. Second, RADIUM relies entirely on simulation for discovering failures and repairing designs; although we can transfer RADIUM's designs and predicted failures from simulation to hardware, there is no immediately obvious way to feed the results of hardware experiments back into RADIUM. In the next chapter, we introduce a method for data-driven failure analysis that allows us to close the loop between simulation and real-world data, while avoiding the need for a hand-specified cost function.

Chapter 6

Robust anomaly diagnosis with calibrating normalizing flows

The methods developed in previous chapters rely heavily on simulation to predict and repair failures in autonomous systems. Although simulation-driven testing is an important part of the development process for these systems, the ultimate goal is to deploy the system in the real world. Unfortunately, testing a system in the real world presents a number of challenges for the methods presented thus far in this thesis. Whereas simulation testing allows us to vary environmental parameters in order to preemptively predict failures, in reality we are often faced with the problem of analyzing a failure after the fact (often called a *post-mortem* analysis). The goal of this analysis is to infer what went wrong based on data collected during a particular failure event; i.e. what changes in the environment were associated with the observed failure.

This analysis is challenging for a number of reasons. First, data collected from a system operating in the real world is often noisy and prone to outliers. Second, if we have been responsible in developing and deploying our system, failures should be relatively rare, meaning that while we may have a large amount of data from normal operation of the system, we typically have very little data from the failure itself. This data imbalance makes it difficult to apply many learning and inference methods directly to post-mortem analysis.

To formalize this problem, we can frame post-mortem analysis as a Bayesian inverse problem (IP), where we aim to infer the distribution of latent variables z from noisy observations x of a stochastic process $x \sim p(x|z; y)$, where y are known context variables [130]–[133][1]. In a traditional Bayesian IP setting, we are given one or more i.i.d. samples $\{y_i, x_i\}$, but in the anomaly diagnosis setting there is a *data imbalance* between a large number of samples $\mathcal{D}_n = \{y_i, x_i\}_{i=1,\ldots,N_n}$ from nominal operations and a much smaller number of examples observed during the anomaly $\mathcal{D}_a = \{y_j, x_j\}_{j=1,\ldots,N_a}$, where $N_a \ll N_n$. This data-constrained setting is related to, but distinct from, out-of-distribution detection (where $\mathcal{D}_a$ is not known; [134]–[136]) and few-shot learning (where $\mathcal{D}_a$ is unknown during training but known at inference time; [137]).

Given these data, anomaly diagnosis aims to infer the *nominal distribution* $p(z|\mathcal{D}_n)$ conditioned solely on the nominal data and the *anomaly distribution* $p(z|\mathcal{D}_a, \mathcal{D}_n)$ conditioned on all available data. Sampling from each of these distributions helps us understand what changes in the latent variables were associated with the observed anomaly (helping us ask "what went wrong?"), while comparing the likelihoods of these distributions allows us to test for the presence of anomalies in future data. Unfortunately, imbalanced data in anomaly diagnosis problems makes it challenging to apply existing inference methods, which risk either overfitting to noise in the limited anomaly data or underfitting the anomaly in favor of the large nominal dataset.

In this chapter, we address this gap by introducing CalNF, or calibrated normalizing flows. To make full use of available data, CalNF amortizes inference over both the nominal and anomaly data, learning a shared representation for both posteriors, but it prevents overfitting using a novel subsample-then-calibrate approach to learn an optimal representation for the anomaly posterior. In contrast to existing methods for regularized distribution

[1]In this chapter, we use x, y, and z to denote the observation, context, and latent variables to align with standard inverse problem notation. In the context of the notation used in previous chapters, x denotes the system's behavior, represented as a trace of states $s_1, \ldots, s_T$, while y and z denote known and unknown components of the exogenous parameters, respectively

learning, our method does not require manual hyperparameter tuning, and it exceeds the performance of hand-tuned baselines on a range of challenging data-constrained inference problems.

To demonstrate the real-world applicability of CalNF, we apply our method to a post-mortem analysis of the 2022 Southwest Airlines scheduling crisis, which stranded more than 2 million passengers during a winter storm and led to more than $750 million in financial losses [138]. Our analysis provides new insights into the dynamics of the Southwest network and suggests that an imbalanced distribution of aircraft at key airports (other than those affected by the storm) may have contributed to the failure.

This chapter is organized as follows. Section 6.1 provides relevant background on inverse problems and normalizing flows. Section 6.2 introduces CalNF, and Section 6.3 compares our approach to existing regularized inference methods on a range of benchmarks. Section 6.4 presents our main case study: a data-driven post-mortem analysis of the 2022 Southwest Airlines scheduling crisis. Section 6.5 concludes and identifies directions for future work.

6.1 Background

6.1.1 Variational inference for Bayesian inverse problems

There is a large body of work dealing with IPs from a Bayesian perspective. Historically, Markov chain Monte Carlo (MCMC) methods have been the gold standard for posterior sampling, but the computational expense of MCMC motivates the use of approximate algorithms like variational inference (VI; [130]). These methods optimize the parameters of a variational guide that approximates the true posterior $q_\phi(z) \approx p(z|x;y)$ by maximizing the evidence lower bound (ELBO) on the dataset $\mathcal{D}$,

$$\mathcal{L}(\phi, \mathcal{D}) = \mathop{\mathbb{E}}_{(x,y)\in\mathcal{D}} \mathop{\mathbb{E}}_{z\sim q_\phi(z)} \left[\log \frac{p(x,z;y)}{q_\phi(z)}\right]. \tag{6.1}$$

In general, q may also depend on x and y [139].

6.1.2 Normalizing flows

$\mathcal{L}$ is maximized when the variational guide matches the true posterior $q_\phi(z) = p(z|x;y)$. Classical VI methods use simple representations for q_ϕ, such as independent Gaussians, which are often not capable of matching the true posterior, motivating the use of more flexible guides like normalizing flows (NFs). NFs represent q_ϕ as the transformation of a simple base distribution q_0 (e.g. Gaussian) through an invertible mapping; e.g., $z = f_\phi(z_0)$, with $z_0 \sim \mathcal{N}(0, I)$ and a smooth bijection f_ϕ with inverse f_ϕ^{-1} [140], [141]. We can sample from this distribution by passing samples from the base distribution through f, and the exact likelihood is given in terms of the Jacobian of f as:

$$\log q_\phi(z) = \log q_0(f^{-1}(z)) - \log \left|\det J_f\left(f^{-1}(z)\right)\right|. \tag{6.2}$$

Normalizing flows have seen substantial success as flexible representations for image generation, density estimation, and inverse problems [133]. Substantial effort has been devoted to developing flows based on different choices for f [142]–[146]. Our focus in this chapter is not on proposing and evaluating a new architecture for f but rather on addressing the challenges involved in training normalizing flows in data-constrained settings.

6.2 Method: calibrated normalizing flows

The key challenge in applying existing VI methods, including those using normalizing flows, to our setting is the imbalance in the size of the nominal and failure datasets. Relying solely on anomaly data risks overfitting to noise in those data, but using both datasets risks underfitting the anomaly in favor of the much larger nominal dataset.

Existing methods attempt to resolve this issue by first learning the nominal posterior, then using it as a prior to regularize the anomaly posterior. This is commonly done by training q_{ϕ_n} on nominal data alone, then learning q_{ϕ_a} subject to a penalty on divergence from the nominal distribution [133], [147]; for example, by solving:

$$\phi_n = \arg\max_{\phi} \mathcal{L}(\phi, \mathcal{D}_n), \tag{6.3}$$

$$\phi_a = \arg\max_{\phi} \mathcal{L}(\phi, \mathcal{D}_a) - \beta D_{KL}(q_\phi || q_{\phi_n}), \tag{6.4}$$

where β is a hyperparameter that controls how close the anomaly posterior is to the nominal distribution. The Kullback-Leibler divergence D_{KL} is often used to measure closeness to the nominal distribution, but other measures may be used instead. The main challenge with this approach is that β can be difficult to tune. As we show in Fig. 6.1, too little regularization results in overfitting to noise in the scarce data, while too much makes it difficult to distinguish between the nominal and anomalous cases. There is no clear choice for how much regularization is appropriate, and so it must be tuned manually, leaving substantial room for error.

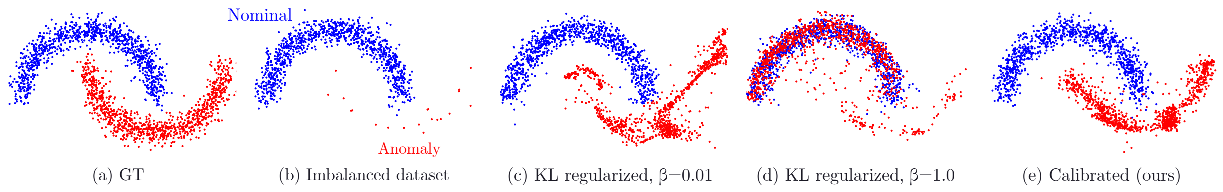


Figure 6.1: **Illustrating the effect of data imbalance.** (a) The ground truth distribution. (b) An imbalanced dataset. (c) When the regularization strength β is too small, existing methods overfit to noise in the anomaly dataset. (d) When β is too large, the learned distribution underfits the anomaly and struggles to distinguish between nominal and anomalous data. (e) Our method yields a more accurate reconstruction of the anomaly distribution by constraining the divergence between the nominal and anomaly distributions.

Our first insight is that instead of choosing a specific regularization strength β beforehand, we can train a single normalizing flow to learn a family of distributions that interpolate between the nominal and anomaly distributions (e.g. by training a single normalizing flow with a label to distinguish between the two cases), then choose the best representation from this family of distributions. Unfortunately, it is not immediately clear how the "best" representation should be chosen; simply picking the representation that best explains the anomaly data (i.e. by maximizing the evidence for the anomaly data) recovers the distribution learned without any regularization. Instead, our second key insight is that we can interpolate between the distribution of the nominal data and the distribution of different random subsets of the anomaly data, then find the best interpolation between these subsets that explains the full anomaly dataset. The family of distributions learned using this approach is shown in Fig. 6.2, where we use a one-hot label to learn the distribution of each random subset. We see that distributions learned for each of the random subsets is somewhat overfit to noise in that particular subset, but we can find a better representation of the anomaly distribution by finding the label that interpolates between these distributions and maximizes the evidence on the overall anomaly dataset. This approach takes inspiration from the intuition behind robust regression methods like RANSAC [148].

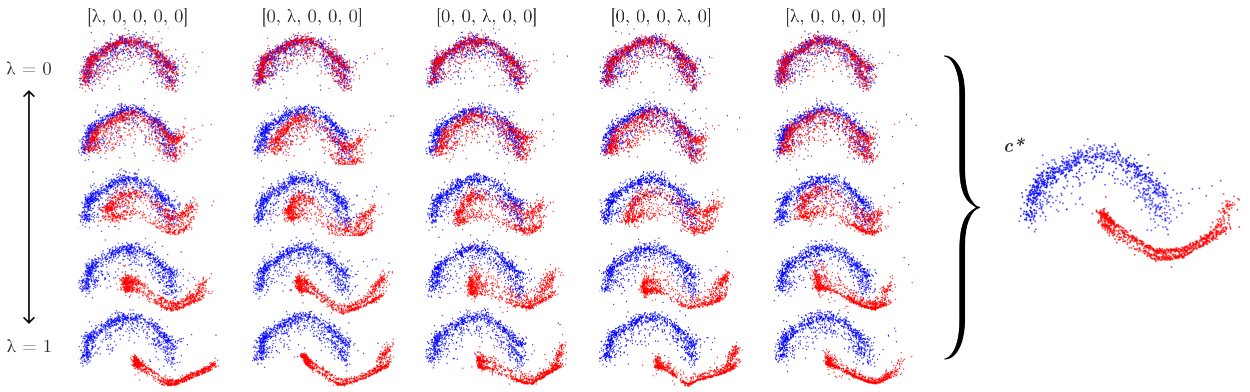


Figure 6.2: **Uncalibrated vs. calibrated posteriors.** (Left) The family of distributions learned prior to the calibration step. The red and blue points are samples from the nominal $q_\phi(z; \mathbf{0})$ and anomaly posteriors $q_\phi(z; \lambda \mathbf{1}_i)$ for $\lambda \in [0, 1]$, respectively. Even though the individual posteriors overfit to their respective subsets, the calibrated posterior (right) fits well across the full anomaly dataset.

We call this approach calibrated normalizing flows, or CalNF. The architecture of CalNF is shown in Fig. 6.3, which shows the label used to distinguish between different subsets of the anomaly data and the nominal data in more detail. In the rest of this section, we describe this architecture and the training process more formally.

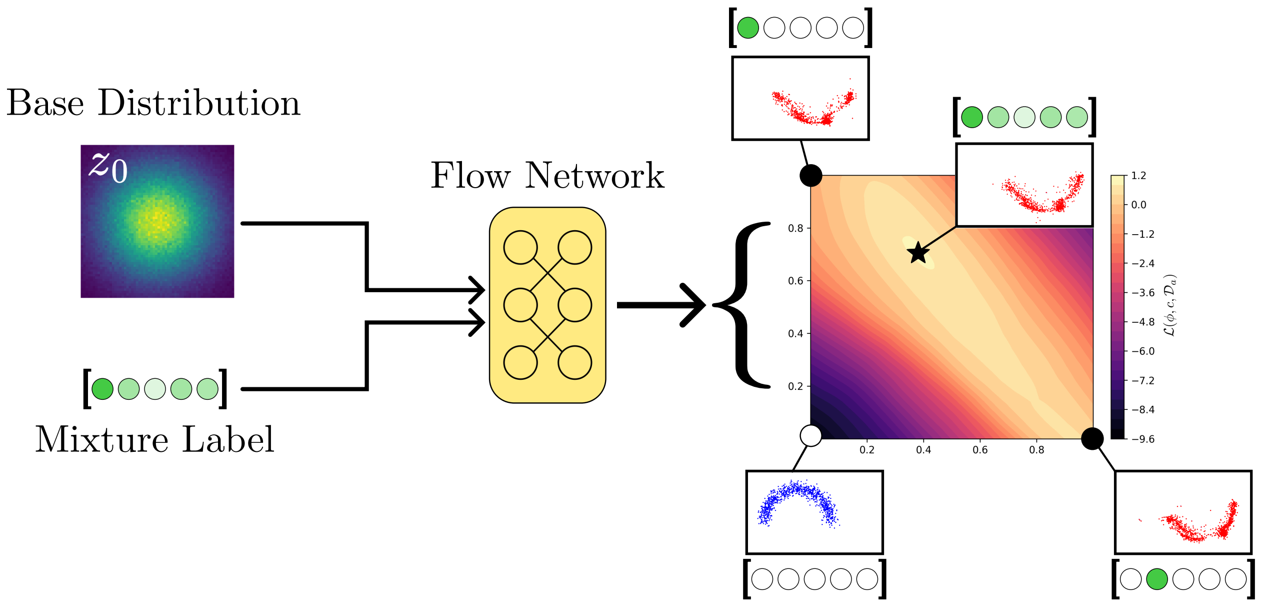


Figure 6.3: **CalNF architecture**: A normalizing flow is trained on random subsets of the anomaly data and the full nominal dataset, using one-hot labels to identify different subsets (●) and the zero vector to identify the nominal data (○). The model is calibrated by optimizing the label to find a posterior distribution that best explains the entire anomaly training dataset (★).

