# Safety in an uncertain world: model-based verification-guided design for safety-critcial autonomous systems

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
Associate Professor of Aeronautics and Astronautics, Thesis Supervisor

Accepted by: Jonathan P. How
Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee, Department of Aeronautics and Astronautics

# Safety in an uncertain world: model-based verification-guided design for safety-critcial autonomous systems

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 robots can be deployed in safety-critical environments, we must be able to verify they they will perform safely, ideally without the risk or expense of real-world testing. A wide variety of formal methods and simulation-driven techniques have been developed to conduct this verification, but they typically rely on difficult-to-construct mathematical models or else use sample-inneficient black-box optimization methods. In this thesis, I propose to develop a suite of tools that use program analysis tools like automatic differentiation to automatically construct mathematical models of the system under test and accelerate verification of robots and other autonomous systems. These tools rely on two technical innovations: first, the use of general-purpose automatic differentiation and probabilistic programming methods to introspect simulators of complex autonomous systems, and second: reframing the verification problem as a Bayesian inference problem (rather than an optimization problem) to make use of high-performance gradient-based inference algorithms. In addition to these technical innovations to solve verification problems, my thesis will also contribute a novel capability in the form of verification-guided design. Existing verification methods provide little insight to system designers about how to improve their systems to make them safer. In my thesis, I propose a novel adversarial inference algorithm to close the loop between verification and design, allowing the system designer to automatically generate and preemtively repair adversarial test cases to improve the safety of the system under test.

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

# Acknowledgments

TODO Write your acknowledgments here.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction & Objectives

Before robots can be deployed in safety-critical environments, we must be able to verify they they will perform safely. In cases where real-world testing is too risky or expensive, robotics engineers must instead rely on mathematical modeling or 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.

In my thesis, I aim to close the gap between formal methods based on mathematical models [1], [2] and simulation-based verification techniques [3]–[5] to develop tools to help engineers more easily *design* and *verify* complex robotic systems. My key insight is that simulators, as computer programs, are not black boxes but instead contain rich mathematical structure. If we can exploit this structure using program analysis tools (e.g. automatic differentiation, tracing, etc.), we can combine the flexibility of simulation with the rigor of mathematical reasoning. Using this insight, I aim to develop tools to support the *design-analysis cycle* for robots and other safety-critical cyberphysical systems in three 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 (either by 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.

These tools 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. Moreover, I will apply my research not just to traditional robotic systems, but to cyberphysical systems such as large-scale energy, transportation, and industrial networks.

### 1.0.1 Impact

Recent years have seen large numbers of learning-enabled autonomous systems deployed in the real world. Unfortunately, increased deployment has seen a corresponding increase in accidents involving these systems. More and more, the development and deployment of autonomous systems risks being stalled due to public and regulatory concern over safety [6]. Unlike other forms of safety-critical infrastructure like bridges and buildings, autonomous systems suffer from both *complexity*, leading to difficult-to-predict emergent behavior resulting from interactions between subsystems, and *uncertainty* stemming from the environment or the actions of other agents. While engineers in other fields have access to a range of computer-aided tools to boost their productivity and ensure the safety of their designs, such as computer-aided design (CAD) and finite-element analysis (FEA) in mechanical and civil engineering or electronic design automation (EDA) in electrical engineering, robotics engineers lack such comprehensive design and verification tools. In the abscence of these tools, engineers must rely on an ad-hoc process relying heavily on experience and tedious parameter tuning. In my thesis, I aim to address this gap by developing efficient computational tools to assist engineers in designing and verifying safety-critical autonomous systems, with the goal of helping engineers more confidently *design*, *debug*, and *deploy* autonomous systems.

### 1.0.2 Outline

# Chapter 2

# Background and Significance

My thesis aims to build on prior work on safety verification (both model-based and black-box) and program analysis (in the form of differentiable and probabilistic programming). This section will review each of these fields with an eye towards framing the significance of my planned thesis contributions.

### 2.0.1 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, and instead use 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 [4]: falsification, likely failure analysis, or failure probability estimation. 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 $\xi$, 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 counterex-

ample

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

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

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

Finally, the failure probability estimation problem aims to estimate the probability that the system violates the safety constraints

$$\Pr\left[\psi(f(y)) < 0\right] = \mathbb{E}_{y \sim p(y)}\left[\mathbb{1}\{\psi(f(y))\}\right] \tag{2.4}$$

### Model-based safety verification

Early approaches to model-based verification and fault identification used symbolic logical models of the system under test to formally reason about failures using (computationally expensive) satisfiability (SAT) solvers or search [7], [8]. More recent approaches to model-based failure mode identification have used mathematical models of the system dynamics to frame the problem through the lens of reachability [9], [10], optimal control [11], or optimization (e.g. sum-of-squares, or SoS [12], [13]).

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. Historically, this difficulty motivated the development of model-free approaches to safety verification.

### 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 characterized by restricting the verification algorithm to sampling input-output pairs from the simulator without side information such as gradients [4]. 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 [14]), they have seen

widespread use, particularly in verification for autonomous vehicles [5], [15]–[23]. The three most common types of black-box verification method involve black-box optimization, reinforcement learning, or black-box inference methods [4].

**Black-box optimization** uses 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 [18], REINFORCE [24], and ant colony optimization [9]. There are a large number of subtly-different black-box optimization schemes that could be applied to this problem [25], all making different tradeoffs between exploration and exploitation.

**Reinforcement learning** break 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 [17]. 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 [24].

**Black-box inference** takes inspiration from algorithms for approximate Bayesian inference and are most often applied to the failure probability estimation [5] and likely failure mode problems [3]. To estimate failure probability, [5] 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, [3] uses gradient-free Markov chain Monte Carlo (MCMC) to sample failure modes that are likely to induce a failure.

### Limitations and significance

In the context of prior work on model-based and black-box safety verification, this thesis aims to address two major technical gaps.

**Model-based approaches are difficult to scale to complex systems** The requirement of a symbolic model is either theoretically impossible (when closed-form models do not exist) or practically infeasible (for large-scale systems with multiple interacting subsystems). The difficulty of constructing symbolic models by hand motivates my work in this thesis, which seeks to automatically extract information about the mathematical structure of a system from the simulator using program analysis techniques (discussed in the next section).

**Black-box methods struggle with sample complexity** Lacking access to gradient information that could help guide their exploration of high-dimensional spaces, black-box methods often struggle to scale to high-dimensional problems, or (if they can scale) require a large number of simulated rollouts to converge. Although black-box methods are easy to integrate with existing simulators, these scalability and sample-complexity issues motivate my work in this thesis, which seeks to improve scalability and sample complexity by incorporating additional side information (e.g. automatically-derived gradients) into the optimization process.

### 2.0.2 Programs as mathematical models

Historically, symbolic models have been synonymous with hand derivation, a tedious and error-prone process that does not scale to complex systems. However, recent work in the programming languages community has shown that there is an alternate way to construct these models: by exploiting the rich mathematical structure available embedded in computer programs themselves.

In practice, often the term "black-box" is 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 introspection 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 exciting 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).

#### 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 [26]. The popularity of neural networks prompted the development of differentiable tensor math libraries such as PyTorch [27], TensorFlow [28], and JAX [29], and specialized AD tools followed in the form of differentiable optimization layers [30], simulators [31], renderers [31], [32], and even task specifications [33], to name but a few (the discussion in this document is largely Python-focused, but similar ecosystems exist in Julia, C++, and other LLVM languages [34]).

At a high level, the aim of automatic differentiation is to allow the user to implement some function $y = f(x) : \mathbb{R}^n \mapsto \mathbb{R}^m$ and then provide the ability, without writing any

additional code, to obtain the Jacobian $Df(x) \in \mathbb{R}^{m \times n}$. 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; [35] provides a more thorough introduction.

Forward-mode AD computes the product between a vector in the input tangent space $\delta x$ and the Jacobian, "pushing forward" into the tangent space of the output $\delta x \mapsto \delta y = Df(x)\delta x$. 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 $\delta x$. 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 just evaluating the function value). 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 $\delta y$ and the transposed Jacobian, "pulling back" into the tangent space of the input $\delta y \mapsto \delta x = Df(x)^T \delta y$, and it is referred to as the pullback map or vector-Jacobian product (VJP) accordingly. This mode is typically implemented as generalized backpropagation that tracks which primitive operations are applied in evaluating the output value $y = f(x)$ and constructs a computation graph. 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 in practice 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 function's 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 [30]. This approach allows for accurate differentiation of implicit dynamics (including certain contact models [36]), 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 [37]–[39].

**Significance of proposed thesis** The last few years have seen a surge of applications of AD, including robotics problems such as control synthesis and system identification [30], [40]–[43]. However, despite these successful applications, two important research 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?

### Probabilistic programming

Differentiable programming, although popular, is just the tip of the program analysis iceberg. Another promising emerging field of program analysis is that of probabilistic programming, which extends its view to include programs that make random choices (e.g. by querying a random number generator). Probabilistic programming takes the perspective that the computation graph of such a program can be viewed as a graphical probabilistic model

(e.g. Bayesian network) that encodes a joint distribution over variables involved in the computation [44]. With this mindset, we can gain an additional level of introspection 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 frameworks require additional annotation of the source code, often defining additional syntax to annotate random choices (so that such 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 [45], [46]), but inference also allows us to answer questions such as the likely failure analysis problem in (2.3). In addition, the probabilistic mindset allows us to deal with models that combine continuous and discrete choices in a principled way (as hybrid continuous-discrete graphical models) and design inference algorithms accordingly [47]. Finally, probabilistic programming can also be applied to program synthesis, since the structure of the program's computation graph can itself be seen as the output of a stochastic program-generating process, allowing us to use conditional inference to search over the space of programs that might solve a particular problem [48].

**Significance of proposed thesis** Probabilistic programming has been applied to a large number of problems in statistical inference; in fact, many of its original applications involved automating (with a convenient programming interface) traditional statistical inference tasks such as regression and hierarchical modeling [47]. Although probabilistic graphical models such as factor graphs have been applied widely in robotics, with deep roots in the SLAM community [49], 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 dynamical systems found in traditional "black-box" settings. 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

This chapter is based on the author's published work [50].

To design complex systems, engineers in many fields use computer-aided tools to boost their productivity. Mechanical engineers can 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, and yet highly complex integrated circuits. Sadly, when it comes to designing autonomous systems and robots, engineers often take an ad-hoc approach, relying heavily on experience and tedious parameter tuning.

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 of this trend). 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 will refine this approach to improve its robustness through the use of adversarial and stochastic optimization, and we will improve the coverage and efficiency of the verification process using techniques from statistical inference and rare event simulation.

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 [43], [51]–[55] or subsystems [56]. 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 Related work

### Design optimization for robotics

Most existing works on design optimization for robotics focus on a particular application, such as simple walking robots [51], quadrotors [52], and soft robots [43], [53], [54]. Other works employ optimization to design specific subsystems, such as controllers [56] or motion plans [57]. In contrast, the purpose of this work is to develop a general-purpose robot design optimization tool that can be applied not only to a range of robot design problems but also to optimize the design of multiple subsystems simultaneously. This goal is related to that of a large family of multi-disciplinary design optimization (MDO) methods in aerospace engineering [58]. As discussed above, our approach differs from MDO in its use of differentiable programming as a flexible modeling tool and our novel statistical approach to robustness analysis. We review the related work for automatic differentiation and robustness analysis in the next two sections.

### Programming languages for design optimization

When it comes to managing complexity in a general-purpose design framework, programming languages are a natural tool. They allow users (i.e. programmers) to define precisely which abstractions are appropriate for any given application (e.g. by defining appropriate class hierarchies and function interfaces) without sacrificing generality. To take advantage of this expressivity, we can view engineering designs as programs that define the behavior of the system given suitable choices for design structure and parameters. We can then use automatic differentiation to derive gradients connecting these parameters to the system's behavior and optimize accordingly. This view is inspired by recent work in 3D design optimization [59], aircraft design [60], and machine learning [27], [29].

In recent years, the robotics community has also developed special-purpose differentiable simulators for robotic systems, particularly those involving rigid body contact dynamics [37], [42], [61], [62]. These simulators have been used to solve system identification and controller design tasks, but they do not represent a general-purpose framework, as gradients are often derived by hand and the simulators are not expressive enough to model full-stack robotic systems (e.g. with perception and navigation capabilities). We take inspiration from these methods in our case studies, where we implement a simple differentiable contact simulator in our second case study.

#### Formal methods for robustness analysis

Safety and robustness are critical concerns for any robotic system. When it comes to low-level control, there is a rich history of reachability [63] and stability [64], [65] analysis tools that can be used to answer questions of safety and robustness for the control subsystem. Other works apply reachability analysis at the system level using black-box tools [66]. This work builds on this history by incorporating formal analysis into a design-optimize-analyze loop to provide rapid feedback on robustness as part of the design process. In particular, we develop a novel statistical method for quantifying the worst-case performance and sensitivity of an optimized design to external perturbations.

## 3.2 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 [59], [60]. This focus is common in design optimization (e.g. aircraft design in [60] and 3D CAD optimization in [59]).

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

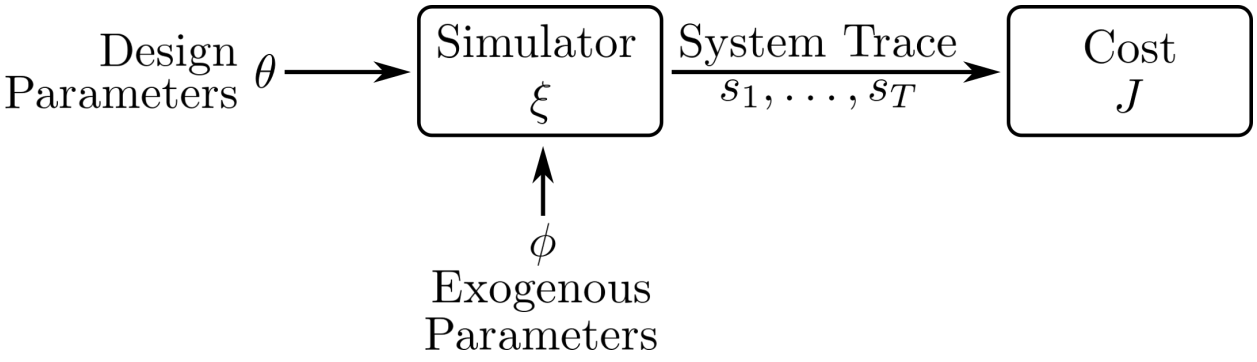



Figure 3.1: A glass-box model of a generic robotic system. Design optimization involves finding a set of design parameters so that the simulated cost is minimized, while robustness analysis involves quantifying how changes in the exogenous parameters affect the simulated cost.

of $\mathbb{R}^m$. We assume no knowledge of $\Phi$ other than the ability to draw samples i.i.d..

**Simulator**

Given particular choices for $\theta$ and $\phi$, the system's state $s \in \mathcal{S}$ evolves in discrete time according to a known simulator $S : \Theta \times \Phi \mapsto \mathcal{S}^T$. 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 $J : \mathcal{S}^T \mapsto \mathbb{R}$ mapping system behaviors (i.e. a trace of states) to a scalar performance metric that we seek to minimize.

**Constraints**

The choice of design parameters is governed by a set of constraints $c_i : \Theta \mapsto \mathbb{R}$ with index set $i \in \mathcal{I}_c$. Design parameters $\theta$ are feasible if $c_i(\theta) \geq 0 \; \forall i \in \mathcal{I}_c$. Here, we consider constraints as functions of $\theta$ only; we leave the extension to robust constraints involving $\phi$ to future work.

We assume that the simulator $S$, cost $J$, and constraints $c_i$ are automatically differentiable almost everywhere with respect to $\theta$ and $\phi$. 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 on the performance of our approach.

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

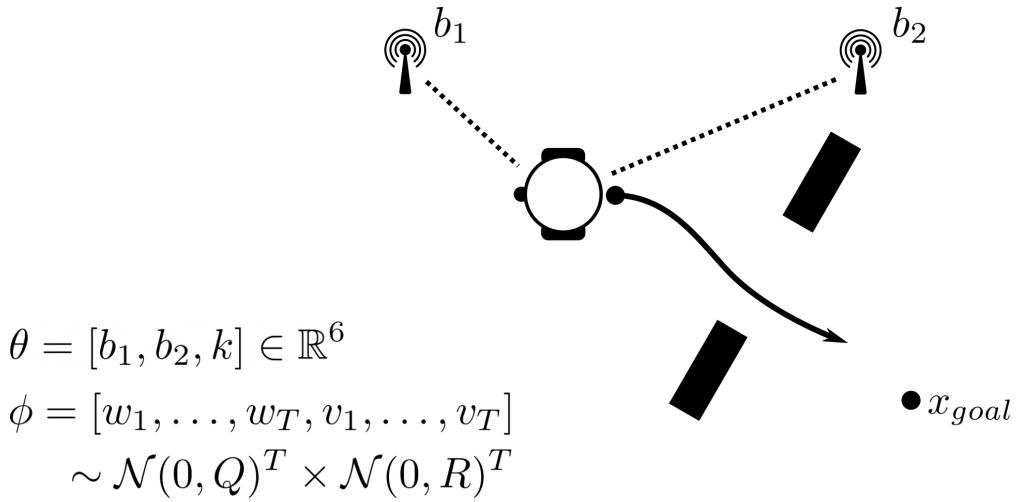



Figure 3.2: A design optimization problem for an AGV localization and navigation system. The goal is to find placements for two range sensors along with parameters for the navigation system that allow the robot to safely pass through the narrow doorway.

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 [67] and feedback controller to track a collision-free path between the obstacles.

In this problem, the design parameters $\theta$ 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 $\phi$ 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 Gaussian). The simulator $\xi$ 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 3.5.1; first, we discuss our approach to design optimization and robustness analysis in Sections 3.3 and 3.4, respectively.

## 3.3 Design optimization using differentiable programming

Given the notation from Section 3.2, 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 $\phi$. 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 $\phi$ compensates for poor performance on other values; this is related to the phenomenon of "reward hacking" in reinforcement learning [68].

Ideally, we would like our designs to be robust to variations in exogenous parameters: changing $\phi$ 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 3.4.3. 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 $\phi_i \sim \Phi, i = 1, \ldots, N$.

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

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

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

Of course, these Monte-Carlo estimators 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 $\theta$, which we can use with any off-the-shelf gradient-based optimization engine. 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 $\theta$, 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 [69], and we plan to add support for hybrid methods combining local gradient descent with gradient-free population methods in a future work.

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 [29], 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 can 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.

## 3.4 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 3.3) 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 from a finite number of simulations to predict worst-case performance. To address this difficulty, we develop a probabilistic approach based on extreme value theory in statistics [70]–[72]. We begin by stating a relevant result:

**Theorem 3.4.1** (Extremal Types Theorem; 3.1.1 in [72]). *Let $X_1, \ldots, X_N$ 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 $\mu$, scale $\sigma$, and shape $\xi$,*

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

*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 $\mu$ and $\sigma$) and the GEVD is fit directly to $M_N$ by either minimizing the log likelihood [72] or estimating the posterior distribution of $(\mu, \sigma, \xi)$ using Markov Chain Monte Carlo sampling [73]. 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 $M_\infty$. If $\xi \geq 0$, then we cannot estimate a strict upper bound, but we can provide for a confidence interval for $M_\infty$ instead. In the following sections, we apply this theorem to analyze the robustness of an optimized design.

### 3.4.1 Estimating the worst-case performance

Our first robustness question concerns the worst-case performance of our design: given variation in $\phi$, what is the maximum cost[1] we can expect for our choice of design parameters $\theta$? 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 3.4.1 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 [72].