CalNF begins by randomly sampling K subsets of the anomaly data $\mathcal{D}_a^1, \ldots, \mathcal{D}_a^K$ and using a conditional flow $q_\phi(z; c)$ to learn a posterior for each, identifying the different subsets with one-hot labels $c_i = \mathbf{1}_i$:

$$q_\phi(z; \mathbf{1}_i) \approx p(z|\mathcal{D}_a^i), \;\; i = 1, \ldots, K$$
$$q_\phi(z; \mathbf{0}_K) \approx p(z|\mathcal{D}_n),$$

where the zero label $c = \mathbf{0}_K$ is used to identify the nominal dataset. Once posteriors have been learned for each of these subsets, we calibrate the model by finding an optimal mixture of these posteriors to explain the full anomaly dataset; i.e. holding the model weights ϕ constant and finding the optimal label c^* such that $q_\phi(z; c^*) \approx p(z|\mathcal{D}_a)$.

This two-step process is illustrated in Fig. 6.3. On an intuitive level, our approach learns a family of anomaly posteriors parameterized by the low-dimensional label c, then optimizes in the lower-dimensional label space to find a good estimate of the overall anomaly posterior,

Algorithm 6.2.1: Calibrated Normalizing Flows

```
  Input: Nominal data $\mathcal{D}_n$, anomaly data $\mathcal{D}_a$, step size $\gamma$, number of anomaly
         subsamples $K$
  Output: Model parameters $\phi$ and calibrated label $c^*$
1 for $k = 1, \dots, K$ do
2     $\mathcal{D}_a^k \leftarrow \lfloor N_a/2 \rfloor$-element random subset of $\mathcal{D}_a$
3 Initialize $\phi$, $c$
4 while $\phi$ not converged do
5     Compute $L = L_a(\phi) + L_n(\phi) + L_{\text{cal}}(\phi, c)$
6     Update model $\phi \leftarrow \phi + \gamma \nabla_\phi L$
7     Update calibration $c \leftarrow c + \gamma \nabla_c L_{\text{cal}}(\phi, c)$
```

as shown in Fig. 6.3.

It is important to note that CalNF is agnostic to the specific architecture chosen for normalizing flow (e.g. the form of f_ϕ). Our main contribution is the higher-level framework for training the model in the context of scarce anomaly data, which could in theory be used with any learned posterior representation.

The CalNF model, together with the optimized label, can be trained using Algorithm 6.2.1. This algorithm modifies the standard variational inference training process in two ways: by training on multiple random subsets of the anomaly data, and by interleaving model updates and label calibration.

First, we split the anomaly training data into K random subsets with one-hot labels and train the model to learn the posterior for each subset. Each subset $\mathcal{D}_a^i$ is created by independently drawing $\lfloor N_a/2 \rfloor$ samples from $\mathcal{D}_a$ without replacement. We denote the ELBO on a given dataset $\mathcal{D}$ as

$$\mathcal{L}\left(\phi, c, \mathcal{D}\right) = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \mathop{\mathbb{E}}_{z\sim q_\phi(z;c)} \left[\log \frac{p(x,z;y)}{q_\phi(z;c)} \right]. \tag{6.5}$$

The model parameters are updated to maximize the sum of several ELBOs: for each anomaly subset (with one-hot labels), for the nominal dataset (with a zero label), and for the full anomaly dataset (with the calibrated label c):

$$L_a(\phi) = -\frac{1}{K}\sum_{i=1}^{K} \mathcal{L}\left(\phi, \mathbf{1}_i, \mathcal{D}_a^i\right), \tag{6.6}$$

$$L_n(\phi) = -\mathcal{L}\left(\phi, \mathbf{0}_K, \mathcal{D}_n\right), \tag{6.7}$$

$$L_{cal}(\phi, c) = -\mathcal{L}(\phi, c, \mathcal{D}_a) \tag{6.8}$$

This leads to the overall loss,

$$L(\phi, c) = L_a(\phi) + L_n(\phi) + L_{cal}(\phi, c). \tag{6.9}$$

The mixture label c is initialized at $[1/K, \dots, 1/K]$ and updated to minimize $L_{cal}(\phi, c)$. In practice, we find that we can interleave optimization for ϕ and c.

6.3 Experiments

6.3.1 Benchmark problems

This section briefly introduces the data-constrained anomaly diagnosis problems used in our experiments. More details on each problem is provided in the appendix. The first benchmark is newly developed to support our case study, but the second and third are previously-published benchmark problems [149], [150]. We implement all baselines using the Pyro framework for probabilistic programming in Python [151].

Air traffic disruptions We develop a stochastic queuing model of the Southwest Airlines network using actual flight arrival and departure data published by the US Bureau of Transportation Statistics [152]. This model tracks the movement of aircraft between airports in the network, accounting for randomness in travel times, runway use times, and air traffic control (ATC) delays, as well as runway congestion and varying aircraft reserves at each airport. We base our model off of that in [153], with extensions to account for aircraft reserves. The latent variables represent travel times between airports, runway delays at each airport, and the number of aircraft stationed at each airport at the start of the day. The context includes the scheduled departures and arrivals for the day, and the observations include the actual departure and arrival time for each flight.

The nominal and anomaly datasets include data from December 1 through December 20 and December 21 through December 30 of 2022, respectively. For benchmarking, we consider only the four busiest airports in the Southwest network, but we consider larger sub-networks in our case study in Section 6.4. The four-airport sub-network has 24 latent variables. We train on $N_n = 9$ and $N_f = 4$ data points and evaluate on 4 anomalous data points (each data point is a single day with between 88–102 flights).

Geophysical imaging Seismic waveform inversion (SWI) is a well-known geophysics problem used as a benchmark for inference and physics-informed learning [150], [154], [155]. SWI seeks to infer the properties of the Earth's subsurface using seismic measurements. A source emits a sound wave that travels through the Earth before being measured by several receivers. The wave is simulated by solving the elastic wave partial differential equation (PDE) numerically, with latent variables z for the subsurface density profile, context y for the source signal, and observations x of the signal measured at each receiver [156]. The latent space has 100 dimensions. We train using $N_n = 100$ and $N_a = 4$ samples and evaluate on 500 synthetic anomaly samples.

Aerial vehicle control We also consider a failure detection benchmark for unmanned aerial vehicles (UAVs) using the ALFA dataset [149]. This dataset includes real-world data from a UAV during normal flight and during a series of failures where various control surfaces are deactivated. In this case, z parameterizes the nonlinear attitude dynamics, y includes the current state and desired orientations, and x is the next state. The latent space has 22 dimensions; we train on 10 nominal trajectories with $N_n = 2235$ data points and 1 anomalous trajectory with $N_a = 58$, and we evaluate on a second anomalous trajectory with 69 data points (both the training and evaluation anomalies are rudder failures).

Other benchmarks For completeness, we include results from the toy 2D problem in Fig. 6.1 ($N_n = 1000$, $N_f = 20$).

6.3.2 Baselines and metrics

Our main claim is that our CalNF framework is an effective way to learn the posterior when a small number of anomaly data points are available. As a result, the most relevant comparisons are to methods for posterior learning with dataset bias, which typically involve regularizing the learned posterior. In particular, we compare against two baselines: a "state-of-the-practice" method regularizing the KL divergence [133], [147] and a state-of-the-art method specific to normalizing flows that regularizes the Wasserstein distance W_2. This second method follows RNODE and related works by penalizing the squared norm of the vector field of a continuous normalizing flow [144], [157]. We implement the KL-regularized method using neural spline flows [146] and label this method β-NSF. Since each of these baselines relies on a hyperparameter to determine the strength of the regularization (β for KL regularization and λ_K for RNODE), we provide results for a range of hyperparameters.

Since the relatively large amount of nominal data makes it easy to fit the nominal distribution, we compare primarily on the basis of the evidence lower bound $\mathcal{L}$ computed on held-out anomaly data. It is important to note that while our method requires less hyperparameter tuning than the other methods, it requires additional likelihood evaluations to fit the subsampled anomaly data. To quantify this trade-off, we report the training time for all methods. All metrics report the mean and standard deviation over four random seeds. When useful, we also provide visual comparisons of the posterior distributions learned using different methods.

We implement CalNF using neural spline flows (NSF) as the underlying normalizing

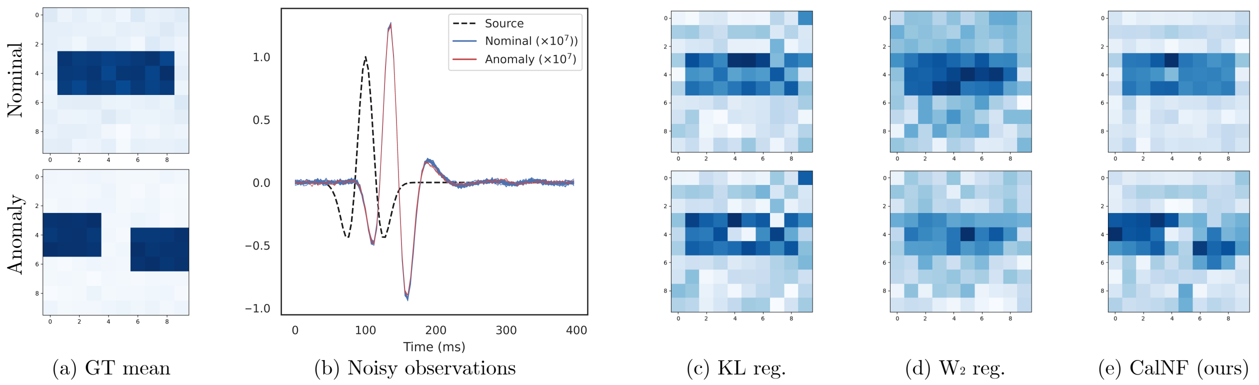


Figure 6.4: **Seismic waveform inversion.** (a) The ground truth nominal and anomalous density profiles. (b) The waveforms observed in each case. (c-e) The posteriors fit using KL regularization, W_2 regularization, and our CalNF method. Ours is the only method to correctly infer the shape of the anomaly density profile.

flow [146]. Since CalNF is agnostic to the underlying flow architecture, we also tried masked autoregressive flows [145], which trained faster but had slightly worse performance, and continuous normalizing flows [158], which trained much more slowly. We implement β-NSF using neural spline flows with a KL regularization penalty between the learned anomaly and nominal posteriors. We implement an RNODE-derived method that includes only the W_2 regularization term, not the Frobenius norm regularization term (which is used only to speed training and inference, not to regularize the learned posterior; [157]).

All methods were implemented in Pytorch using the Zuko library for normalizing flows [159]. The neural spline flows used 3 stacked transforms, and all flows used two hidden layers of 64 units each with ReLU activation (except for the continuous flows on the 2D problem, which use two hidden layers of 128 units each). All flows were trained using the Adam optimizer with the learning rate 10^{-3} (except on the UAV problem, which used a learning rate of 10^{-2}) and gradient clipping. CalNF used $K = 5$ on all problems. All methods were trained on a single NVIDIA GeForce RTX 2080 Ti GPU, with 200, 500, 1000, and 300 epochs for the 2D, SWI, UAV, and ATC problems, respectively.

Table 6.1: ELBO (nats/dim) on held-out anomaly data and training times (in minutes) on benchmark problems. 2D and SWI use synthetic data, so additional anomaly data were generated for the test set; in all other cases, half of the anomaly data was withheld for testing. Mean and standard deviation across four seeds are reported. Our method takes longer to train (requiring K times as many likelihood evaluations) but meets or exceeds the state of the art without needing additional hyperparameter tuning. †scaled by $\times 10^{-3}$

	2D	SWI	UAV	ATC
	NATS/DIM $\uparrow$	NATS/DIM† $\uparrow$	NATS/DIM $\uparrow$	NATS/DIM† $\uparrow$
β-NSF ($\beta = 0.01$)	$-3.22_{\pm 0.13}$	$43.8_{\pm 0.61}$	$3.30_{\pm 0.83}$	$-2.33_{\pm 0.05}$
β-NSF ($\beta = 0.1$)	$-2.03_{\pm 0.04}$	$43.9_{\pm 0.79}$	$3.64_{\pm 1.27}$	$-2.30_{\pm 0.05}$
β-NSF ($\beta = 1.0$)	$-1.04_{\pm 0.06}$	$44.1_{\pm 0.84}$	$2.78_{\pm 1.71}$	$-2.12_{\pm 0.09}$
RNODE ($\lambda_K = 0.01$)	$-4.58_{\pm 0.18}$	$36.0_{\pm 3.14}$	$0.76_{\pm 2.31}$	$-4.36_{\pm 1.02}$
RNODE ($\lambda_K = 0.1$)	$-2.95_{\pm 0.14}$	$36.0_{\pm 3.13}$	$0.76_{\pm 2.28}$	$-4.39_{\pm 1.08}$
RNODE ($\lambda_K = 1.0$)	$-1.67_{\pm 0.05}$	$36.0_{\pm 3.06}$	$1.14_{\pm 2.50}$	$-4.35_{\pm 1.04}$
CALNF (OURS)	$\mathbf{-0.90}_{\pm 0.10}$	$\mathbf{46.3}_{\pm 0.18}$	$\mathbf{6.95}_{\pm 1.24}$	$\mathbf{-2.01}_{\pm 0.10}$
	TIME $\downarrow$	TIME $\downarrow$	TIME $\downarrow$	TIME $\downarrow$
β-NSF ($\beta = 0.01$)	$\mathbf{0.43}_{\pm 0.02}$	$33.5_{\pm 0.2}$	$\mathbf{16.9} \pm 0.09$	$\mathbf{81.6}_{\pm 9.2}$
β-NSF ($\beta = 0.1$)	$\mathbf{0.45}_{\pm 0.03}$	$33.6_{\pm 0.2}$	$\mathbf{17.0} \pm 0.08$	$\mathbf{81.7}_{\pm 8.5}$
β-NSF ($\beta = 1.0$)	$\mathbf{0.45}_{\pm 0.03}$	$33.6_{\pm 0.1}$	$\mathbf{16.9} \pm 0.28$	$\mathbf{81.4}_{\pm 8.7}$
RNODE ($\lambda_K = 0.01$)	$5.37_{\pm 0.17}$	$\mathbf{25.1}_{\pm 0.5}$	68.0 ± 2.98	$\mathbf{82.0}_{\pm 8.4}$
RNODE ($\lambda_K = 0.1$)	$5.38_{\pm 0.19}$	$\mathbf{25.1}_{\pm 0.7}$	67.5 ± 3.60	$\mathbf{82.2}_{\pm 7.6}$
RNODE ($\lambda_K = 1.0$)	$5.23_{\pm 0.06}$	$\mathbf{24.9}_{\pm 0.7}$	69.7 ± 12.6	$\mathbf{81.8}_{\pm 8.7}$
CALNF (OURS)	$0.53_{\pm 0.02}$	$80.1_{\pm 0.5}$	45.9 ± 0.32	$148.8_{\pm 16.5}$

6.3.3 Results & discussion

Our main empirical results are shown in Table 6.1. We find that our method achieves better performance on held-out anomaly data than baselines on all problems; moreover, our method does not require manual hyperparameter tuning ($K = 5$ was sufficient for all problems). CALNF's improved performance comes at the cost of increased training time, requiring K additional likelihood evaluations per step; this difference is most significant on the SWI and ATC problems, where evaluating the likelihood is particularly expensive. On problems where the likelihood is easy to evaluate, the RNODE-derived methods are slowest to train due to their use of neural ODEs.