**Algorithm 3.4.1:** 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 $\phi_{ij} \sim \Phi$, $1 \leq j \leq N$, $1 \leq i \leq M$
$M_N^i \leftarrow \max\{X_1^i, \ldots, X_N^i\}$ for $i = 1, \ldots, M$
$(\mu, \sigma, \xi) \leftarrow$ posterior GEVD estimate given $\{M_N^i\}$

In practice, we use the automatic parallelization features of JAX to efficiently compute $X_j^i$ and obtain the posterior distribution of $\mu$, $\sigma$, and $\xi$ using Markov Chain Monte Carlo sampling with the PyMC3 library [73]. From this posterior distribution, we take the 97% confidence level for each parameter $(\mu^*, \sigma^*, \xi^*)$. 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 $(\mu^*, \sigma^*, \xi^*)$.

[1] Any function of the simulation trace can be substituted for cost without changing the framework.

### 3.4.2 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 $\phi$), then how much can we expect its performance to degrade if $\phi$ 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 $L_N = \max\{X_1, \ldots, X_N\}$ using a GEVD. Algorithm 3.4.2 provides our method for fitting this distribution, and a concrete Python implementation is provided in the supplementary materials. This approach is similar to that in [70], [74] but removes the assumption that $L$ is bounded by fitting a GEVD instead of a reverse Weibull distribution, allowing our approach to apply when $J \circ S$ is not Lipschitz.

**Algorithm 3.4.2:** 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 $\phi_{ij,1}, \phi_{ij,2} \sim \Phi$, $j = 1, \ldots, N$, $i = 1, \ldots, M$

$L_N^i \leftarrow \max\{X_1^i, \ldots, X_N^i\}$ for $i = 1, \ldots, M$

$(\mu, \sigma, \xi) \leftarrow$ posterior GEVD estimate given $\{L_N^i\}$

Algorithm 3.4.2 is similar to Algorithm 3.4.1, but the interpretation of the results differs in that the fit parameters from Algorithm 3.4.2 allow us to understand the sensitivity of a design. In particular, if the 97% confidence level for the shape parameter $\xi^*$ 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.

### 3.4.3 Connections to Design Optimization

Here, we will attempt to justify the variance regularization heuristic introduced in Section 3.3 with reference to the worst-case performance $J_{max}$ and sensitivity $L$ computed by Algorithms 3.4.1 and 3.4.2. First, let's 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 $\alpha$ of $J_{max}$ $(0 < \alpha < 1)$ and apply Cantelli's inequality [75], we see that

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

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 $\phi$ are independent. The Bobkov-Houdré variance bound for Lipschitz functions [76] 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 $\phi$. This bound does not explicitly show that minimizing $\mathrm{Var}_{\phi \sim \Phi}[\mathrm{J}]$ decreases $L$, but it suggests a correlation that we hope to revisit in future work.

## 3.5 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.

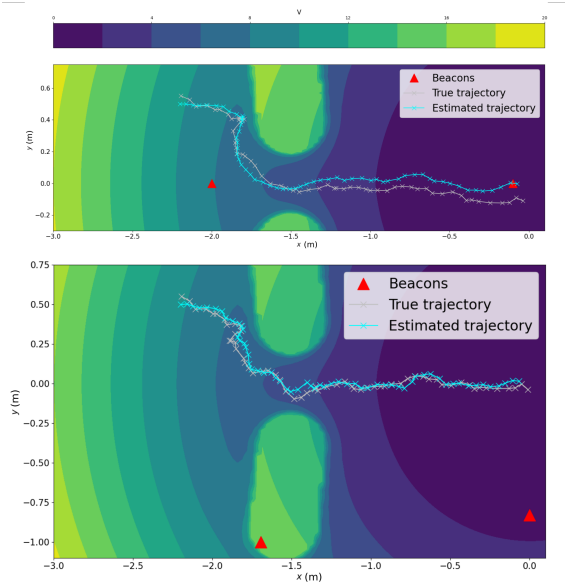
### 3.5.1 Case study: optimal sensor placement for navigation

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.

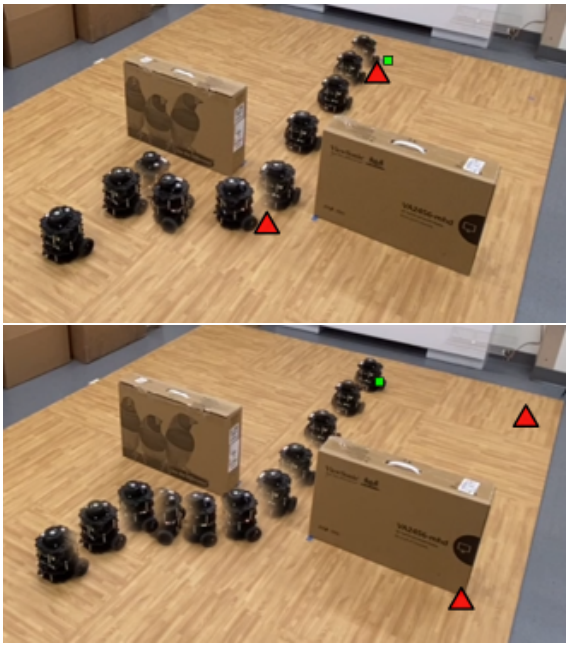
The design parameters are the $(x, y)$ locations of two range beacons and two feedback controller gains (6 total design parameters). The exogenous parameters include uncertainty in the robot's initial state along with actuation and sensing noise at each of $T$ timesteps ($3 + 6T$ total exogenous parameters). 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. We also include code in the supplementary materials for defining this design problem in our framework and running our design optimization and sensitivity analysis methods. The simulator and cost functions are implemented in Python using the JAX framework for automatic differentiation.

Fig. 3.3a 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) took 3 minutes 34 s on a laptop computer (8 GB RAM, 1.8 GHz 8-core processor).

We tested 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 were fused into a full state estimate from which range measurements were derived (the full state estimate was hidden from the robot, which only received the emulated range measurements). The control frequency was increased from 2 Hz in simulation to 10 Hz in hardware, and the obstacles were recreated in our laboratory. The hardware results, shown in Figs. 3.3b and 3.4a, 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




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



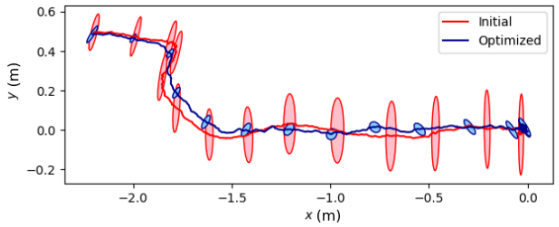
(b) Hardware performance of initial (top) and optimized (bottom) AGV designs. Square (green) shows the goal; triangles (red) show beacon locations. The optimized design eliminates drift relative to goal.

the EKF; Fig. 3.4a shows how the optimized design greatly reduces uncertainty in the state estimate compared to the initial design. No parameter estimation or tuning was required.

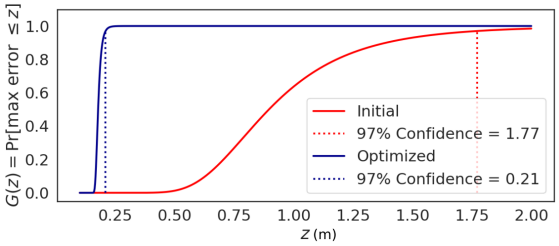
Finally, we apply the robustness analysis from Section 3.4 to certify the maximum absolute estimation error $\|x_t - \hat{x}_t\|$ in the optimized design (in meters, projected into the $xy$ plane). Note that this error is different from the cost used during optimization, but we can still apply Algorithm 3.4.1 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 3.4.1 to the maximum estimation error for both the initial and optimized designs. These distributions are shown in Fig. 3.4b; 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.

### 3.5.2 Case study: collaborative multi-robot manipulation

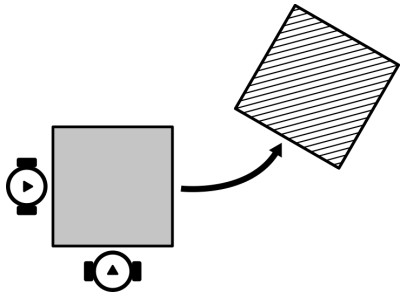
Our second example involves finding a control strategy for multi-agent collaborative manipulation. In this setting, two ground robots must collaborate to push a box from its current location to a target pose (as in Fig. 3.5a). Given the desired box pose and the current loca-



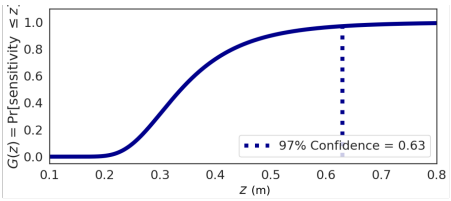

(a) Hardware results for EKF state estimates and posterior error covariance $3\sigma$ ellipse for initial and optimized AGV designs.




(b) GEVD CDF fit using Algorithm 3.4.1 for the maximum absolute estimation error in the $xy$-plane in both the initial and optimized AGV designs, with 97% confidence levels.



(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 3.4.2 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.

tion of each robot, a neural network plans a trajectory for each robot, which the robots then track using a feedback controller ($\theta$ 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 position 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

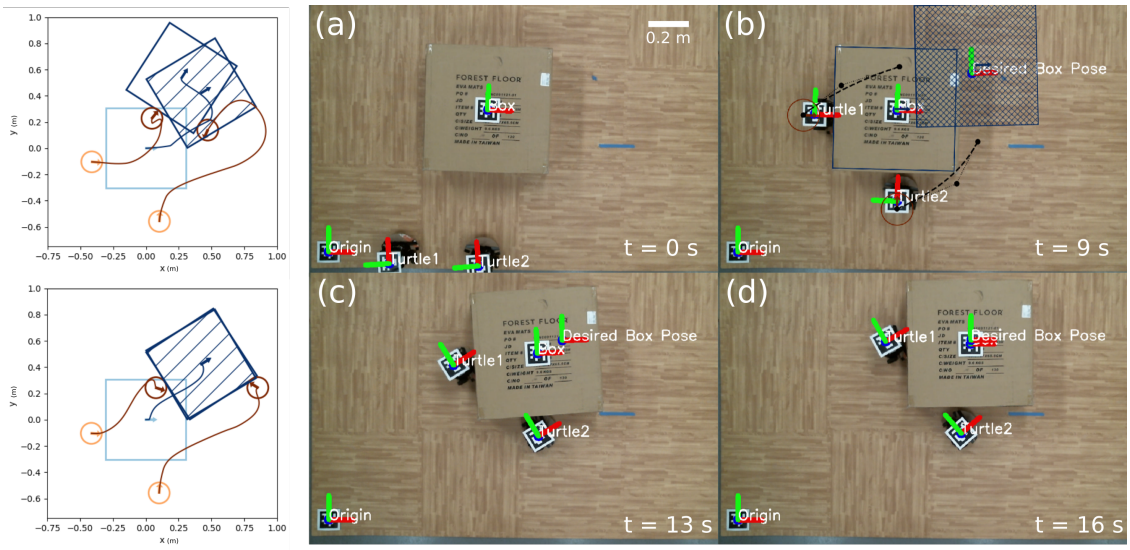



Figure 3.6: Left: Initial (top) and optimized (bottom) manipulation strategies in simulation (light/dark colors indicate initial/final positions, stripes indicate desired position). Right: Optimized manipulation strategy deployed in hardware (video included in the supplementary materials). (a) The robots 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.

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 cannot choose this value at design-time it must be included in $\phi$. As a result, minimizing the expected cost with respect to variation in $\phi$ 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. 3.6 shows a comparison between the initial and optimized strategies, and Fig. 3.7 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 tested the optimized design in hardware, again using the Turtlebot 3 platform. An overhead camera and AprilTag [77] markers were 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

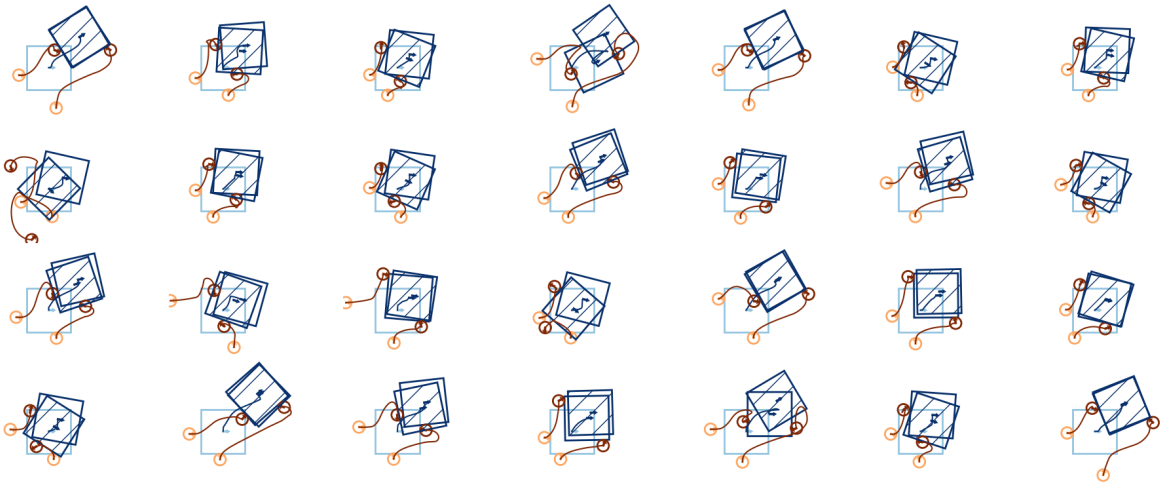


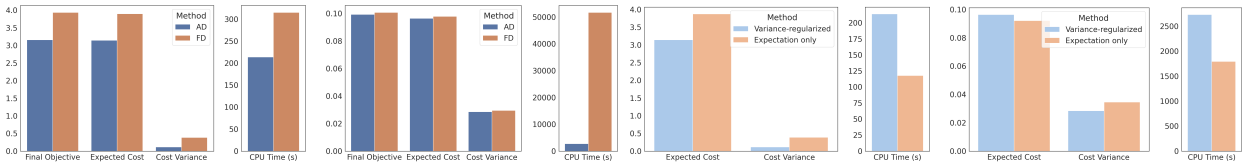
Figure 3.7: Additional examples of optimized multi-agent manipulation behavior in simulation, showing that the optimized strategy reaches the goal in most cases. Each example shows the results of executing the optimized pushing strategy for 4 s with a randomly selected set of friction coefficients, random target pose, and random initial robot poses. Light/dark colors indicate initial/final positions, respectively, and the striped box indicates the target pose.

100 Hz until the box reaches its desired location or a time limit is reached. Results from this hardware experiment are shown in Fig. 3.6, and a video is included in the supplementary materials. Again, no parameter tuning or estimation 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 3.4.2 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 $\phi$ at nominal values (box mass 1 kg and target pose $[0.3, 0.3, 0.3]$). The fit distribution is shown in Fig. 3.5b. 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.

### 3.5.3 Design optimization ablation study

Our case studies in Sections 3.5.1 and 3.5.2 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




(a) AD vs. FD; sensor placement

(b) AD vs. FD; manipulation

(c) Effect of VR; sensor placement

(d) Effect of VR; manipulation

Figure 3.8: (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.

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 [37] suggest that exact gradients may be less useful than finite-difference or stochastic approximations when the objective is stiff 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. 3.8 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. An exciting extension of our framework involves combining AD with stochastic population methods, but we leave this to future work.

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. 3.8; 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 in our second case study.

### 3.5.4 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. Put simply, is our approach falsely optimistic (underestimating the worst-case) or conservative (overestimating)?

To answer these questions, we compare the cumulative distribution function (CDF) of the fit GEVD with an empirical CDF observed from data. Algorithms 3.4.1 and 3.4.2 both estimate a posterior distribution for $\mu$, $\sigma$, and $\xi$, 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 [78].

Fig. 3.9 compares the estimated GEVDs and empirical data for worst-case performance in the sensor placement example (fit using Algorithm 3.4.1) and sensitivity in the manipulation example (fit using Algorithm 3.4.2). 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 appendix). In the latter case, 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. 3.9.

| | 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 3.4.1 is sound in this case.

| | Null Hypothesis | KS Statistic | p-value | Conclusion ($p < 0.05$) |
|---|---|---|---|---|
| False Optimism | 97% GEVD under-estimates sensitivity | 0.0399 | $6.75 \times 10^{-5}$ | Reject; 97% GEVD *does not* under-estimate sensitivity |
| Conservatism | 3% GEVD over-estimates sensitivity | 0.0618 | $1.03 \times 10^{-10}$ | 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 3.4.2 is sound in this case.

## 3.6 Discussion

In this paper, we develop 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 algorithmic and theoretical 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.

To validate this framework and demonstrate the usefulness of our contributions, we present two case studies to highlight how our framework can be used for design optimization in practical robotics problems. Moreover, we show that our optimized designs are robust enough to deploy in hardware, and data from these hardware experiments validate our optimization approach. Finally, we provide an ablation study to justify the architecture of our optimization framework and a statistical analysis showing the soundness of our robustness analysis techniques. We hope that by combining flexible design optimization with robustness certification in our framework we can increase the productivity of robotics engineers, shorten the design cycle, and help bring more complex robotic systems to life.

There are a number of interesting directions for future work. First, since our approach relies on sampling from $\Phi$ without any further information, it will require a large number of samples to accurately capture rare events. We can close this gap when more information about $\Phi$ is available, perhaps using adversarial testing or importance sampling. Second, our framework is currently focused on tuning continuous parameters; we hope to incorporate stochastic search over discrete parameters in a future work. Finally, we hope to expand the software implementation of our framework to include a richer library of autonomy building blocks and demonstrate a wider range of applications in designing autonomous systems, including robotic arms, autonomous air and spacecraft, and networked autonomous systems.

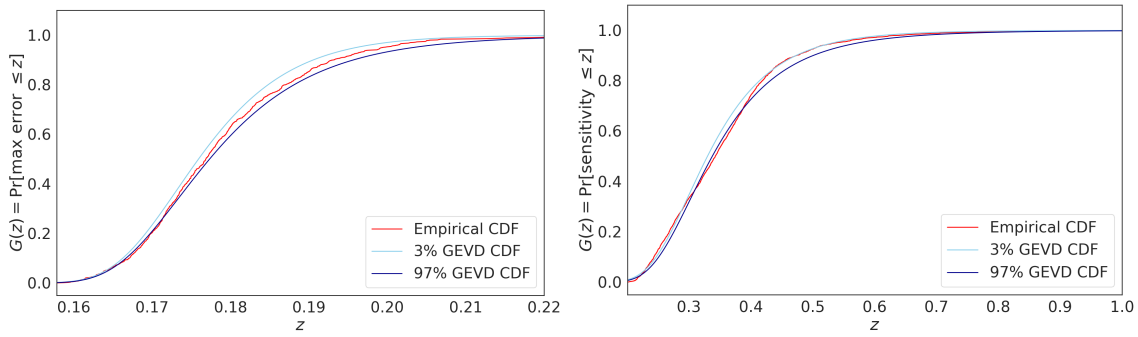



Figure 3.9: Comparison of fit GEVD CDFs and empirical CDF for worst-case estimation error in the sensor placement example (top) and sensitivity in the manipulation example (bottom).

# Chapter 4

# Counterexample-guided Optimization with Formal Specifications

In Chapter 3, I demonstrate 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 $\phi$. First, 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 $\phi$ that lead to high cost).

In this chapter, we present a *counterexample-guided* design optimization pipeline that replaces the variance regularization used in Chapter 3 with an adversarial optimization that alternates between optimizing the design $\theta$ and a set of adversarial counterexamples $\phi$. In addition, we combine end-to-end design optimization with 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 [79].

There are different types of temporal logic that may be used to specify robot behaviors, but most relevant to robotics problems is *signal temporal logic* (STL), a formal language for specifying properties of real-values continuous-time signals [80]–[82]. Using STL, a user can specify a variety of robot behaviors by combining logical and temporal operators that specify the desired order and dependencies between subtasks [81], [83], [84]. Although the syntax of STL can be opaque, there are tools for translating between STL and natural language [85].