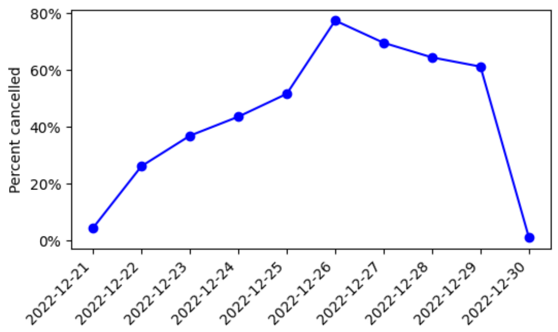
To understand the difference in performance, Fig. 6.4 compares the learned anomaly

Table 6.2: ELBO (nats/dim) on held-out anomaly data for ablations of CalNF. The first is our proposed method, the second fixes c, the third excludes the nominal data during training, and the fourth does not subsample the anomaly data. †scaled by $\times 10^{-3}$

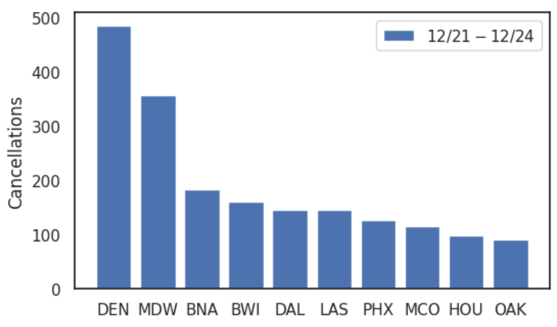
	2D	SWI†	UAV	ATC†
CalNF	$\mathbf{-0.90}_{\pm 0.1}$	$\mathbf{46.3}_{\pm 0.2}$	$6.95_{\pm 1.2}$	$\mathbf{-2.01}_{\pm 0.1}$
w/o c^*	$-0.96_{\pm 0.2}$	$46.2_{\pm 0.4}$	$\mathbf{7.86}_{\pm 1.0}$	$-2.02_{\pm 0.1}$
w/o L_n	$-1.12_{\pm 0.2}$	$46.1_{\pm 0.4}$	$-9.22_{\pm 10}$	$-2.03_{\pm 0.2}$
w/o $\mathcal{D}_a^i$	$-1.03_{\pm 0.2}$	$43.9_{\pm 2.8}$	$-3.65_{\pm 11}$	$-2.05_{\pm 0.1}$

posteriors on the SWI example, which lend themselves to easy visualization. 6.4a shows the ground truth subsurface profiles, and 6.4b shows the noisy observations. We see that the two regularization-based methods are partially successful: the KL-regularized method (6.4c) partially infers the break in the anomaly subsurface profile, and while the W_2-regularized method (6.4d) does not infer the break, it does infer the increased uncertainty in the anomaly case. However, only our method (6.4e) is able to correctly infer the shape of the anomaly profile. This suggests that our method is able to appropriately balance the information gained from the nominal distribution with the limited number of anomaly data points.

We also provide the results of an ablation study in Table 6.2, comparing the ELBO achieved when we omit the calibration step (using a constant c), omit the nominal data, and remove the subsampling step. These results indicate that most of the performance improvement from CalNF is due to training on random subsamples of the anomaly data. We observe that in cases with plentiful nominal data (like the UAV problem), including the L_n term also substantially boosts performance. We find that the benefit of optimizing c is relatively minor compared to the other components, but this step can be included for little additional computational cost, re-using the likelihood evaluation and backward pass from the main model update.



(a) Timeline of cancellations during the 2022 Southwest Airlines scheduling crisis.



(b) Cancellations at the 10 busiest airports in Southwest's network during the first four days of the disruption.

Figure 6.5: Analysis of data observed during the 2022 Southwest Airlines incident.

6.4 Post-mortem analysis of the 2022 Southwest Airlines scheduling crisis

In this section, we apply our method to a post-mortem analysis of the 2022 Southwest Airlines scheduling crisis. In the period between December 21st and December 30th, 2022, a series of cascading delays and cancellations severely disrupted the Southwest network, starting in Denver and spreading across the United States. The disruption occurred in roughly two stages, as shown in Figs. 6.5a and 6.5b. In the first stage, from 12/21 to 12/24, weather and operational difficulties caused cancellations to increase from a $< 5\%$ baseline to over 50% of scheduled flights. In the second phase, after trying and failing to recover normal operations, Southwest flight dispatchers started preemptively canceling flights and ferrying crew between airports to reset the network, canceling up to 77% of scheduled flights between 12/25 and 12/29 before returning to near-normal operations on 12/30. Southwest ultimately canceled more than 16,000 flights, affecting more than 2 million passengers, and the airline later paid a $140 million penalty imposed by the US Department of Transportation (28% of its 2023 net income; [138]) in addition to lost revenue.

This incident has been the subject of extensive investigation, with a report from Southwest Airlines [160], testimony before the US Senate from the Southwest Airlines Pilots Association (SWAPA; [161]), and press coverage [138], [162]. These sources propose a number of hypotheses on the root cause of the 2022 incident. While there is broad agreement that winter weather was a major factor, sources differ on the role of other factors; e.g. the SWAPA report emphasizes poor crew management, while press coverage emphasizes the point-to-point nature of the Southwest network.

Given this context, we have two goals for our case study. First, we are interested in identifying changes in the network state that coincided with the disruption, and how those disrupted parameters compare to the nominal state of the network. Second, we aim to produce a generative model of the nominal and disrupted network conditions to act as a tool for network design and analysis, so that future operational, scheduling, and recovery policies might be proactively stress-tested.

6.4.1 Implementation

Due to the difficulty of modeling the decision-making process of the Southwest flight dispatchers during the second half of the disruption, we focus on the first four days of the scheduling crisis, prior to the wave of cancellations aimed at resetting the network. We conduct our analysis at multiple levels of spatial resolution, looking at both the top-4 and top-10 subnetworks that include only flights between the 4 and 10 busiest airports in the Southwest network, respectively.

We modify the CalNF framework slightly for this use case; instead of randomly subsampling the anomaly data (one data point for each of the first four days of the crisis), we use each day of the crisis as a separate subsample $\mathcal{D}_a^i$. This allows us to simultaneously fit a posterior for the overall disruption (using the calibrated label c^*) and for each individual day of the disruption, respecting the time-varying nature of this problem.

The input to our air traffic model is a list of scheduled flights, each specifying an origin and destination airport and a scheduled departure and arrival time. The latent state z

includes the mean travel time between each origin/destination pair, the mean service time at each airport (which affects both arriving and departing aircraft and models taxi, deicing, and ATC delays), the mean turnaround time at each airport (the minimum time that must elapse before an arriving aircraft may depart), the baseline cancellation rate at each airport, and the initial number of aircraft at each airport.

The model steps through the scheduled flights in 15 minute increments. In each increment, it checks for the flights that are scheduled to depart from each airport. Each of these flights receives a certain probability of cancellation given by

$$P(\text{canceled}) = 1 - (1 - p_c)\sigma\left(10\frac{\#\text{ available aircraft}}{\#\text{ departing flights in this block}}\right) \tag{6.10}$$

where p_c is the baseline cancellation rate for the origin airport and σ is the sigmoid function, so the probability of cancellation is p_c when there are more available aircraft than scheduled departures and approaches 1 as the number of available aircraft decreases. Cancellations are sampled from a relaxed Bernoulli distribution with this cancellation probability and a straight-through gradient estimator. If a flight is canceled, it is marked as such and will not include actual departure and arrival times. If the flight is not canceled, then it is moved to the runway queue if there are enough aircraft available; otherwise, it is delayed until the next time block.

Both departing and arriving flights are served using a single M/M/1 queue for each airport, with service times drawn from an exponential distribution with the mean specified according to each airport's mean service time. Once airborne, departing flights are assigned a random flight time from a Gaussian with mean given by the mean travel time for each route and fixed variance. Once this travel time has elapsed, they enter the runway queue at the destination airport. Once an aircraft has landed, it does not become available to serve new flights until the minimum turnaround time has elapsed (which is sampled from a Gaussian with mean given by the mean turnaround time for each airport). Observations for noncanceled flights include the simulated arrival and departure times, plus some fixed-variance Gaussian noise. For reference, Table 6.3 includes a key of relevant three-letter airport codes.

Table 6.3: International Air Transport Association (IATA) codes and full names of the ten busiest airports in the Southwest network.

DEN	Denver International Airport
DAL	Dallas Love Field Airport
MDW	Chicago Midway International Airport
PHX	Phoenix Sky Harbor International Airport
HOU	William P. Hobby Airport
LAS	McCarran International Airport
MCO	Orlando International Airport
BNA	Nashville International Airport
BWI	Baltimore/Washington International Thurgood Marshall Airport
OAK	Oakland International Airport

6.4.2 Results

Localized delays due to winter weather. Our first observation confirms a common explanation for the disruption: that localized delays at airports across the US coincided with winter weather. For example, Fig. 6.6 shows CalNF's posterior estimates of nominal and disrupted service times, which include taxiing, deicing, and ATC delays, at the four busiest airports. Of these four, DEN, MDW, and DAL, which saw severe cold temperatures, experienced a 50% increase in average service time, while there was no corresponding increase at LAS, which did not experience severe weather. This result agrees with press and official accounts that identify winter weather and a lack of deicing equipment at critical airports like DEN as a contributing factor [160], [162]. However, the more important question is how these localized service delays cascaded into the nationwide disruption.

Cascading failures due to aircraft flow interruption. Our main finding comes from modeling the movement of aircraft within the Southwest network. The number of aircraft that start the day at each airport provides an important measure of robustness, since if

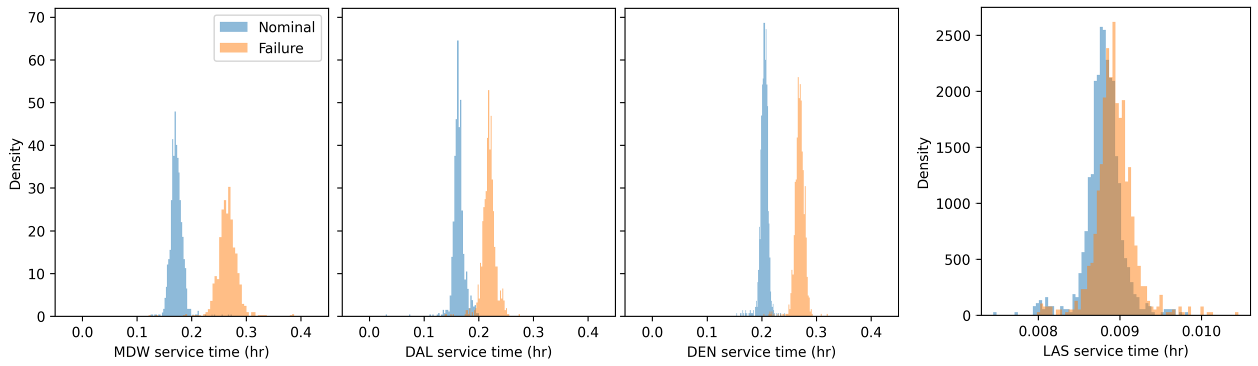


Figure 6.6: The posterior distribution (inferred using our method) indicates that service times (including taxiing, de-icing, and ATC delays) increased at DEN, MDW, and DAL, which were hit by a winter storm, but were unchanged at LAS, which did not see severe weather.

there are insufficient aircraft to meet demand, then departing flights must be delayed or canceled.[2] A lack of aircraft can also cause cancellations to cascade through the network if down-stream airports are deprived of the aircraft needed to serve scheduled departures. Despite its importance, aircraft distribution is not directly observable from public data, and so it must be inferred.

Fig. 6.7 shows our results from inferring the distribution of aircraft in the top-10 network. We use CalNF to learn the posterior distribution for each of the first four days of the disruption, then plot this inferred aircraft distribution over time. Fig. 6.7a shows that there was no detectable deviation from the nominal aircraft distribution on the first day of the disruption, but we see a steadily increasing deficit at LAS, DAL, and PHX over the following three days. The fact that the aircraft deficit at these airports continued to worsen may have been a factor in Southwest's decision to "hard reset" the network by ferrying empty planes between airports.

Figs. 6.7(b-d) show how these accumulating deficits connect localized weather-related disruptions to system-wide failure. Fig. 6.7(b) shows how cancellations during the first four days were concentrated at DEN and MDW; these cancellations likely contributed to decreased aircraft reserves at LAS, PHX, and DAL, which receive nearly 50% of their last-

[2]The same logic holds for the crew distribution. Our model assumes that crews and aircraft move together, but a separate crew model with duty time limits would be an important extension.

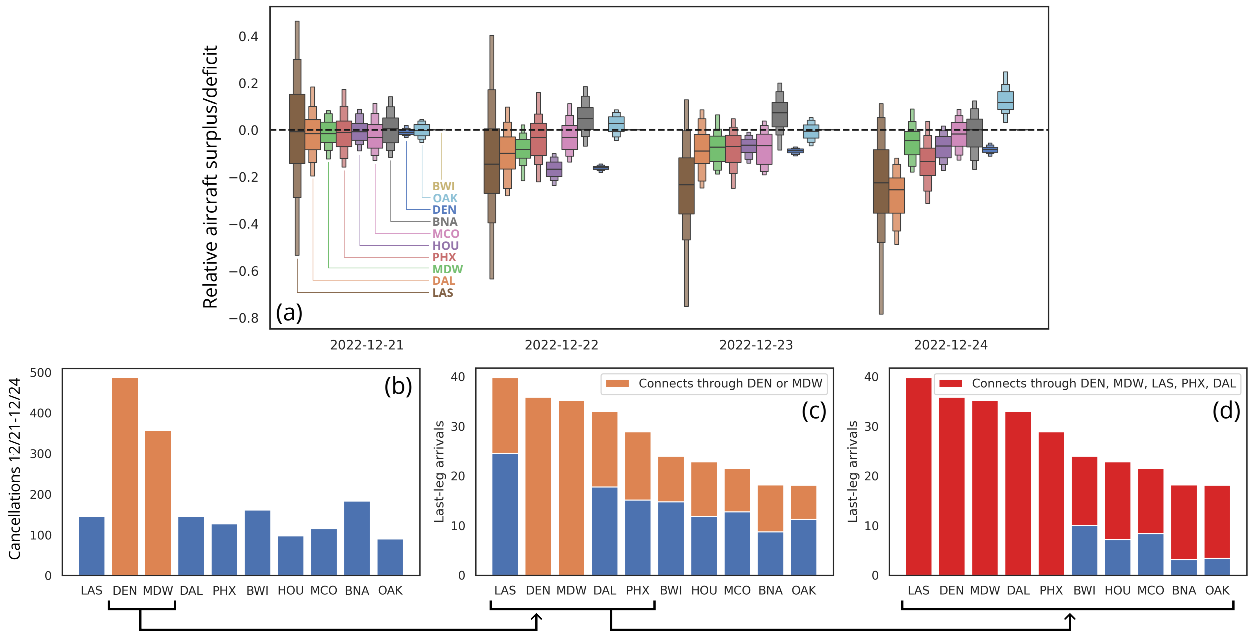


Figure 6.7: (a) Posterior estimates (inferred using our method) of the distribution of Southwest aircraft at the start of the first four days of the disruption, normalized by the number of scheduled departures at each airport; positive/negative indicates more/fewer aircraft than in the nominal case, respectively. We see that LAS, DAL, and PHX accumulate a large aircraft deficit over the course of the disruption. (b) Cancellations during the first four days of the disruption were concentrated in DEN and MDW. (c) During normal operations, LAS, DAL, and PHX (together with DEN and MDW) host the five largest overnight aircraft reserves in the network (i.e. the largest number of arrivals on the last leg of each aircraft's daily sequence). Cancellations at DEN and MDW may have cascaded to create the deficits at LAS/DAL/PHX that we observe using our method. (d) The large majority of normally-scheduled flights in the network connect through either DEN/MDW or LAS/DAL/PHX. Our analysis with CalNF suggests that the overnight aircraft reserves at LAS, PHX, and DAL played a key role in propagating weather-related disruptions at DEN and MDW to the rest of the network.

leg flights[3] from either DEN or MDW, per Fig. 6.7(c). LAS, PHX, and DAL, together with DEN and MDW, host the five largest overnight aircraft reserves in the network, and the rest of the Southwest network receives the vast majority of their incoming flights from routes visiting these airports, as shown in Fig. 6.7(d). Even though LAS and PHX did not see the same severe winter weather as DEN and MDW (DAL did experience freezing temperatures), our analysis suggests that LAS, PHX, and DAL may have played a key role in allowing the disruption to spread throughout the Southwest network. Our results indicate that trends

[3]Aircraft arriving on the last leg of their flight sequence for the day, which remain at the airport overnight.

in overnight aircraft reserves at these airports may be a valuable warning sign for detecting future disruptions.

Generative modeling Once we have learned the nominal and disruption posteriors for the Southwest network, we can use these as generative models for stress-testing proposed modifications to the Southwest network or scheduling system. In future work, these generative models could be used to design more resilient schedule recovery algorithms.

6.5 Summary

In this chapter, we propose a novel algorithm for data-constrained posterior inference, which uses a subsampling and calibration strategy to avoid overfitting to scarce data. We apply our algorithm to anomaly diagnosis problems, achieving competitive performance on challenging inverse problem benchmarks with both simulated and real data. We also apply our algorithm to a real-world anomaly diagnosis problem, providing new insight into the factors behind the 2022 Southwest Airlines scheduling crisis.

There are two major limitations of our work, which indicate directions for future research. First, it has been reported that normalizing flows struggle on out-of-distribution detection tasks, since they can assign high likelihoods to out-of-distribution samples [136]. Since our method relies heavily on normalizing flows, more work is needed before our method can be used to detect previously-unseen anomalies, building on existing out-of-distribution detection techniques.

Second, our method does not provide any estimate of the risk associated with that anomaly (i.e. how likely was it?). Estimating the risk of failure is challenging due to the size of the dataset, but we hope that future work will close this gap, potentially through the application of large deviation theory [163], allowing us to not only explain observed anomalies but also estimate their probability of occurrence.

Chapter 7

Conclusion

As autonomous systems increase in scale and complexity, there will be an increased need for automated tools for designing and verifying the safety of these systems. In particular, these tools will need to be increasingly flexible in order to support the variety and complexity of practical robotic and cyberphysical systems, which often have multiple interacting subsystems and do not easily lend themselves to mathematical abstractions and formal analysis.

In this thesis, I developed a new set of simulation-driven tools that take a program analysis approach to design and verification of complex autonomous systems. By considering the end-to-end behavior of the system, as specified by a simulator, these tools are able to simultaneously consider interactions between different subsystems and between the system and its environment. To address the scalability challenges facing previous end-to-end verification methods, I used program analysis techniques like automatic differentiation to not only solve verification problems with high-dimensional search spaces but also feed the results of verification back into the design process to improve robustness. In addition, I resolved the trade-off between local gradient-based optimization, which is efficient but can get stuck in local minima or fail due to poor-quality gradients, and black-box methods that are more robust but correspondingly more computationally expensive, through a novel Bayesian inference reformulation. Using this reformulation, I developed efficient gradient-based algorithms that can gracefully degrade to gradient-free applications.

In addition to tools that can be used to test and improve a system's performance prior to deployment, I also introduced inference-based tools for diagnosing and explaining anomalies encountered after deployment. These tools also make use of program analysis methods, particularly probabilistic programming, but they are designed to work with limited, noisy real-world datasets. I show how these tools can be used to provide explanations of system failures, linking the observed behaviors of complex systems to physically interpretable root causes.

I have applied these tools to solve challenging design and verification problems for both robotic systems, including vision-in-the-loop control for autonomous vehicles, and real-world cyberphysical infrastructure like power grids and air transportation networks. These applications demonstrate the flexibility of this approach, where the use of general-purpose program analysis tools allows us to easily adapt to new applications, but there remain a number of interesting areas for future work.

7.1 Future work

7.1.1 Integrating simulation and hardware testing

In this thesis, I focused on simulation-driven tools for safety verification and design optimization. However, no matter how extensively a system is tested in simulation, the gap between simulation and reality means that most practical systems will also require extensive testing on hardware. These hardware tests are much more expensive than testing in simulation; as a result, engineers devote considerable effort to manually designing test campaigns that begin with simulation testing and gradually build up to more expensive hardware tests. These engineers take great care to consider which subsystems will be tested at each phase of the campaign and how (or even whether) to adjust plans for later tests based on the results of early experiments. While substantial research effort, including this thesis, has been devoted to subsystem and end-to-end testing in simulation, relatively little work has gone towards the higher level problem of designing test campaigns that strike an optimal balance between simulation and hardware testing. Traditional approach to test campaign design take a linear, waterfall approach that proceeds step-by-step through increasingly higher-fidelity test environments, but the complexity of the system under test makes it difficult to design adaptive test campaigns that can flexibly intermix low- and high-fidelity testing to maximize the likelihood of detecting issues while minimizing the cost of experiments.

Recent advances in so-called "real2sim" methods that use data from hardware experiments to improve the accuracy of low-fidelity simulation environments means that it is important to consider the interplay between simulation-based and physical testing. Historically, hardware testing is conducted after exhaustive testing in simulation, but it may be more efficient to instead interleave simulation and physical tests so that early physical tests can be used to improve the accuracy of the simulation environment, which in turn will reduce the need for more expensive testing later on. There are three interesting areas of future work along these lines.

Uncertainty quantification of simulation-based testing Existing methods for testing in simulation testing do not provide a way to estimate the risk of false positive or false negative results. Quantifying this uncertainty, which may vary even between scenarios in the same simulated environment, is important not only for calibrating users' confidence in the results of simulation testing, but also for prioritizing efforts to gather additional physical data or improve the simulation environment. For example, a user might choose to focus on improving the accuracy in regions of the state space where uncertainty is high. Future work in this area may explore extending existing methods for uncertainty quantification (e.g. model-free methods based on Gaussian process regression or conformal prediction) to this problem, where the high-dimensional search space of simulation parameters may require a novel combination of model-free and model-based methods (e.g. using differentiable and probabilistic programming).

Optimal design of real2sim experiments Once we are able to quantify the uncertainty of the simulation environment in different regimes, future work might extend the automatic failure prediction and repair strategies developed in this thesis to select optimally-informative hardware experiments to reduce uncertainty in the simulation environment.

End-to-end design of test campaigns Given the ability to select optimal test cases for hardware experiments, a next step for future work would be to design test campaigns that efficiently interleave simulation and hardware testing. This would requires solving an exploration/exploitation trade-off between conducting experiments intended to improve the accuracy of the simulation environment and conducting experiments designed to verify the safety of the system.

7.1.2 Integrating safety verification and runtime monitoring

In this thesis, I used predictions of likely failure modes to repair the system's design to be more robust (e.g. fine-tuning a control policy to avoid the predicted failure modes). When we repair the policy, there is a risk of that these repairs may introduce additional conservatism; for instance, we saw in Chapter 5 that repairing the policy used by an autonomous vehicle to overtake another vehicle led to less aggressive behavior that avoided the overtake maneuver altogether. An alternative approach, and an interesting avenue for future work, would be to use the predicted failure modes as the basis for an online monitoring and recovery system that attempts to infer when a failure is imminent (e.g. by matching observations to a database of previously predicted failure modes), then taking corrective action.

Appendix A

Appendix to Chapter 3

A.1 Sensor placement design problem statement

We model the robot with discrete-time Dubins dynamics with three state variables ($q = [x, y, \theta]$), two control inputs for linear and angular velocity ($u = [v, \omega]$), and noisy transition model

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{t+1} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{t} + \begin{bmatrix} \Delta t v \cos(\theta + \Delta t \omega/2) \\ \Delta t v \sin(\theta + \Delta t \omega/2) \\ \Delta t \omega \end{bmatrix} + w_t$$

where $\Delta t = 0.5$ and $w_t \in \mathbb{R}^3$ is the actuation noise ($w_t \sim \mathcal{N}(0, Q)$ with covariance $Q \in \mathbb{R}^{3\times 3}$). The measurement model is

$$z_t = \begin{bmatrix} (x_t - x_{b1})^2 + (y_t - y_{b1})^2 \\ (x_t - x_{b2})^2 + (y_t - y_{b2})^2 \\ \theta \end{bmatrix} + v_t$$

where v_t is the measurement noise ($v_t \sim (0, R)$ and covariance $R \in \mathbb{R}^{3\times 3}$), modeling range measurements from radio or acoustic beacons b_1 and b_2 and inertial or magnetic measurements of θ. The initial state of the robot is normally distributed $q_0 \sim \mathcal{N}(\bar{q}_0, P_0)$ for mean initial state $\bar{q}_0 \in \mathbb{R}^3$ and initial covariance $P_0 \in \mathbb{R}^{3\times 3}$. The navigation function (shown in Fig. 3.4a) is $V_t(x_t, y_t) = 2(x_t^2 + y_t^2) + 0.05/d_t$ (d_t is the distance from the robot to the nearest obstacle at step t). Formally, we define this problem in the language of our framework in Table A.1.

A.2 Multi-agent manipulation design problem statement

We model each ground robot as a double integrator with states $[p_x, p_y, \theta, v_x, v_y, \omega]$. Given control inputs representing desired linear velocity v_d in the $[\cos\theta, \sin\theta]$ direction and desired angular velocity ω_d, the robot tracks those desired velocities by applying forces and torques subject to a friction cone constraint. The box is modeled as a rigid body with friction against the ground. Contact forces between the box and each robot are modeled using a penalty method described in [69], where the normal force is given by $f_n = k_c \min(\phi, 0) - k_d\dot{\phi}\mathbb{1}_{\phi<0}$ (ϕ is the signed distance between the robot and the box, $k_c = 300\,\mathrm{N/m}$ is the contact stiffness, k_d is a damping coefficient chosen to ensure critical damping, and $\mathbb{1}_{\phi<0}$ is the indicator function equal to 1 when the box and robot are in contact and 0 otherwise). Friction in the box/ground and box/robot contacts was modeled as Coulomb friction, resulting in a tangential force $f_t = \mu f_n$ with $\mu = c\psi$ if $\psi < \psi_s$ and $\mu = \mu_d$ otherwise, where mu_d is the coefficient of dynamic friction (μ_d varies for each contact pair), ψ is the tangential velocity at the point of contact, $\psi_s = 0.3\,\mathrm{m/s}$ is the tangential velocity where slipping begins, and $c = \mu_d/\psi_s$ was chosen to ensure a continuous friction model.

Each ground robot uses a proportional controller (with tunable gains) to find v_d and ω_d to track a cubic spline reference trajectory. The start point of each spline is set to match the robot's current position, the end point is set based a known offset from the desired box location, and the central control point of the spline is set using a neural network (with tunable parameters). The neural network is given inputs including the current position of each robot and the desired box pose, all referenced against the current box pose, and it predicts (x, y) locations for the control point for each robot. The network uses tanh activations on each hidden layer.

Formally, we define this problem in the language of our framework (design parameters, exogenous parameters, etc.) in Table A.2. The design parameters include the trajectory tracking control gains and network parameters, while the exogenous parameters include the desired box pose, coefficients of friction, box mass, and initial robot poses.

Table A.1: Formal statement of the sensor placement design problem with T discrete timesteps.

Design parameters	$\theta = [b_1, b_2, k] \in \mathbb{R}^6$ Beacon locations: $b_i = (x_{bi}, y_{bi}) \in \mathbb{R}^2$ for $i = 1, 2$ Feedback gains: $k \in \mathbb{R}^2$
Exogenous parameters	$\phi = [q_0, w_0, \dots, w_{T-1}, v_0, \dots, v_{T-1}] \in \mathbb{R}^{3+6T}$ Initial state: $q_0 \in \mathbb{R}^3,\ q_0 \sim \mathcal{N}(\bar{q}_0, P_0)$; $P_0 = 0.001 I_{3\times 3}$ Actuation noise: $w_t \in \mathbb{R}^3,\ w_t \sim \mathcal{N}(0, Q)$; $Q = (\Delta t)^2 \mathrm{diag}\left([0.001, 0.001, 0.01]\right)$ Measurement noise: $v_t \in \mathbb{R}^3,\ v_t \sim \mathcal{N}(0, R)$ $R = \mathrm{diag}\left([0.1, 0.01, 0.01]\right)$
Simulator	S initializes the robot with state q_0 and EKF state estimate $\bar{q}_0$ and error covariance P_0, then steps forward with interval $\Delta t = 0.5$ for $T = 60$ total steps. At each step, the simulator 1. Evaluates the navigation function to find a collision-free path to the goal, 2. Uses a feedback controller to track that path, 3. Updates the state using forward Euler integration, 4. Performs an EKF prediction, obtains a measurement z_t, and performs an EKF update. S returns a trace $s_t = [q, \hat{q}, P_{t\|t}, V_t]$ containing true states, estimated states, estimated posterior error covariance, and the value of the navigation function at each time step.
Cost	J has three components. The first ($\\|q_t - \hat{q}_t\\|^2$) minimizes the estimation error of the EKF, the second ($\\|q_t\\|$) guides the robot towards the goal, and the third (both V_t terms) avoids collision with the environment: $J = \frac{1}{T}\sum_{t=1}^{T}\left(100\left\\|q_t - \hat{q}_t\right\\|^2 + \left\\|q_t\right\\|^2 + 0.1 V_t\right) + 0.1 \max_t V_t$
Constraints	$(x_{bi}, y_{bi}) \in [-3, 0] \times [-1, 1]$ for $i = 1, 2$

Table A.2: Formal statement of the collaborative manipulation design problem using a planning network with n_p total parameters (weights and biases).