A number of previous works have approached the problem of synthesizing robot behaviors, most often in the form of motion plans, from STL specifications, using either abstraction-based methods [83], mixed-integer optimization [81], [86], sampling-based planning [87], [88],

or nonlinear optimization [33], [89], [90].

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 [86], [91], [92]. 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 [81]).

Some recent work has used nonlinear optimization to solve robot planning and design optimization problems with STL specifications [33], [82], [84], [90], [93]. 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 they 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 ([93] 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 [92], 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 $\phi$) to guide the optimization of the design parameters $\theta$. 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 to enable efficient optimization new designs and counterexamples.

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


Figure 4.1: TODO An illustration of robot behaviors as continuous-time signals.

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 $T$: $s_1, \ldots, s_T$. 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 $(t_i, s_i)$. This continuous-time view of robot behavior is illustrated in Fig. 4.1. 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, but still maintain 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 [94]. We can define the syntax of an STL formula $\psi$ 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 $\mu : \mathcal{S} \mapsto \mathbb{R}$ 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 $I$, $\psi_1$ must be true until $\psi_2$ 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 $\psi$, 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 [94]:

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

A useful feature of STL is that it also admits a so-called *quantitative semantics*. Intuitively, if the Boolean semantics tell us if 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 quantiative semantics as $\rho :: \Psi \times \mathcal{S} \times \mathbb{R}^+ \mapsto \mathbb{R}$, where $\rho(\psi, s, t)$ is the *robustness* of the signal $s$ with respect to the formula $\psi$ 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:

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

where $\top$ is a constant defined to be greater than all other real values. In practice, linear-time algorithms exist to compute $\rho$ from a given piecewise-affine signal $s$ [94].

## 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:

$$\min_{\theta} \max_{\phi} \quad \lambda J(\theta, \phi) - \rho\left(\psi, S(\theta, \phi), 0\right) \tag{4.2a}$$

$$\text{s.t.} \quad c_{\theta,i}(\theta) \geq 0 \quad \forall i \in \mathcal{I}_\theta \tag{4.2b}$$

$$c_{\phi,i}(\phi) \geq 0 \quad \forall i \in \mathcal{I}_\phi \tag{4.2c}$$

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 $\theta$ and $\phi$ with respect to STL specification $\psi$. We subtract $\rho$ 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 $\lambda$ 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 have also included 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 in practice.

## 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 $\theta$ and corresponding adversarial exogenous parameters $\phi$ such that neither the designer nor the adversary have an incentive to change [95].

In order to solve this problem, we must address two challenges. First, in order to efficiently solve this nonlinear optimization problem, we must be able to compute gradients of $\rho$ with respect to $\theta$ and $\phi$; we address this problem using differentiable programming, as discussed in Section 4.3.1. Second, although gradient-based optimization is a sample-efficient method for solving problems like (4.2), there is a risk that the resulting local Nash equilibrium will not be robust; i.e., overfitting the design $\theta$ to a particular adversarial $\phi$ so that the design performs poorly when $\phi$ changes. We address this problem by developing a novel 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 $\rho$, for example using zero-order gradient estimators [37] or black-box optimizers [4], 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, $\rho$ is a continuous but non-smooth function of $s$, $t$, and all parameters of the formula $\psi$. Smooth approximations to $\rho$ 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 [82] and later used in [89], [90]; other works [33] 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 [94]. We use the JAX framework [29] for efficient automatic differentiation and just-in-time compilation. In contrast to the implementation in [33], 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 a novel 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 $\theta$ should perform well against a wide range of adversarial exogenous parameters $\phi$. A common approach to finding Nash equilibria in two-player games like (4.2) is the family of nonlinear Gauss-Seidel-type methods [95], which solve min-max problems alternating between optimizing $\theta$ and $\phi$. 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 $()\theta^*, \phi^*)$ will be a local Nash equilibrium [95].

The risk in using this simple iterative algorithm is that the optimization for both $\theta$ and $\phi$ can get stuck in local minima. These local minima not only lead to suboptimal performance of the design $\theta^*$, but also "overconfidence" when $\theta^*$ is overfit to a particular value of $\phi^*$, 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 $\phi$. 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 [64], [96].

First, we build on the success of domain randomization in improving the robustness of machine learning models to environmental uncertainty [96]. Instead of optimizing $\theta$ against a single $\phi$, we can maintain a dataset $\mathcal{D}_N = \{\phi_i\}_{i=1}^N$ of exogenous parameters, and optimize $\theta$ against all of them simultaneously.

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

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

This approach encourages the design to be robust to a wide range of adversarial exogenous parameters, rather than overfitting to a single value of $\phi$. Unfortunately, domain randomization is relatively sample-inefficient; it may require a large number of random samples to cover the range of possible exogenous parameters. To address this issue, we take inspiration from a second idea from the optimization and learning literature: learning from counterexamples [64]. Our key insight is that we are not limimted to contructing the training dataset purely from random examples. Instead, we can use the values of $\phi^*_i$ from successive iterations of the alternating Gauss-Seidel process as adversarial counterexamples to guide the optimization of $\theta$.

Together, these insights lead to our counterexample-guided iterative Gauss-Seidel optimization method, pseudocode for which is given in Algorithm 4.3.1. This algorithm begins by initializing a dataset with $N_0$ i.i.d. $\phi$ sampled randomly from $\Phi$, then alternates between

solving the two optimization problems in (4.7a). At each step, the adversary's current best counterexample $\phi^*$ 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 $\theta^*$ that are more robust to disturbances despite using fewer samples of $\phi$.

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 $J_\psi$, 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 $\theta$, we are not aware of any prior work using gradients with respect to disturbance parameters $\phi$ for robust optimization. Some prior works have used counterexamples to guide mixed-integer solvers to plan from STL specifications [92], 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 $\quad \theta^* = \arg\min_\theta \mathbb{E}_{\mathcal{D}_N}[J(\theta, \phi_i)]$
5 $\quad \phi^* = \arg\max_\phi J(\theta^*, \phi)$
6 $\quad$ **if** $\phi^* = \phi^*_{prev}$ **then**
7 $\quad\quad$ **break**
8 $\quad \phi^*_{prev} \leftarrow \phi^*$
9 $\quad$ 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 renezvous problem, which was originally proposed as a benchmark in [97]. 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 [97], 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 [97].

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. 4.2. We define

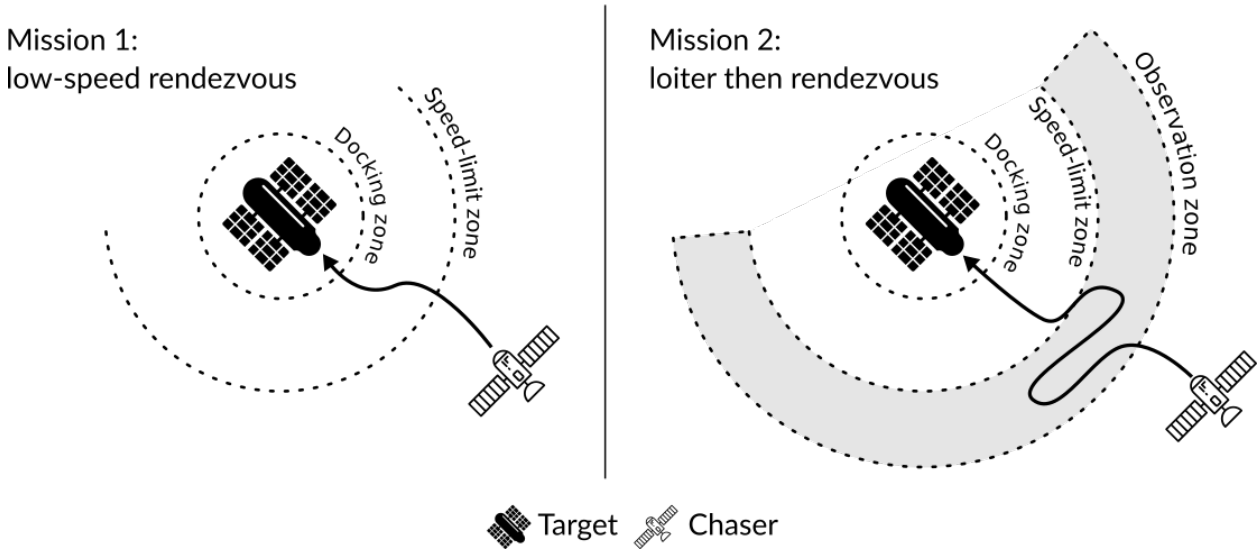



Figure 4.2: Two satellite rendezvous missions used in our experiments. In the first mission, the chaser satellite must eventually reach the target while respecting a maximum speed constraint in the region immediately around the target. In the second mission, the chaser must still reach the target and 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.

STL specifications for each mission, $\psi_1$ and $\psi_2$:

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

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 $\psi_{\text{loiter}}$ 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 $T_{obs} = 10$ 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 $\theta$ 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 $\phi$ 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 [92] and [91] and the nonlinear optimization approach from [82], [90].

The MIP approach uses a model-predictive control formulation, and [92] 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 [82], [90] that include domain randomization with either 32 or 64 samples of $\phi$. These methods are similar to those proposed in [33], 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. 4.3 and 4.4 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 were run on a laptop computer with 8 GB of RAM and a 1.8 GHz 8-core CPU.

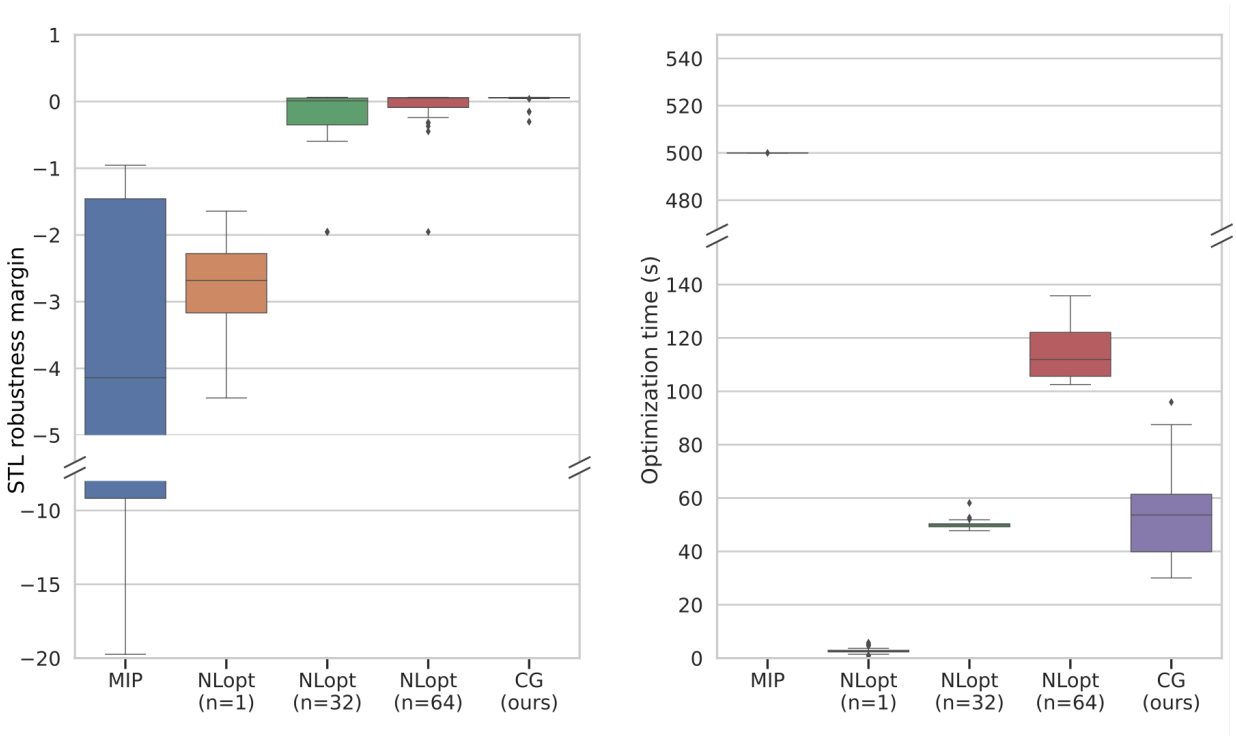



Figure 4.3: Comparison of different STL planning methods on the first example mission, averaged over 50 random seeds. Left: the robustness margin $\rho(\psi_1)$ computed for the optimized design parameters and worst-case exogenous parameters. Right: the planning time required by each method. Our method (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.

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. 4.5a), 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 $\phi$, 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. 4.5b. Despite

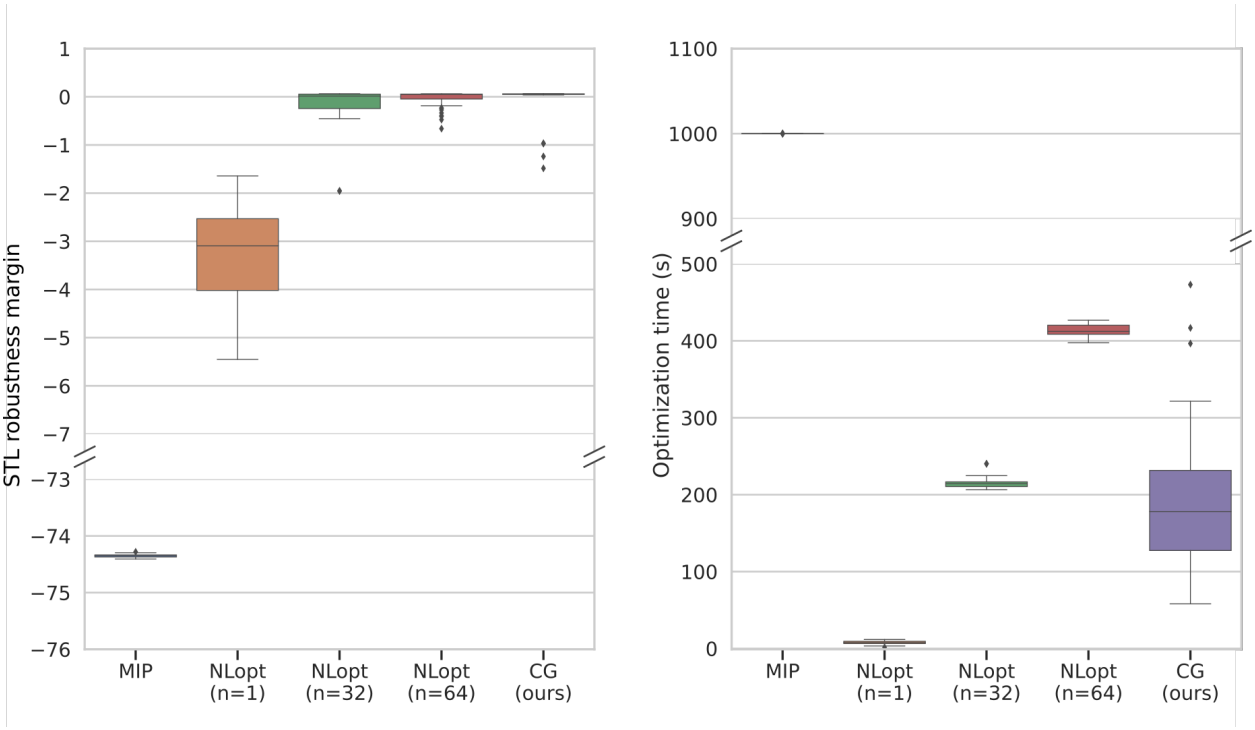
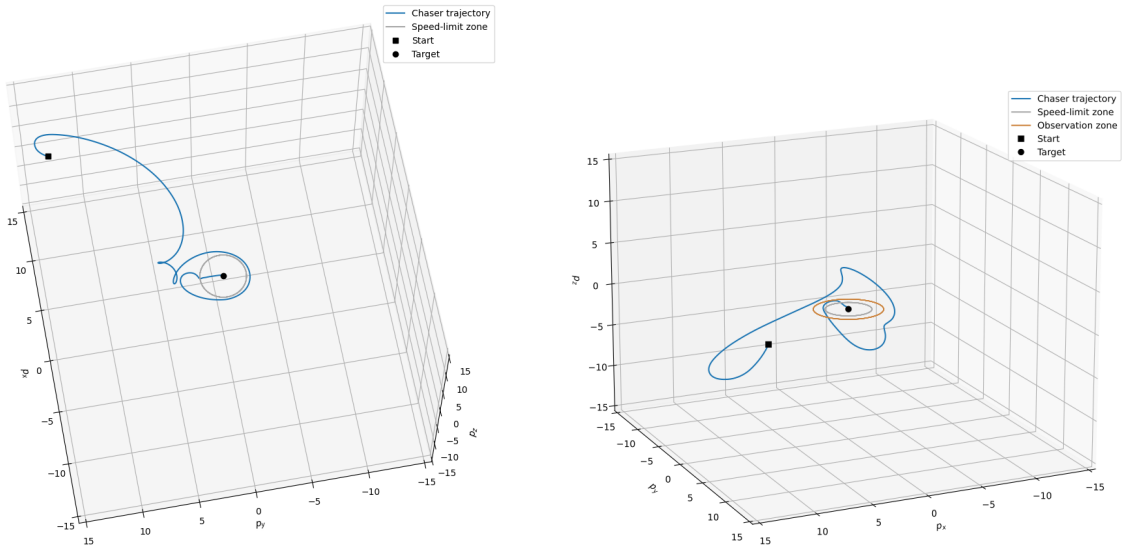



Figure 4.4: Comparison of different STL planning methods on the second example mission, averaged over 50 random seeds. Left: the robustness margin $\rho(\psi_2)$ computed for the optimized design parameters and worst-case exogenous parameters. Right: time required by each method to find a plan. Our method (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.

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 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 Discussion

In this chapter, we introduce a counterexample-guided method for end-to-end robot design optimization based on formal behavior specification. 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 design time using counterexample-guided adversarial optimization (rather




(a) Mission 1.

(b) Mission 2.

Figure 4.5: Optimized trajectories found using our method for the two satellite rendezvous missions. Figures © IEEE 2022; used with permission.

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 (or any 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; after all, local optimization methods have been successful in many robotics and learning applications [50]. Unfortunately, for safety-critical applications, a sub-optimal counterexample is potentially worse than no counterexample at all. This is because sub-optimal counterexamples — i.e. artificially easy counterexamples — can lead to overoptimistic false negatives where we erroneously believe that we have successfully addressed the worst-case disturbance. We can see this issue manifest in Figs. 4.3 and 4.4 in the few failures that remain even after Algorithm 4.3.1 converges; if Algorithm 4.3.1 converges to a local Nash equilibrium, then it will not correctly identify 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, combing global probabilistic inference methods with efficient gradient-based search to reduce the risk of overconfidence due to

artificially easy counterexamples.

# 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.

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. This chapter is based in part on the author's published work in [98].

## 5.1 Related work

Our work builds on a rich literature of different verification and testing techniques, which we review here.

**Model-based verification** Early work on model based verification relied on logical models to search for failures using satisfiability (SAT) solvers [7], [8]. The computational expense of SAT, which prevented these methods from scaling to high-dimensional problems, motivated the development of more recent approaches using mathematical dynamics models and tools like reachability analysis [9] and optimal control [11] to identify counterexamples. The challenge in applying all of these methods is that it is often difficult (or impossible) to construct a symbolic model of the system under test (e.g. when vision is involved). In this work, our aim is to preserve the interpretability of model-based verification without relying on a symbolic model. Instead, we develop a simulation-based approach using automatic differentiation (when available) to accelerate the search for counterexamples. Simulators are widely used in robotics, and recent work on differentiable simulation and rendering has made gradients available for a range of scenarios [20], [31], [41], [42], [50], [79], [99], [100].