Design parameters	$\theta = [k_v, k_\omega, w_i, b_i] \in \mathbb{R}^{2+n_p}$ Trajectory tracking gains: $[k_v, k_w] \in \mathbb{R}^2$ Network weights and biases: (w_i, b_i) for $i = 1, \dots, n_p$
Exogenous parameters	$\phi = [\mu_{rg}, \mu_{bg}, \mu_{br}, m_b, p_{bd}, p_{r1}, p_{r2}] \in \mathbb{R}^{13}$ Robot/ground, box/ground, box/robot coefficients of friction: $[\mu_{rg}, \mu_{bg}, \mu_{br}] \in [0.6, 0.8] \times [0.4, 0.6] \times [0.1, 0.3]$ Box mass: $m_b \in [0.9, 1.1]$ Desired box pose: $p_{bd} = [x_d, y_d, \theta_d] \in [0, 0.5]^2 \times [-\pi/4, \pi/4]$ (Above parameters are uniformly distributed) Initial robot pose: $p_{ri} = [x_0, y_0, \theta_0] \sim \mathcal{N}(\bar{p}_{ri}, \Sigma)$; $\Sigma = 0.01 I_{3\times3}$, $i = 1, 2$.
Simulator	S initializes the robots at the initial states in ϕ relative to the box. Since these initial states may be in contact, we simulate 0.5 s of settling time at a 0.01 s timestep, then re-index the robot positions and desired box pose relative to the settled box pose. We then evaluate the planning network and track the planned path for 4 s at a 0.01 s timestep. At each timestep, 1) evaluate the spline tracking controller, 2) evaluate contact dynamics between the box, robots, and ground, and 3) integrate forces and torques to obtain box and robot states at the next timestep. S returns a trace $s_t = [q_{r1}, q_{r2}, q_b]$ containing the states of each robot and the box over time (relative to the initial pose of the box after the settling period).
Cost	J is simply the squared distance between the final box pose and the desired box position $(x - x_d)^2 + (y - y_d)^2 + (\theta - \theta_d)^2$
Constraints	Network parameters were not constrained. k_v and k_w were constrained to be less than 10.

Appendix B

Appendix to Chapter 5

B.1 Environment details

This section provides more details on the environments used as benchmark problems in Chapter 5.

B.1.1 AC power flow problem definition

The design parameters $x = (P_g, |V|_g, P_l, Q_l)$ include the real power injection P_g and AC voltage amplitude $|V|_g$ at each generator in the network and the real and reactive power draws at each load P_l, Q_l; all of these parameters are subject to minimum and maximum bounds that we model using a uniform prior distribution $p_{x,0}$. The exogenous parameters are the state $y_i \in \mathbb{R}$ of each transmission line in the network; the admittance of each line is given by $\sigma(y_i)Y_{i,nom}$ where σ is the sigmoid function and $Y_{i,nom}$ is the nominal admittance of the line. The prior distribution $p_{y,0}$ is an independent Gaussian for each line with a mean chosen so that $\int_{-\infty}^{0} p_{y_i,0}(y_i)dy_i$ is equal to the likelihood of any individual line failing (e.g. as specified by the manufacturer; we use 0.05 in our experiments). The simulator S solves the nonlinear AC power flow equations [124] to determine the AC voltage amplitudes and phase angles $(|V|, \theta)$ and the net real and reactive power injections (P, Q) at each bus (the behavior

ξ is the concatenation of these values). We follow the 2-step method described in [128] where we first solve for the voltage and voltage angles at all buses by solving a system of nonlinear equations and then compute the reactive power injection from each generator and the power injection from the slack bus (representing the connection to the rest of the grid). The cost function J is a combination of the generation cost implied by P_g and a hinge loss penalty for violating constraints on acceptable voltages at each bus or exceeding the power generation limits of any generator, as specified in Eq. B.3. The data for each test case (minimum and maximum voltage and power limits, demand characteristics, generator costs, etc.) are loaded from the data files included in the MATPOWER software [164].

Cost The cost function combines the economic cost of generation c_g (a quadratic function of P_g, P_l, Q_l) with the total violation of constraints on generator capacities, load requirements, and voltage amplitudes:

$$J = c_g + v(P_g, P_{g,min}, P_{g,max}) + v(Q_g, Q_{g,min}, Q_{g,max}) \tag{B.1}$$

$$+ v(P_l, P_{l,min}, P_{l,max}) + v(Q_l, Q_{l,min}, Q_{l,max}) \tag{B.2}$$

$$+ v(|V|, |V|_{min}, |V|_{max}) \tag{B.3}$$

where $v(x, x_{min}, x_{max}) = L\left([x - x_{max}]_+ + [x_{min} - x]_+\right)$, L is a penalty coefficient ($L = 500$ in our experiments), and $[\circ]_+ = \max(\circ, 0)$ is a hinge loss.

Hyperparameters Prediction and repair experiments were run for $N = 50$ rounds of $K = 10$ steps with $n = 10$, learning rate $\lambda = 10^{-2}$ for ϕ and 10^{-6} for θ, tempering parameter $\alpha = 10$, and $J^* = 4$ and 6 for the 14-bus and 57-bus networks, respectively. R_1 was quenched for 25 and 20 rounds for the small and large networks, respectively. Gradients were clipped with magnitude 100 times the square root of the dimension of the parameter space.

B.1.2 Search problem definition

This problem includes n_{seek} seeker robots and n_{hide} hider robots. Each robot is modeled using single-integrator dynamics and tracks a pre-planned trajectory using a proportional controller with saturation at a maximum speed chosen to match that of the Robotarium platform [129]. The trajectory $\mathbf{x}_i(t)$ for each robot is represented as a Bezier curve with 5 control points $\mathbf{x}_{i,j}$,

$$\mathbf{x}_i(t) = \sum_{j=0}^{4} \binom{4}{j} (1-t)^{4-j} t^j \mathbf{x}_{i,j}$$

The design parameters are the 2D position of the control points for the trajectories of the seeker robots, while the exogenous parameters are the control points for the hider robots. The prior distribution for each set of parameters is uniform over the width and height of the Robotarium arena (3.2 m $\times$ 2 m).

Cost We simulate the behavior of the robots tracking these trajectories for 100 s with a discrete time step of 0.1 s (including the effects of velocity saturation that are observed on the physical platform), and the cost function is

$$J = 100 \sum_{i=1}^{n_{hide}} \left(\widetilde{\min_{t=t_0,\dots,t_n}} \left(\widetilde{\min_{j=1,\dots,n_{seek}}} \left\| \mathbf{p}_{hide,i}(t) - \mathbf{p}_{seek,j}(t) \right\| - r \right) \right)$$

where r is the sensing range of the seekers (0.5 m for the $n_{seek} = 2$ case and 0.25 m for the $n_{seek} = 3$ case); $\widetilde{\min}(\circ) = -\frac{1}{b}\text{logsumexp}(-b\ \circ)$ is a smooth relaxation of the element-wise minimum function where b controls the degree of smoothing ($b = 100$ in our experiments); $t_0, \dots, t_n$ are the discrete time steps of the simulation; and $\mathbf{p}_{hide,i}(t)$ and $\mathbf{p}_{seek,j}(t)$ are the (x, y) position of the i-th hider and j-th seeker robot at time t, respectively. In plain language, this cost is equal to the sum of the minimum distance observed between each hider and the closest seeker over the course of the simulation, adjusted for each seeker's search radius.

Hyperparameters Prediction and repair experiments were run for $N = 100$ rounds of $K = 50$ steps with $n = 10$, learning rate $\lambda = 10^{-2}$ for ϕ and θ, tempering parameter $\alpha = 5$, and $J^* = 0$. R_1 was quenched for 20 and 40 rounds for the small and large problems, respectively.

B.1.3 Formation control problem definition

This problem includes n drones modeled using double-integrator dynamics, each tracking a pre-planned path using a proportional-derivative controller. The path for each drone is represented as a Bezier, as in the pursuit-evasion problem.

The design parameters are the 2D position of the control points for the trajectories, while the exogenous parameters include the parameters of a wind field and connection strengths between each pair of drones. The wind field is modeled using a Gaussian kernel with saturation at a maximum speed that induces 0.5 N of drag force on each drone.

Cost We simulate the behavior of the robots tracking these trajectories for 30 s with a discrete time step of 0.05 s, and the cost function is

$$J = 50||COM_T - COM_{goal}|| + 5 \max_t \frac{1}{\lambda_2(q_t) + 10^{-2}}$$

where COM indicates the center of mass of the formation and $\lambda_2(q_t)$ is the second eigenvalue of the Laplacian of the drone network in configuration q_t. The Laplacian $L = D - A$ is defined in terms of the adjacency matrix $A = \{a_{ij}\}$, where $a_{ij} = s_{ij}\sigma\left(20(R^2 - d_{ij}^2)\right)$, d_ij is the distance between drones i and j, R is the communication radius, and s_{ij} is the connection strength (an exogenous parameter) between the two drones. The degree matrix D is a diagonal matrix where each entry is the sum of the corresponding row of A.

Hyperparameters Prediction and repair experiments were run for $N = 50$ rounds of $K = 50$ steps with $n = 5$, learning rate $\lambda = 10^{-5}$ for ϕ and θ on the small problem and 10^{-4}

on the large problem, tempering parameter $\alpha = 5$, and $J^* = 10.0$. R_1 was quenched for 20 rounds. Gradients were clipped with magnitude 100 times the square root of the dimension of the parameter space.

B.1.4 Grasping

Each task in the grasping environment involved a target object (either a box or a mug) on a table along with a distractor object (a cube). A 64x64 depth image was rendered from a fixed camera position and passed to a grasp identification policy, which was structured as an auto-encoder with 3 convolutional layers and 3 transposed convolutional layers (each with kernel size 7, stride 2, and 32 channels, and ReLU activations). The policy was trained to identify grasp affordances on the object in the form of an image of the same size as the output highlighting the "graspable" regions of the object (this is a common strategy in robot manipulation). θ includes all parameters of the autoencoder, and the environmental parameters ϕ included the 2D pose $[x, y, \psi]$ of the target object and the x position of the distractor object, all treated as Gaussian random variables. The affordance autoencoder network was trained using ground-truth affordances from hand-labeling of the target object.

Cost The cost function for all grasping tasks was the mean square error between the predicted and ground-truth grasp affordances.

Hyperparameters We ran experiments for $N = 5$ rounds of $K = 5$ steps with population size $n = 5$. All methods used learning rate $\lambda = 10^{-3}$ for ϕ and 10^{-5} for θ, with tempering parameter $\alpha = 40$. Quenching was not used on this problem.

B.1.5 Drone

All drone environments included a corridor 8 m wide and 30 m long, with the drone starting roughly 10 m away from the target (placed at the origin and marked by a black square and red "H"). There are 5 obstacles in the scene, modeled as green cylinders with radius 0.5 m. The drone has Dubins car dynamics restricted to the xy plane, with control actions for velocity and yaw rate. Control actions are predicted by a policy that takes 32x32 RGB and depth images as input, encodes the images using three convolutional layers (kernel size 7, stride 1, 32 channels), and predicts control actions using a fully connected network with 4 hidden layers of 32 units each (all layers used ReLU activation). θ includes all parameters of the policy, and the environmental parameters ϕ include the initial position of the drone (treated as a Gaussian random variable centered 10 m away from the target) and the initial positions of all obstacles (treated as uniformly distributed throughout the corridor between the starting point and the target). The drone's initial policy was trained to mimic an oracle with perfect state information, which we implemented as an optimization-based receding-horizon path planner with perfect information about the state and velocity of the drone and all obstacles.

Cost The cost for both drone examples was the (soft) minimum reward over a trajectory plus the distance to the goal at the end of the trajectory:

$$J = -\text{logsumexp}(-r_t) + \frac{1}{2}\sqrt{x_T^2 + y_T^2} \tag{B.4}$$

$$r_t = -10\sigma(5 \min_i d_i) \tag{B.5}$$

where the reward at each timestep r_t is based on the minimum distance $\min_i d_i$ to any obstacle in the scene and *sigma* is the sigmoid function (used as a smooth approximation of the indicator function).

Hyperparameters We ran experiments for $N = 5$ rounds of $K = 5$ steps with $n = 5$ and learning rate $\lambda = 10^{-2}$ for ϕ and θ, with tempering parameter $\alpha = 40$. Quenching was not used on this problem.

B.1.6 AV

All AV examples use bicycle kinematics for all agents, with state $[x, y, \psi, v]$, including position, heading, and velocity, and control actions $[\delta, a]$ for steering angle and acceleration. The continuous time dynamics

$$\dot{x} = v \cos \psi \tag{B.6}$$

$$\dot{y} = v \sin \psi \tag{B.7}$$

$$\dot{\psi} = \frac{v}{l} \tan \delta \tag{B.8}$$

$$\dot{v} = a \tag{B.9}$$

were discretized with timestep 0.1 s. The parameter l denotes axle length and was set to 1 m. Control actions were predicted based on 32x32 RGB and depth images using the same structure as the drone policy (3 convolutional layers and 4 fully connected layers, but the convolutional layers had kernel size 6). In both the highway and intersection tasks, θ includes all trainable parameters of the policy network.

Highway

The highway example included 2 lanes of traffic, with total width 15 m. We placed one non-ego agent in each lane; these agents track a series of waypoints using an LQR controller. The environmental parameters ϕ include all of these waypoints (5 2D waypoints per non-ego agent), which we modeled as drawn from a Gaussian distribution about a straight-line path. Future work could explore using a generative model as the prior for non-ego driving agents. The task was simulated for 60 timesteps. The driving policy was pre-trained using proximal policy optimization (PPO) with the same reward as used for testing.

Cost The cost was the soft minimum reward observed over the course of the trajectory:

$$J = -\text{logsumexp}(-r_t) \tag{B.10}$$

$$r_t = -10\sigma(5\min_i d_i) + 0.1v_t \tag{B.11}$$

where the reward at each timestep r_t is based on the minimum distance $\min_i d_i$ to any obstacle in the scene and the forward velocity ; *sigma* is the sigmoid function (used as a smooth approximation of the indicator function).