**Adversarial testing** A common approach in the falsification literature is adversarial optimization, which formulates the search for counterexamples as a two-player game between the system designer and the environment [4], [5], [15]–[23], [79], [101], [102]. Adversarial methods have been developed in both model-based and model-free contexts, where the main distinction is that the former assume access to gradients. Gradient-based adversarial methods have demonstrated impressive sample efficiency [79], [101], but the drawback is that they rely on local gradient ascent to search for counterexamples, typically yielding only a single adversarial counterexample that may be only locally optimal. Moreover, existing gradient-based methods cannot handle systems with visual feedback. [23] find that skipping the rendering step during backpropagation is sufficient for gradient-based failure-case generation, but this method is not able to repair visual-feedback policies. [103] learn a proxy model for the end-to-end dynamics, including a vision-based policy, but end-to-end proxy modeling is generally not applicable to rare-event generation, since it is difficult to include enough failure examples in the training data for the proxy model. On the other hand,

gradient-free methods reduce the risk of getting stuck in local optima and support vision in the loop, but they require additional computation and struggle to scale to high-dimensional search spaces. In contrast, in this work we develop a probabilistic gradient-based approach, using sample-efficient gradient-based sampling algorithms to avoid local minima and scale to high dimensional search spaces, and using differentiable rendering to support visual feedback systems. We also provide a gradient-free variant of our method for use in cases when a differentiable simulator is not available.

**Inference** Our probabilistic approach builds off of prior work on inference as a verification tool. [5] propose an end-to-end verification tool for autonomous vehicles based on gradient-free adaptive importance sampling. [3] use gradient-free Markov Chain Monte Carlo (MCMC) to find counterexamples. [103] and [104] 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 are not able to improve the system (e.g. by re-optimizing the controller) to fix these failures once they have been discovered. Our method aims to fill this gap by combining 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.

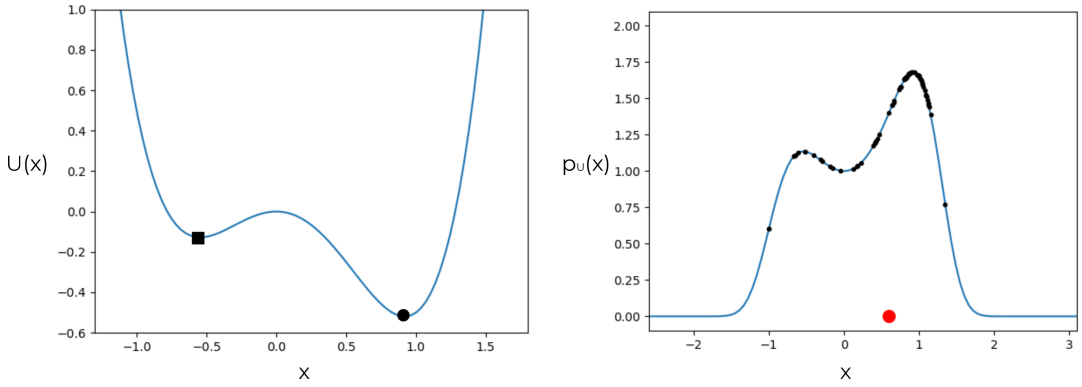
## 5.2 From optimization to inference

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 $x = -0.544$.

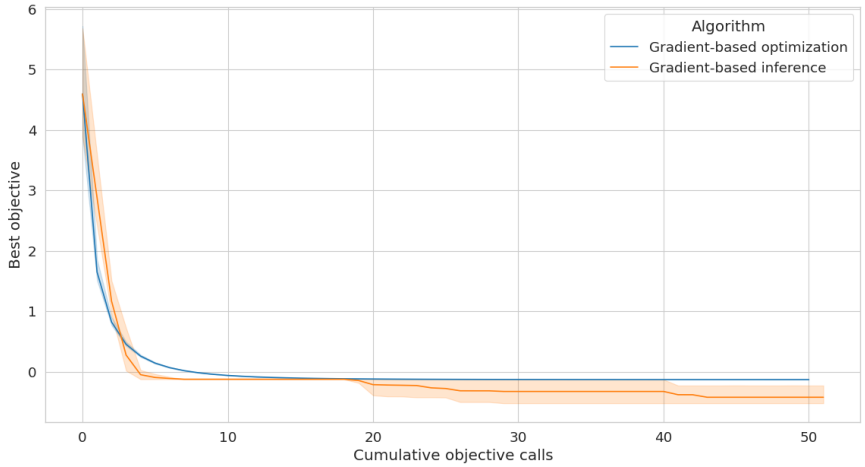
To avoid getting stuck in this local minimum, we can convert this optimization problem into an inference problem; i.e. sampling from the unnormalized probability density:

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

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




(a) Left: a simple bimodal cost landscape $U(x) = x^4 - 0.5x^3 - x^2$, with a locally optimal solution at $x \approx -0.544$ and a global optimum at $x \approx 0.919$. Right: the corresponding unnormalized likelihood $p(x) \propto e^{-U(x)}$, showing 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.

the global minimum. By sampling from this distribution, we can find the global minimum with high probability (this is the so-called *Optimization as Inference* perspective [45], [46]). In practice, we can approximately sample from this distribution using Markov Chain Monte Carlo (MCMC) methods, which we will discuss in more detail shortly. The benefit of these methods is that they balance exploring different regions of the distribution with exploiting high-likelihood areas; we can see this behavior in Fig. 5.1b, which 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 [105]; 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 5.2.1. 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 [67], [105] 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.

**Algorithm 5.2.1:** Metropolis-adjusted Langevin algorithm (MALA, [45], [106])

**Input:** Initial $x_0$, steps $K$, stepsize $\epsilon$, density $p(x)$.
**Output:** A sample drawn from $p(x)$.

1 **for** $i = 1, \ldots, K$ **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** $x_K$

We have shown how transitioning from gradient-based optimization to inference (in particular, 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).

### 5.2.1 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 the modes are close together in the search space). However, although MCMC methods enjoy good asymptotic guarantees, including ergodicity and convergence to the target distribu-

tion in the inifinite-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 [107]. 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.

### 5.2.2 Sensitivity to gradient quality

As discussed at the start of this chapter and in prior work [37], [38], 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 discontinuous at the point when contact is made or broken. 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 [38] 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 discernable difference between the gradient-free and gradient-based inference methods. In higher dimensions $N > 10$, gradient-free RMH does not even reach the flat region where gradient descent gets stuck, while gradient-based MALA converges even in $N = 1000$ dimensions.

We attribute this improved performance to three factors. First, the Gaussian diffision 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

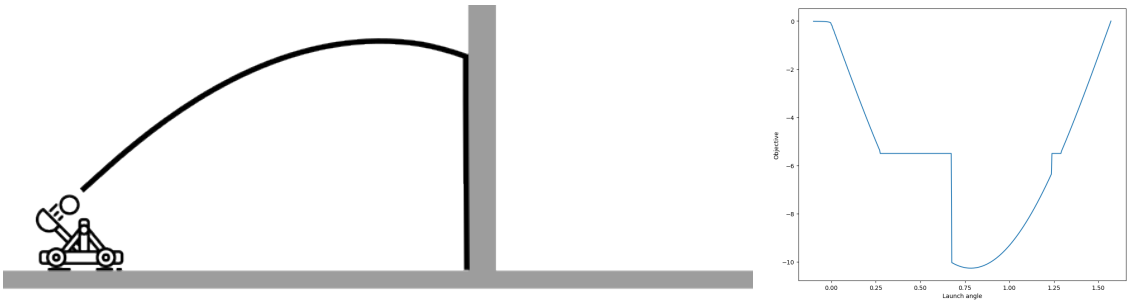
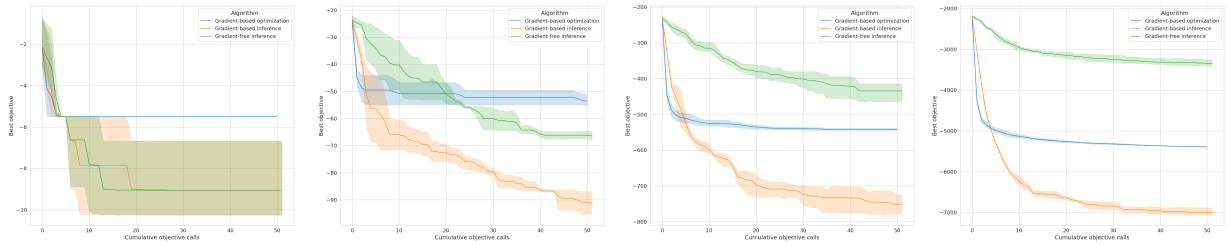


Figure 5.2: Left: the ballistic optimization problem from [38]. Right: the corresponding cost landscape.



(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.

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.

### 5.2.3 Scaling gradient-based inference to high dimensions

## 5.3 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.

### 5.3.1 Preliminaries

Let us define a "failure mode" as a set of environmental parameters $\phi$ that induce high cost for given policy parameters $\theta$, 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 $\phi^*$ given a design $\theta$, while *failure repair* is the search for updated design parameters $\theta^*$ that achieve low costs despite possible variation in $\phi$. 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 $\phi$ 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 psuedo-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 $\phi$ conditioned on a failure occurring [3], [45], [98], [103]. 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, [5], [103], and [3] also approach failure prediction using this lens. One of our key insights is that 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 $\theta_0$ 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.