Hyperparameters We ran experiments for $N = 10$ rounds of $K = 10$ steps with population size $n = 5$. All methods used learning rate $\lambda = 10^{-2}$ for ϕ and 2×10^{-5} for θ, with tempering parameter $\alpha = 20$. Quenching was not used on this problem.

Intersection

The intersection example included a 4-way intersection, with 2 lanes of traffic in each 15 m wide road. We placed two non-ego agent moving left to right in the crossing street and one non-ego agent crossing right to left. Similarly to the highway task, these agents were controlled via LQR to track uncertain trajectories centered about straight lines. The task was simulated for 70 timesteps. The driving policy was trained using behavior cloning with supervision from an oracle with perfect information about the state of the other agents, which we implemented as a receding-horizon optimization-based path planner.

Cost The cost was the soft minimum reward observed over the course of the trajectory plus a term to test whether the ego agent successfully crosses the intersection.

$$J = -\text{logsumexp}(-r_t) + 0.1[x_T - x_{target}]_+ \tag{B.12}$$

$$r_t = -10\sigma(5\min_i d_i) \tag{B.13}$$

where the reward at each timestep r_t is based on the minimum distance $\min_i d_i$ to any obstacle in the scene and $x_{target} = 20$; $sigma$ is the sigmoid function (used as a smooth approximation of the indicator function).

Hyperparameters Prediction and repair experiments were run for $N = 10$ rounds of $K = 10$ steps with $n = 5$ and learning rate $\lambda = 10^{-2}$ for ϕ and 10^{-4} for θ, with tempering parameter $\alpha = 20$. Quenching was not used on this problem.

Appendix C

Appendix to Chapter 6

C.1 Environment details

This section provides additional details for the three types of inverse problem studied in our paper. All problems are implemented using the Pyro probabilistic programming framework [151].

C.1.1 Seismic Waveform Inversion

An illustration of the SWI problem is given in Fig. C.1. We implement the SWI problem using the Deepwave library [156]. We use latent parameters $z \in \mathbb{R}^{n_x \times n_y}$ representing the subsurface density profile (with spatial resolution n_x and n_y), context $y \in \mathbb{R}^{n_T}$ representing the source signal, and observations $x \in \mathbb{R}^{n_s \times n_r \times n_T}$ representing the signal measured at each receiver, where n_s, n_r, n_T are the number of sources, receivers, and timesteps, respectively. The observations are corrupted with additive isotropic Gaussian noise.

C.1.2 UAV Control

We model the nonlinear attitude dynamics of the UAV as a combination of an unknown linear mapping from the current and desired states to angular rates, then a nonlinear mapping from

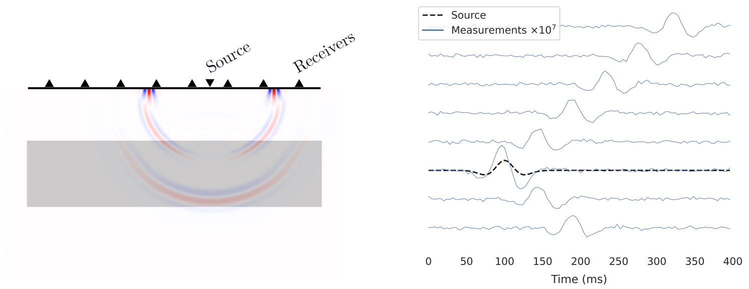


Figure C.1: (Left) An illustration of the SWI problem and (right) the receiver measurements (blue) given a source signal (black).

angular rates to updated UAV orientation. The state $q = [\phi, \theta, \psi]$ includes the roll, pitch, and yaw angles of the UAV, and $\hat{q}$ denotes the commanded orientation. We model the angular rates of the UAV as

$$\omega = \begin{bmatrix} p \\ q \\ r \end{bmatrix} = Aq + K(\hat{q} - q) + d + \eta \tag{C.1}$$

where A, K, and d are unknown feedforward, feedback, and bias dynamics, and η is Gaussian process noise. The state derivative is related to ω by

$$\frac{d}{dt}q = J^{-1}(q)\omega \tag{C.2}$$

$$J^{-1}(q) = \begin{bmatrix} 1 & \tan(\theta)\sin(\phi) & \tan(\phi)\cos(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi)/\cos(\theta) & \cos(\phi)/\cos(\theta) \end{bmatrix} \tag{C.3}$$

We apply a first-order time discretization to yield the one-step stochastic dynamics

$$q_{t+1} = q_t + \delta_t J^{-1}(q)\left(Aq + K(\hat{q} - q) + d + \eta\right)$$

and observed states are additionally corrupted by Gaussian noise.

References

[1] Y. Hu, J. Liu, A. Spielberg, J. B. Tenenbaum, W. T. Freeman, J. Wu, D. Rus, and W. Matusik, "ChainQueen: A Real-Time Differentiable Physical Simulator for Soft Robotics," in *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, QC, Canada: IEEE Press, May 2019, pp. 6265–6271. DOI: 10.1109/ICRA.2019.8794333. (visited on 03/10/2024).

[2] T. Howell, S. Le Cleac'h, Z. Kolter, M. Schwager, and Z. Manchester, "Dojo: A differentiable simulator for robotics," *arXiv preprint arXiv:2203.00806*, 2022. arXiv: 2203.00806.

[3] P. Ma, T. Du, J. Z. Zhang, K. Wu, A. Spielberg, R. K. Katzschmann, and W. Matusik, "DiffAqua: A differentiable computational design pipeline for soft underwater swimmers with shape interpolation," *ACM Transactions on Graphics (TOG)*, vol. 40, no. 4, p. 132, 2021.

[4] 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 *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 23–30. DOI: 10.1109/IROS.2017.8202133.

[5] C. Dawson and C. Fan, "Certifiable Robot Design Optimization using Differentiable Programming," in *Robotics: Science and Systems XVIII*, vol. 18, Jun. 2022, ISBN: 978-0-9923747-8-5. (visited on 05/23/2023).

[6] J. Xu, V. Makoviychuk, Y. Narang, F. Ramos, W. Matusik, A. Garg, and M. Macklin, "Accelerated Policy Learning with Parallel Differentiable Simulation," in *International Conference on Learning Representations*, Jan. 2022. (visited on 05/15/2023).

[7] F. Facchinei and C. Kanzow, "Generalized Nash equilibrium problems," *4OR 2007 5:3*, vol. 5, no. 3, pp. 173–210, Sep. 2007, ISSN: 1614-2411. DOI: 10.1007/S10288-007-0054-4. (visited on 02/27/2022).

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and Harnessing Adversarial Examples*, Mar. 2015. DOI: 10.48550/arXiv.1412.6572. arXiv: 1412.6572 `[cs, stat]`. (visited on 03/17/2023).

[9] W. Ding, B. Chen, M. Xu, and D. Zhao, "Learning to Collide: An Adaptive Safety-Critical Scenarios Generating Method," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 2243–2250. DOI: 10.1109/IROS45743.2020.9340696.

[10] Y. Zhou, S. Booth, N. Figueroa, and J. Shah, “RoCUS: Robot Controller Understanding via Sampling,” in *5th Annual Conference on Robot Learning*, Nov. 2021. (visited on 12/31/2022).

[11] M. O’ Kelly, A. Sinha, H. Namkoong, R. Tedrake, and J. C. Duchi, “Scalable End-to-End Autonomous Vehicle Testing via Rare-event Simulation,” in *Advances in Neural Information Processing Systems*, vol. 31, Curran Associates, Inc., 2018. (visited on 03/14/2023).

[12] A. Sinha, M. O’Kelly, R. Tedrake, and J. Duchi, “Neural bridge sampling for evaluating safety-critical autonomous systems,” in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS’20, Red Hook, NY, USA: Curran Associates Inc., Dec. 2020, pp. 6402–6416, ISBN: 978-1-71382-954-6. (visited on 11/23/2022).

[13] H. Delecki, A. Corso, and M. Kochenderfer, “Model-based Validation as Probabilistic Inference,” in *Proceedings of The 5th Annual Learning for Dynamics and Control Conference*, PMLR, Jun. 2023, pp. 825–837. (visited on 09/21/2023).

[14] A. Corso, R. Moss, M. Koren, R. Lee, and M. Kochenderfer, “A Survey of Algorithms for Black-Box Safety Validation of Cyber-Physical Systems,” *Journal of Artificial Intelligence Research*, vol. 72, pp. 377–428, Oct. 2021, ISSN: 1076-9757. DOI: 10.1613/jair.1.12716. (visited on 03/14/2023).

[15] J. de Kleer and B. C. Williams, “Diagnosing multiple faults,” *Artificial Intelligence*, vol. 32, no. 1, pp. 97–130, Apr. 1987, ISSN: 0004-3702. DOI: 10.1016/0004-3702(87)90063-4. (visited on 01/04/2023).

[16] D. Benard, G. A. Dorais, E. Gamble, *et al.*, “Remote Agent Experiment,” Jan. 2000. (visited on 01/04/2023).

[17] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, “S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6605 LNCS, pp. 254–257, 2011, ISSN: 03029743. DOI: 10.1007/978-3-642-19835-9_21. (visited on 02/18/2022).

[18] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, “Hamilton-Jacobi reachability: A brief overview and recent advances,” in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, Dec. 2017, pp. 2242–2253. DOI: 10.1109/CDC.2017.8263977.

[19] G. Chou, Y. E. Sahin, L. Yang, K. J. Rutledge, P. Nilsson, and N. Ozay, “Using control synthesis to generate corner cases: A case study on autonomous driving,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2906–2917, Nov. 2018, ISSN: 02780070. DOI: 10.1109/TCAD.2018.2858464. arXiv: 1807.09537. (visited on 10/15/2022).

[20] A. A. Ahmadi and A. Majumdar, “Some applications of polynomial optimization in operations research and real-time decision making,” *Optimization Letters*, vol. 10, no. 4, pp. 709–729, Apr. 2016, ISSN: 1862-4480. DOI: 10.1007/s11590-015-0894-3. (visited on 05/23/2023).

[21] A. Majumdar, A. A. Ahmadi, and R. Tedrake, "Control and verification of high-dimensional systems with DSOS and SDSOS programming," in *53rd IEEE Conference on Decision and Control*, Dec. 2014, pp. 394–401. DOI: 10.1109/CDC.2014.7039413.

[22] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *OpenAI Gym*, Jun. 2016. DOI: 10.48550/arXiv.1606.01540. arXiv: 1606.01540 [cs]. (visited on 05/24/2023).

[23] C. Xu, W. Ding, W. Lyu, Z. Liu, S. Wang, Y. He, H. Hu, D. Zhao, and B. Li, "SafeBench: A Benchmarking Platform for Safety Evaluation of Autonomous Vehicles," *Advances in Neural Information Processing Systems*, vol. 35, pp. 25 667–25 682, Dec. 2022. (visited on 04/11/2023).

[24] S. Riedmaier, T. Ponn, D. Ludwig, B. Schick, and F. Diermeyer, "Survey on Scenario-Based Safety Assessment of Automated Vehicles," *IEEE access : practical innovations, open solutions*, vol. 8, pp. 87 456–87 477, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2993730.

[25] A. Corso, P. Du, K. Driggs-Campbell, and M. J. Kochenderfer, "Adaptive Stress Testing with Reward Augmentation for Autonomous Vehicle Validatio," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, Oct. 2019, pp. 163–168. DOI: 10.1109/ITSC.2019.8917242.

[26] J. Wang, A. Pun, J. Tu, S. Manivasagam, A. Sadat, S. Casas, M. Ren, and R. Urtasun, "AdvSim: Generating Safety-Critical Scenarios for Self-Driving Vehicles," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 9909–9918. (visited on 09/15/2023).

[27] H. Sun, S. Feng, X. Yan, and H. X. Liu, "Corner Case Generation and Analysis for Safety Assessment of Autonomous Vehicles," *Transportation Research Record*, vol. 2675, no. 11, pp. 587–600, Nov. 2021, ISSN: 0361-1981. DOI: 10.1177/03611981211018697. (visited on 03/14/2023).

[28] Z. Zhong, D. Rempe, D. Xu, Y. Chen, S. Veer, T. Che, B. Ray, and M. Pavone, *Guided Conditional Diffusion for Controllable Traffic Simulation*, Oct. 2022. DOI: 10.48550/arXiv.2210.17366. arXiv: 2210.17366 [cs, stat]. (visited on 05/15/2023).

[29] A. Corso and M. J. Kochenderfer, "Interpretable Safety Validation for Autonomous Vehicles," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, Sep. 2020, pp. 1–6. DOI: 10.1109/ITSC45102.2020.9294490.

[30] Q. Zhang, S. Hu, J. Sun, Q. A. Chen, and Z. M. Mao, "On Adversarial Robustness of Trajectory Prediction for Autonomous Vehicles," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 15 159–15 168. (visited on 04/12/2023).

[31] M. J. Kochenderfer and T. A. Wheeler, *Algorithms for Optimization*. Cambridge, Massachusetts: The MIT Press, 2019.

[32] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust Physical-World Attacks on Deep Learning Visual Classification," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp. 1625–1634. DOI: 10.1109/CVPR.2018.00175. (visited on 05/10/2024).

[33] N. Hanselmann, K. Renz, K. Chitta, A. Bhattacharyya, and A. Geiger, "KING: Generating Safety-Critical Driving Scenarios for Robust Imitation via Kinematics Gradients," in *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXVIII*, Berlin, Heidelberg: Springer-Verlag, Oct. 2022, pp. 335–352, ISBN: 978-3-031-19838-0. DOI: 10.1007/978-3-031-19839-7_20. (visited on 09/15/2023).

[34] P. Donti, A. Agarwal, N. V. Bedmutha, L. Pileggi, and J. Z. Kolter, "Adversarially robust learning for security-constrained optimal power flow," in *Advances in Neural Information Processing Systems*, vol. 34, Curran Associates, Inc., 2021, pp. 28 677–28 689. (visited on 12/04/2022).

[35] J. DeCastro, L. Liebenwein, C.-I. Vasile, R. Tedrake, S. Karaman, and D. Rus, "Counterexample-Guided Safety Contracts for Autonomous Driving," in *Algorithmic Foundations of Robotics XIII*, M. Morales, L. Tapia, G. Sánchez-Ante, and S. Hutchinson, Eds., Cham: Springer International Publishing, 2020, pp. 939–955, ISBN: 978-3-030-44051-0. DOI: 10.1007/978-3-030-44051-0_54.

[36] C. Pek and M. Althoff, "Fail-Safe Motion Planning for Online Verification of Autonomous Vehicles Using Convex Optimization," *IEEE Transactions on Robotics*, vol. 37, no. 3, pp. 798–814, Jun. 2021, ISSN: 1941-0468. DOI: 10.1109/TRO.2020.3036624. (visited on 05/10/2024).

[37] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, Aug. 2014, ISSN: 0278-3649. DOI: 10.1177/0278364914528132.

[38] D. Liberzon, *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton ; Oxford: Princeton University Press, Jan. 2012, ISBN: 978-0-691-15187-8.

[39] F. Dellaert, "Factor Graphs: Exploiting Structure in Robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 141–166, 2021. DOI: 10.1146/annurev-control-061520-010504. (visited on 06/07/2023).

[40] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, and M. J. Kochenderfer, "Algorithms for Verifying Deep Neural Networks," *Foundations and Trends® in Optimization*, vol. 4, no. 3-4, pp. 244–404, Feb. 2021, ISSN: 2167-3888, 2167-3918. DOI: 10.1561/2400000035. (visited on 09/15/2023).

[41] E. Heiden, D. Millard, E. Coumans, Y. Sheng, and G. S. Sukhatme, "NeuralSim: Augmenting differentiable simulators with neural networks," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[42] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, Curran Associates, Inc., 2018.

[43] Y.-L. Qiao, J. Liang, V. Koltun, and M. Lin, "Differentiable Simulation of Soft Multibody Systems," in *Advances in Neural Information Processing Systems*, Nov. 2021. (visited on 03/10/2024).

[44] Y.-L. Qiao, J. Liang, V. Koltun, and M. C. Lin, *Efficient Differentiable Simulation of Articulated Bodies*, Sep. 2021. DOI: 10.48550/arXiv.2109.07719. arXiv: 2109.07719 `[cs]`. (visited on 03/10/2024).

[45] S. Chen, Y. Xu, C. Yu, L. Li, X. Ma, Z. Xu, and D. Hsu, "DaxBench: Benchmarking Deformable Object Manipulation with Differentiable Physics," in *The Eleventh International Conference on Learning Representations*, Feb. 2023. (visited on 06/12/2023).

[46] J. H. Lee, M. Y. Michelis, R. Katzschmann, and Z. Manchester, *Aquarium: A Fully Differentiable Fluid-Structure Interaction Solver for Robotics Applications*, Mar. 2023. DOI: 10.48550/arXiv.2301.07028. arXiv: 2301.07028 `[cs]`. (visited on 03/10/2024).

[47] S. Zhao, W. Jakob, and T.-M. Li, "Physics-based differentiable rendering: From theory to implementation," in *ACM SIGGRAPH 2020 Courses*, ser. SIGGRAPH '20, New York, NY, USA: Association for Computing Machinery, Aug. 2020, pp. 1–30, ISBN: 978-1-4503-7972-4. DOI: 10.1145/3388769.3407454. (visited on 07/06/2023).

[48] W. Jakob, S. Speierer, N. Roussel, and D. Vicini, "Dr.Jit: A just-in-time compiler for differentiable rendering," *Transactions on Graphics (Proceedings of SIGGRAPH)*, vol. 41, no. 4, Jul. 2022. DOI: 10.1145/3528223.3530099.

[49] Q. Le Lidec, I. Laptev, C. Schmid, and J. Carpentier, "Differentiable rendering with perturbed optimizers," in *Advances in Neural Information Processing Systems*, vol. 34, Curran Associates, Inc., 2021, pp. 20 398–20 409. (visited on 03/15/2023).

[50] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. Carr, J. Ragan-Kelley, and F. Durand, "DiffTaichi: Differentiable Programming for Physical Simulation," in *International Conference on Learning Representations*, Dec. 2019. (visited on 05/15/2023).

[51] P. Kidger, *On Neural Differential Equations*, Feb. 2022. DOI: 10.48550/arXiv.2202.02435. arXiv: 2202.02435 `[cs, math, stat]`. (visited on 02/20/2024).

[52] J. K. Murthy, M. Macklin, F. Golemo, *et al.*, "gradSim: Differentiable simulation for system identification and visuomotor control," in *International Conference on Learning Representations*, Jan. 2021. (visited on 05/15/2023).

[53] A. Schulz, C. Sung, A. Spielberg, W. Zhao, R. Cheng, E. Grinspun, D. Rus, and W. Matusik, "Interactive robogami: An end-to-end system for design of robots with ground locomotion," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1131–1147, 2017.

[54] T. Du, A. Schulz, B. Zhu, B. Bickel, and W. Matusik, "Computational multicopter design," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, p. 227, 2016.

[55] F. Chen and M. Y. Wang, "Design optimization of soft robots: A review of the state of the art," *IEEE Robotics Automation Magazine*, vol. 27, no. 4, pp. 27–43, 2020. DOI: 10.1109/MRA.2020.3024280.

[56] D. Matthews, A. Spielberg, D. Rus, S. Kriegman, and J. Bongard, "Efficient automatic design of robots," *Proceedings of the National Academy of Sciences*, vol. 120, no. 41, e2305180120, Oct. 2023. DOI: 10.1073/pnas.2305180120. (visited on 03/10/2024).

[57] T. Du, J. Hughes, S. Wah, W. Matusik, and D. Rus, "Underwater soft robot modeling and control with differentiable simulation," *IEEE Robotics and Automation Letters*, 2021.

[58] H. J. Suh, M. Simchowitz, K. Zhang, and R. Tedrake, "Do Differentiable Simulators Give Better Policy Gradients?" In *Proceedings of the 39th International Conference on Machine Learning*, PMLR, Jun. 2022, pp. 20 668–20 696. (visited on 06/07/2023).

[59] E. Heiden, C. E. Denniston, D. Millard, F. Ramos, and G. S. Sukhatme, "Probabilistic Inference of Simulation Parameters via Parallel Differentiable Simulation," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3638–3645, Sep. 2021, ISSN: 10504729. DOI: 10.48550/arxiv.2109.08815. arXiv: 2109.08815. (visited on 09/26/2022).

[60] L. Metz, C. D. Freeman, S. S. Schoenholz, and T. Kachman, *Gradients are Not All You Need*, Jan. 2022. DOI: 10.48550/arXiv.2111.05803. arXiv: 2111.05803 `[cs, stat]`. (visited on 05/15/2023).

[61] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986, ISSN: 1476-4687. DOI: 10.1038/323533a0. (visited on 06/07/2023).

[62] A. Paszke, S. Gross, F. Massa, *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035.

[63] M. Abadi, A. Agarwal, P. Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015.

[64] J. Bradbury, R. Frostig, P. Hawkins, *et al.*, *JAX: Composable transformations of Python+NumPy programs*, 2018.

[65] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable Convex Optimization Layers," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019. (visited on 05/15/2023).

[66] K. Leung, N. Arechiga, and M. Pavone, "Back-Propagation Through Signal Temporal Logic Specifications: Infusing Logical Structure into Gradient-Based Methods," in *Algorithmic Foundations of Robotics XIV*, S. M. LaValle, M. Lin, T. Ojala, D. Shell, and J. Yu, Eds., ser. Springer Proceedings in Advanced Robotics, Cham: Springer International Publishing, 2021, pp. 432–449, ISBN: 978-3-030-66723-8. DOI: 10.1007/978-3-030-66723-8_26.

[67] W. Moses and V. Churavy, "Instead of rewriting foreign code for machine learning, automatically synthesize fast gradients," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 12 472–12 485.

[68] *The Autodiff Cookbook — JAX documentation.* (visited on 06/07/2023).

[69] H. Suh, T. Pang, and R. Tedrake, "Bundled gradients through contact via randomized smoothing," *ArXiv*, vol. abs/2109.05143, 2021.

[70] B. Amos and J. Z. Kolter, "OptNet: Differentiable optimization as a layer in neural networks," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, Sydney, NSW, Australia: JMLR.org, Aug. 2017, pp. 136–145. (visited on 01/04/2023).

[71] F. Wood, J. W. Meent, and V. Mansinghka, "A New Approach to Probabilistic Programming Inference," in *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, PMLR, Apr. 2014, pp. 1024–1032. (visited on 06/07/2023).

[72] Y. A. Ma, Y. Chen, C. Jin, N. Flammarion, and M. I. Jordan, "Sampling can be faster than optimization," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 116, no. 42, pp. 20 881–20 885, Oct. 2019, ISSN: 10916490. DOI: 10.1073/PNAS.1820003116/-/DCSUPPLEMENTAL. arXiv: 1811.08413. (visited on 03/17/2022).

[73] S. Levine, *Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review*, May 2018. DOI: 10.48550/arXiv.1805.00909. arXiv: 1805.00909 [cs, stat]. (visited on 06/06/2023).

[74] M. F. Cusumano-Towner, F. A. Saad, A. K. Lew, and V. K. Mansinghka, "Gen: A general-purpose probabilistic programming system with programmable inference," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2019, New York, NY, USA: Association for Computing Machinery, Jun. 2019, pp. 221–236, ISBN: 978-1-4503-6712-7. DOI: 10.1145/3314221.3314642. (visited on 01/04/2024).

[75] R. Tedrake and t. D. D. Team, *Drake: Model-based design and verification for robotics*, 2019.

[76] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[77] J. Zhang, J. Liu, C. Wang, Y. Song, and B. Li, "Study on multidisciplinary design optimization of a 2-degree-of-freedom robot based on sensitivity analysis and structural analysis," *Advances in Mechanical Engineering*, vol. 9, no. 4, p. 1 687 814 017 696 656, 2017. DOI: 10.1177/1687814017696656. eprint: https://doi.org/10.1177/1687814017696656.

[78] J. Xu, T. Du, M. Foshey, B. Li, B. Zhu, A. Schulz, and W. Matusik, "Learning to fly: Computational controller design for hybrid UAVs with reinforcement learning," *ACM Trans. Graph.*, vol. 38, no. 4, Jul. 2019, ISSN: 0730-0301. DOI: 10.1145/3306346.3322940.

[79] P. D. Sharpe, "AeroSandbox: A differentiable framework for aircraft design optimization," M.S. thesis, MIT, 2021.

[80] D. Cascaval, M. Shalah, P. Quinn, R. Bodik, M. Agrawala, and A. Schulz, "Differentiable 3D CAD programs for bidirectional editing," *arXiv*, vol. abs/2110.01182, 2021.

[81] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (2nd Edition)*. Dec. 2002, ISBN: 0-13-790395-2.

[82] D. Amodei, C. Olah, J. Steinhardt, P. F. Christiano, J. Schulman, and D. Mané, "Concrete problems in AI safety," *ArXiv*, vol. abs/1606.06565, 2016.

[83] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "SciPy 1.0: Fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

[84] K. Sridhar, O. Sokolsky, I. Lee, and J. Weimer, "Improving neural network robustness via persistency of excitation," *arXiv*, 2021. arXiv: 2106.02078 `[stat.ML]`.

[85] G. Wood and B. Zhang, "Estimation of the Lipschitz constant of a function," *Journal of Global Optimization*, vol. 8, no. 1, 1996. DOI: 10.1007/bf00229304.

[86] S. Coles, *An Introduction to Statistical Modeling of Extreme Values*. London: Springer, 2001.

[87] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck, "Probabilistic programming in python using PyMC3," *PeerJ Computer Science*, vol. 2, 2016. DOI: 10.7717/peerj-cs.55.

[88] C. Knuth, G. Chou, N. Ozay, and D. Berenson, "Planning with Learned Dynamics: Probabilistic Guarantees on Safety and Reachability via Lipschitz Constants," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5129–5136, Jul. 2021, ISSN: 23773766. DOI: 10.1109/LRA.2021.3068889. arXiv: 2010.08993. (visited on 02/22/2022).

[89] S. Boucheron, G. Lugosi, and P. Massart, *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford: Oxford University Press, 2016.

[90] S. G. Bobkov and C. Houdré, "Variance of lipschitz functions and an isoperimetric problem for a class of product measures," *Bernoulli. Official Journal of the Bernoulli Society for Mathematical Statistics and Probability*, vol. 2, no. 3, pp. 249–255, 1996, ISSN: 13507265.

[91] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2011, pp. 3400–3407.

[92] *Kolmogorov-smirnov goodness-of-fit test.*

[93] C. Dawson and C. Fan, "Robust Counterexample-guided Optimization for Planning from Differentiable Temporal Logic," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2022, pp. 7205–7212. DOI: 10.1109/IROS47612.2022.9981382.

[94] A. Donzé, "Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6174 LNCS, pp. 167–170, 2010, ISSN: 03029743. DOI: 10.1007/978-3-642-14295-6_17. (visited on 02/18/2022).

[95] D. Sun, J. Chen, S. Mitra, and C. Fan, "Multi-agent Motion Planning from Signal Temporal Logic Specifications," *IEEE Robotics and Automation Letters (RA-L)*, Jan. 2022. arXiv: 2201.05247. (visited on 01/30/2022).

[96] Y. V. Pant, H. Abbas, and R. Mangharam, "Smooth Operator: Control using the Smooth Robustness of Temporal Logic," *IEEE Conference on Control Technology and Applications, 2017*, Aug. 2017. (visited on 02/28/2022).

[97] E. Plaku and S. Karaman, "Motion planning with temporal-logic specifications: Progress and challenges," *AI Communications*, vol. 29, no. 1, pp. 151–162, Jan. 2016, ISSN: 0921-7126. DOI: 10.3233/AIC-150682. (visited on 02/07/2022).

[98] R. Takano, H. Oyama, and M. Yamakita, "Continuous Optimization-Based Task and Motion Planning with Signal Temporal Logic Specifications for Sequential Manipulation," in *Proceedings of the 40th IEEE Conference on Decision and Control*, Institute of Electrical and Electronics Engineers (IEEE), Oct. 2021, pp. 8409–8415. DOI: 10.1109/ICRA48506.2021.9561209. (visited on 02/03/2022).

[99] Y. Chen, R. Gandhi, Y. Zhang, and C. Fan, "NL2TL: Transforming Natural Languages to Temporal Logics using Large Language Models," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds., Singapore: Association for Computational Linguistics, Dec. 2023, pp. 15 880–15 903. DOI: 10.18653/v1/2023.emnlp-main.985. (visited on 02/24/2024).

[100] L. Yang and N. Ozay, "Synthesis-guided Adversarial Scenario Generation for Gray-box Feedback Control Systems with Sensing Imperfections," *ACM Transactions on Embedded Computing Systems*, vol. 20, no. 5s, 102:1–102:25, Sep. 2021, ISSN: 1539-9087. DOI: 10.1145/3477033. (visited on 03/14/2023).

[101] Y. Kantaros and M. M. Zavlanos, "STyLuS*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems," *The International Journal of Robotics Research*, vol. 39, no. 7, pp. 812–836, 2020. DOI: 10.1177/0278364920913922. eprint: https://doi.org/10.1177/0278364920913922.

[102] C.-I. Vasile, V. Raman, and S. Karaman, "Sampling-based synthesis of maximally-satisfying controllers for temporal logic specifications," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 3840–3847. DOI: 10.1109/IROS.2017.8206235.

[103] Y. V. Pant, H. Abbas, R. A. Quaye, and R. Mangharam, "Fly-by-Logic: Control of Multi-Drone Fleets with Temporal Logic Objectives," *Proceedings - 9th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2018*, pp. 186–197, Aug. 2018. DOI: 10.1109/ICCPS.2018.00026. (visited on 02/14/2022).

[104] A. Pantazides, D. Aksaray, and D. Gebre-egziabher, "Satellite Mission Planning with Signal Temporal Logic Specifications," Jan. 2022. DOI: 10.2514/6.2022-1091. (visited on 02/14/2022).

[105] S. Sadraddini and C. Belta, "Robust temporal logic model predictive control," *2015 53rd Annual Allerton Conference on Communication, Control, and Computing, Allerton 2015*, pp. 772–779, Apr. 2016. DOI: 10.1109/ALLERTON.2015.7447084. arXiv: 1511.00347. (visited on 03/01/2022).

[106] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '15, New York, NY, USA: Association for Computing Machinery, 2015, pp. 239–248, ISBN: 978-1-4503-3433-4. DOI: 10.1145/2728606.2728628.

[107] K. M. B. Lee, C. Yoo, and R. Fitch, "Signal Temporal Logic Synthesis as Probabilistic Inference," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2021-May, pp. 5483–5489, 2021, ISSN: 10504729. DOI: 10.1109/ICRA48506.2021.9560929. arXiv: 2105.06121. (visited on 07/17/2022).

[108] A. Donzé, T. Ferrère, and O. Maler, "Efficient Robust Monitoring for STL," in *Computer Aided Verification*, N. Sharygina and H. Veith, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2013, pp. 264–279, ISBN: 978-3-642-39799-8. DOI: 10.1007/978-3-642-39799-8_19.

[109] Y.-C. Chang, N. Roohi, and S. Gao, "Neural Lyapunov Control," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019. (visited on 06/05/2023).

[110] C. Jewison and R. S. Erwin, "A spacecraft benchmark problem for hybrid control and estimation," *2016 IEEE 55th Conference on Decision and Control, CDC 2016*, pp. 3300–3305, Dec. 2016. DOI: 10.1109/CDC.2016.7798765. (visited on 02/15/2022).

[111] C. Dawson and C. Fan, "A Bayesian approach to breaking things: Efficiently predicting and repairing failure modes via sampling," in *7th Annual Conference on Robot Learning*, Aug. 2023. (visited on 09/21/2023).

[112] C. Geyer, *Introduction to Markov Chain Monte Carlo*. Chapman and Hall/CRC, May 2011, pp. 29–74, ISBN: 978-0-429-13850-8. DOI: 10.1201/b10905-6. (visited on 12/16/2022).

[113] G. O. Roberts and O. Stramer, "Langevin Diffusions and Metropolis-Hastings Algorithms," *Methodology And Computing In Applied Probability*, vol. 4, no. 4, pp. 337–357, Dec. 2002, ISSN: 1573-7713. DOI: 10.1023/A:1023562417138. (visited on 01/05/2023).

[114] N. Chopin and O. Papaspiliopoulos, *An Introduction to Sequential Monte Carlo*. Cham: Springer International Publishing, 2020, ISBN: 978-3-030-47844-5. (visited on 10/17/2022).

[115] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-Based Generative Modeling through Stochastic Differential Equations," in *International Conference on Learning Representations*, Jan. 2023. (visited on 03/23/2023).

[116] G. Rubino and B. Tuffin, "Introduction to Rare Event Simulation," in *Rare Event Simulation Using Monte Carlo Methods*, John Wiley & Sons, Ltd, 2009, ch. 1, pp. 1–13, ISBN: 978-0-470-74540-3. DOI: 10.1002/9780470745403.ch1. (visited on 03/14/2023).

[117] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, Apr. 1970, ISSN: 0006-3444. DOI: 10.1093/biomet/57.1.97. (visited on 09/22/2023).

[118] R. M. Neal, "MCMC Using Hamiltonian Dynamics," in *Handbook of Markov Chain Monte Carlo*, Chapman and Hall/CRC, 2011, ISBN: 978-0-429-13850-8.

[119] J. Bresag, "Comments on U. Grenadier, M. Miller, "Representations of Knowledge in Complex Systems"," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 56, no. 4, pp. 549–603, 1994, ISSN: 0035-9246. (visited on 05/15/2023).

[120] Q. Liu and D. Wang, "Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm," in *Advances in Neural Information Processing Systems*, vol. 29, Curran Associates, Inc., 2016. (visited on 09/22/2023).

[121] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods* (Springer Texts in Statistics). New York, NY: Springer, 2004, ISBN: 978-1-4419-1939-7 978-1-4757-4145-2. DOI: 10.1007/978-1-4757-4145-2. (visited on 09/25/2023).

[122] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal Policy Optimization Algorithms*, Aug. 2017. DOI: 10.48550/arXiv.1707.06347. arXiv: `1707.06347 [cs]`. (visited on 05/17/2023).

[123] F. Capitanescu, J. L. Martinez Ramos, P. Panciatici, D. Kirschen, A. Marano Marcolini, L. Platbrood, and L. Wehenkel, "State-of-the-art, challenges, and future trends in security constrained optimal power flow," *Electric Power Systems Research*, vol. 81, no. 8, pp. 1731–1741, Aug. 2011, ISSN: 0378-7796. DOI: 10.1016/j.epsr.2011.04.003. (visited on 12/15/2022).

[124] M. B. Cain, R. P. O'Neill, and A. Castillo, "History of Optimal Power Flow and Formulations," Federal Energy Regulatory Commission, Tech. Rep., 2012.

[125] U.S. Department of Energy, *Grid Optimization Competition*. (visited on 12/15/2022).

[126] Illinois Center for a Smarter Electric Grid, *IEEE 57-Bus System*. (visited on 06/07/2023).

[127] S. Huang, K. Filonenko, and C. T. Veje, "A Review of The Convexification Methods for AC Optimal Power Flow," in *2019 IEEE Electrical Power and Energy Conference (EPEC)*, Oct. 2019, pp. 1–6. DOI: 10.1109/EPEC47565.2019.9074824. (visited on 03/09/2024).

[128] P. L. Donti, D. Rolnick, and J. Z. Kolter, *DC3: A learning method for optimization with hard constraints*, Apr. 2021. DOI: 10.48550/arXiv.2104.12225. arXiv: 2104.12225 [cs, math, stat]. (visited on 12/17/2022).

[129] S. Wilson, P. Glotfelter, L. Wang, S. Mayya, G. Notomista, M. Mote, and M. Egerstedt, "The Robotarium: Globally Impactful Opportunities, Challenges, and Lessons Learned in Remote-Access, Distributed Control of Multirobot Systems," *IEEE Control Systems Magazine*, vol. 40, no. 1, pp. 26–44, Feb. 2020, ISSN: 1941-000X. DOI: 10.1109/MCS.2019.2949973.

[130] A. M. Stuart, "Inverse problems: A Bayesian perspective," *Acta Numerica*, vol. 19, pp. 451–559, May 2010, ISSN: 1474-0508, 0962-4929. DOI: 10.1017/S0962492910000061. (visited on 01/02/2024).

[131] R. Molinaro, Y. Yang, B. Engquist, and S. Mishra, "Neural Inverse Operators for Solving PDE Inverse Problems," in *Proceedings of the 40th International Conference on Machine Learning*, PMLR, Jul. 2023, pp. 25 105–25 139. (visited on 01/03/2024).

[132] T. Liu, T. Yang, Q. Zhang, and Q. Lei, "Optimization for Amortized Inverse Problems," in *Proceedings of the 40th International Conference on Machine Learning*, PMLR, Jul. 2023, pp. 22 289–22 319. (visited on 01/03/2024).

[133] M. Asim, M. Daniels, O. Leong, A. Ahmed, and P. Hand, "Invertible generative models for inverse problems: Mitigating representation error and dataset bias," in *Proceedings of the 37th International Conference on Machine Learning*, PMLR, Nov. 2020, pp. 399–409. (visited on 01/04/2024).

[134] S. Liang, Y. Li, and R. Srikant, *Enhancing The Reliability of Out-of-distribution Image Detection in Neural Networks*, Aug. 2020. DOI: 10.48550/arXiv.1706.02690. arXiv: 1706.02690 [cs, stat]. (visited on 01/30/2024).

[135] D. Hendrycks, M. Mazeika, and T. Dietterich, "Deep Anomaly Detection with Outlier Exposure," in *International Conference on Learning Representations*, Sep. 2018. (visited on 01/30/2024).

[136] P. Kirichenko, P. Izmailov, and A. G. Wilson, "Why Normalizing Flows Fail to Detect Out-of-Distribution Data," in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 20 578–20 589. (visited on 01/30/2024).

[137] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a Few Examples: A Survey on Few-shot Learning," *ACM Computing Surveys*, vol. 53, no. 3, 63:1–63:34, Jun. 2020, ISSN: 0360-0300. DOI: 10.1145/3386252. (visited on 01/30/2024).

[138] J. Rose, "Southwest will pay a $140 million fine for its meltdown during the 2022 holidays," *NPR*, Dec. 2023. (visited on 01/03/2024).

[139] D. P. Kingma and M. Welling, *Auto-Encoding Variational Bayes*, May 2014. DOI: 10.48550/arXiv.1312.6114. arXiv: 1312.6114 [cs, stat]. (visited on 01/04/2024).

[140] E. G. Tabak and E. Vanden-Eijnden, "Density estimation by dual ascent of the log-likelihood," *Communications in Mathematical Sciences*, vol. 8, no. 1, pp. 217–233, Mar. 2010, ISSN: 1539-6746, 1945-0796. (visited on 01/21/2024).

[141] D. J. Rezende and S. Mohamed, “Variational inference with normalizing flows,” in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML’15, Lille, France: JMLR.org, Jul. 2015, pp. 1530–1538. (visited on 01/06/2024).

[142] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, “Normalizing flows for probabilistic modeling and inference,” *The Journal of Machine Learning Research*, vol. 22, no. 1, 57:2617–57:2680, Jan. 2021, ISSN: 1532-4435.

[143] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, “FFJORD: Free-Form Continuous Dynamics for Scalable Reversible Generative Models,” in *International Conference on Learning Representations*, Sep. 2018. (visited on 01/21/2024).

[144] D. Onken, S. W. Fung, X. Li, and L. Ruthotto, “OT-Flow: Fast and Accurate Continuous Normalizing Flows via Optimal Transport,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, pp. 9223–9232, May 2021, ISSN: 2374-3468. DOI: 10.1609/aaai.v35i10.17113. (visited on 01/26/2024).

[145] C.-W. Huang, D. Krueger, A. Lacoste, and A. Courville, “Neural Autoregressive Flows,” in *Proceedings of the 35th International Conference on Machine Learning*, PMLR, Jul. 2018, pp. 2078–2087. (visited on 01/21/2024).

[146] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, “Neural spline flows,” in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 675, Red Hook, NY, USA: Curran Associates Inc., Dec. 2019, pp. 7511–7522. (visited on 01/12/2024).

[147] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, “Beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework,” in *International Conference on Learning Representations*, Nov. 2016. (visited on 01/08/2024).

[148] M. A. Fischler and R. C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981, ISSN: 0001-0782. DOI: 10.1145/358669.358692. (visited on 01/30/2024).

[149] A. Keipour, M. Mousaei, and S. Scherer, “ALFA: A dataset for UAV fault and anomaly detection,” *The International Journal of Robotics Research*, vol. 40, no. 2-3, pp. 515–520, Feb. 2021, ISSN: 0278-3649. DOI: 10.1177/0278364920966642. (visited on 01/16/2024).

[150] C. Deng, S. Feng, H. Wang, X. Zhang, P. Jin, Y. Feng, Q. Zeng, Y. Chen, and Y. Lin, “OpenFWI: Large-scale Multi-structural Benchmark Datasets for Full Waveform Inversion,” in *Thirty-Sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, Jun. 2022. (visited on 01/17/2024).

[151] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman, “Pyro: Deep universal probabilistic programming,” *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 973–978, Jan. 2019, ISSN: 1532-4435.

[152] Bureau of Transportation Statistics, *TranStats. U.S. Department of Transportation.*

[153] N. Pyrgiotis, K. M. Malone, and A. Odoni, "Modelling delay propagation within an airport network," *Transportation Research Part C: Emerging Technologies*, Selected Papers from the Seventh Triennial Symposium on Transportation Analysis (TRISTAN VII), vol. 27, pp. 60–75, Feb. 2013, ISSN: 0968-090X. DOI: 10.1016/j.trc.2011.05.017. (visited on 01/20/2024).

[154] W. P. Gouveia and J. A. Scales, "Bayesian seismic waveform inversion: Parameter estimation and uncertainty analysis," *Journal of Geophysical Research: Solid Earth*, vol. 103, no. B2, pp. 2759–2779, 1998, ISSN: 2156-2202. DOI: 10.1029/97JB02933. (visited on 01/19/2024).

[155] R. Zhang, C. Czado, and K. Sigloch, "Bayesian Spatial Modelling for High Dimensional Seismic Inverse Problems," *Journal of the Royal Statistical Society Series C: Applied Statistics*, vol. 65, no. 2, pp. 187–213, Feb. 2016, ISSN: 0035-9254. DOI: 10.1111/rssc.12118. (visited on 01/19/2024).

[156] A. Richardson, *Deepwave*, Zenodo, Sep. 2023. DOI: 10.5281/zenodo.8381177. (visited on 01/17/2024).

[157] C. Finlay, J.-H. Jacobsen, L. Nurbekyan, and A. M. Oberman, "How to train your neural ODE: The world of Jacobian and Kinetic regularization," in *Proceedings of the 37th International Conference on Machine Learning*, ser. ICML'20, vol. 119, JMLR.org, Jul. 2020, pp. 3154–3164. (visited on 01/25/2024).

[158] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural Ordinary Differential Equations," in *Advances in Neural Information Processing Systems*, vol. 31, Curran Associates, Inc., 2018. (visited on 01/21/2024).

[159] *Probabilists/Zuko*, The Probabilists, Jan. 2024. (visited on 02/01/2024).

[160] Southwest Airlines, *Final Summary and Action Plan*, 2023.

[161] C. Murray, *Strengthening airline operations and consumer protections*, Feb. 2023.

[162] M. Cramer and M. Levenson, "What Caused the Chaos at Southwest," *The New York Times*, Dec. 2022, ISSN: 0362-4331. (visited on 01/26/2024).

[163] A. Dembo and O. Zeitouni, *Large Deviations Techniques and Applications* (Stochastic Modelling and Applied Probability). Berlin, Heidelberg: Springer, 2010, vol. 38, ISBN: 978-3-642-03310-0 978-3-642-03311-7. DOI: 10.1007/978-3-642-03311-7. (visited on 02/01/2024).

[164] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education," *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 12–19, Feb. 2011, ISSN: 1558-0679. DOI: 10.1109/TPWRS.2010.2051168.

Image acknowledgment

Figs. 5.5 and 5.9 was created using assets designed by Macrovector/Freepix; used with permission.