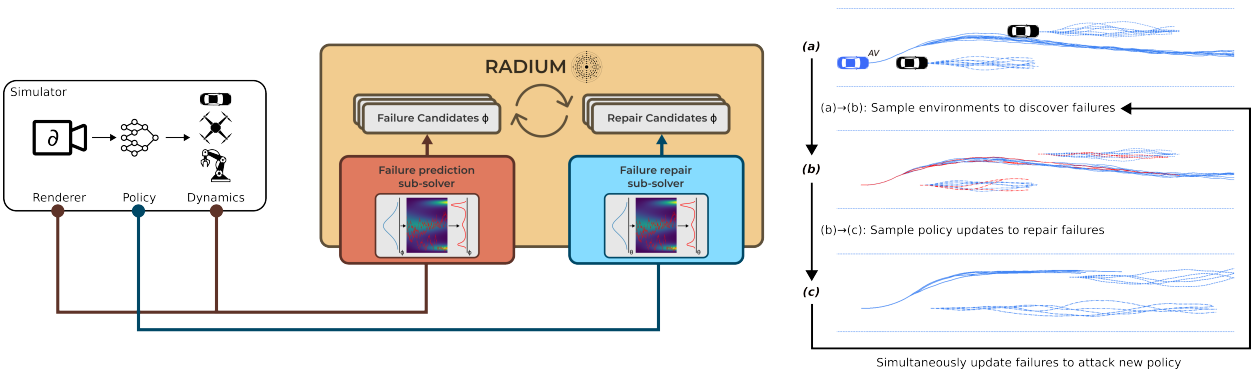



Figure 5.4: An overview of our approach for closed-loop rare-event prediction, which efficiently predicts and repairs failures in autonomous systems. Our framework alternates between failure prediction and repair sub-solvers, which use a simulated environment to efficiently sample from the distributions (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.

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 5.4 that the increased diversity from sampling rather than straightforward gradient optimization yields better solutions in practice.

### 5.3.2 Approach

Previous works have shown that sampling from a failure distribution like Eq. (5.1) can generate novel failures [3], [103], [104], 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 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 [20], [108] and sequential Monte Carlo algorithms [109] 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 $\tau$ 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; [110]), Hamiltonian Monte Carlo (HMC; [111]), and the Metropolis-adjusted Langevin algorithm (MALA; [112]), variational inference methods like Stein Variational Gradient Descent (SVGD; [113]), and other black-box methods like adaptive importance sampling [5]. RADIUM is able to use any of these sampling methods as sub-solvers for either the 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 (but non-convex) problems [45],

but autonomous systems with visual feedback have historically been treated as black-boxes due to an inability to backpropagate through the rendering step [3], [5], [103]. To enable the use of gradient-based samplers in RADIUM, we draw upon recent advances in differentiable simulation and rendering [31], [32] provide end-to-end gradients. In Sections 5.3.4 and 5.4, 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 where RADIUM uses gradient-free sampling subroutines.

### 5.3.3 RADIUM

Pseudocode for RADIUM is provided in Algorithm 5.3.1. The algorithm maintains a population of candidate repaired policies $[\theta_1, \ldots, \theta_n]$ and failures $[\phi_1, \ldots, \phi_n]$ 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 and the provided implementation, 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 5.2.1.

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.

### 5.3.4 Theoretical analysis

The iterative adversarial sampling process defined in Alg. 5.3.1 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 con-

**Algorithm 5.3.1:** 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. 5.2.1)

Sample initial failures and policies using priors:
$[\phi_1, \ldots, \phi_n]_0 \overset{\text{iid}}{\sim} p_{\phi,0}$, $[\theta_1, \ldots, \theta_n]_0 \overset{\text{iid}}{\sim} p_{\theta,0}$;

**for** $i = 1, \ldots, N$ **do**

  $\tau \leftarrow 1 - e^{-\alpha i/N}$; `// Tempering schedule`

  Sample $[\theta_1, \ldots, \theta_n]_i \overset{\text{iid}}{\sim}$ (5.4); `// Sample repaired policies`

  Sample $[\phi_1, \ldots, \phi_n]_i \overset{\text{iid}}{\sim}$ (5.3); `// Generate failures attacking` $\theta_i^*$

**end**

**Return:** Repaired policy $\theta_N^* = \arg\max_i$ (5.4) and failures $[\phi_1, \ldots, \phi_n]_N$ attacking that policy.

verge 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. 5.3.1?

**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 [110], 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; [38]). Here, we build on recent theoretical results by [45] 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 5.3.1.** *Let $J(\theta, \phi)$ be a $L$-Lipschitz smooth cost function (i.e. $\nabla J$ 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 $\epsilon$ 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

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

[45] show that MALA sampling enjoys the convergence guarantees of Theorem 5.3.1 so long as the target log likelihood is strongly convex outside of a ball of finite radius $R$ (see Theorem 1 in [45]). 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 $m$-stong 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. □

Theorem 5.3.1 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. 5.3.1, 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 $\theta$ and $\phi$ 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 5.3.2.** *The iterative adversarial sampling procedure in Alg. 5.3.1 yields policies drawn from a marginal distribution with density function*

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

*where* $\mathbb{P}(J(\theta^*,\phi) > J^* = \mathbb{E}_{\phi\sim p_{\phi,0}}[\mathbb{1}(J(\theta^*,\phi)\ge J^*)]$ *is the probability of failure when* $\phi$ *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 bounded $[0, 1]$, while the denominator grows exponentially large when a failure occurs. As a result, the marginal distribution of $\theta^*$ assigns higher probability (relative to the prior) for policies that avoid failure.

*Proof.* We can treat Alg. 5.3.1 as a two-stage Gibbs sampling procedure and apply the Hammersley-Clifford Theorem [114] 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 $\phi$ yields the marginal distribution of $\theta$, completing the proof.

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

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

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

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

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

$$\tag{5.14}$$

□

## 5.4 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 5.5 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 cyber-physical systems literature, comparing against previously-published adversarial optimization methods. More detail on each benchmark, baseline, and implementation is provided in the appendix.

### 5.4.1 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 [23], [79]. *Learning to collide (L2C)* uses black-box optimization (REINFORCE) to search for failure cases [24]. 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 sample budget for each task. Hyperparameters for all experiments are given in the appendix.

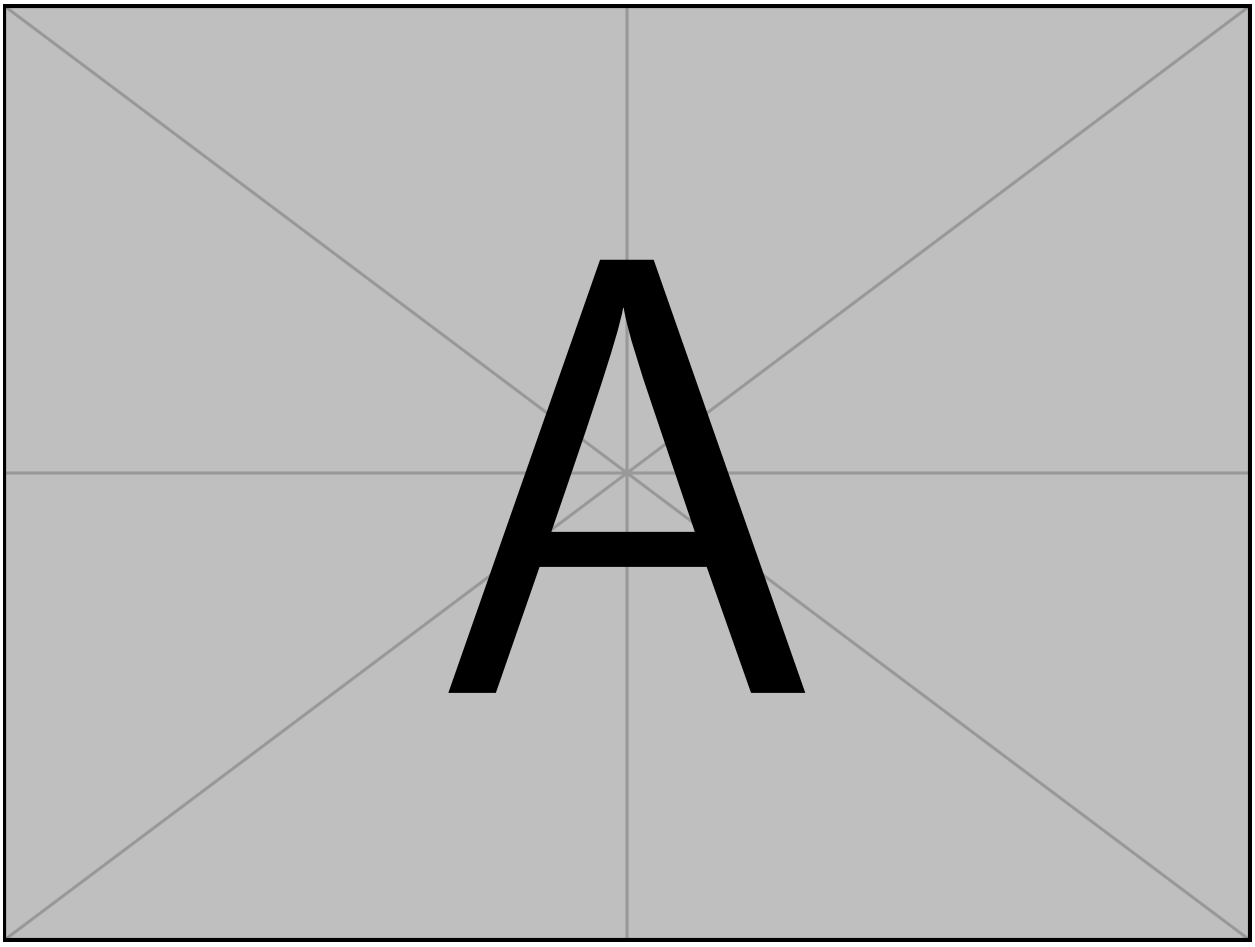



Figure 5.5: The different environments used in our simulation studies, including 5 domains without visual feedback and 3 domains with vision in the loop.

### Benchmark problems

We rely on two classes of benchmark problem in this work: 3 problems without vision in the loop, and 3 problems with vision in the loop. Each of these problem domains includes 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 supplementary material.

**Non-vision benchmarks** *Search:* a set of seeker robots must cover a region to detect a set of hiders. $\theta$ and $\phi$ define trajectories for the seekers and hiders, respectively. Failure occurs if any hider escapes detection by the seekers (which have fixed sensing radius). This 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. $\theta$ defines trajectories for each robot in the swarm, while $\phi$ 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, $\dim \theta = 30$, $\dim \phi = 1280$) and large (10 agent, $\dim \theta = 100$, $\dim \phi = 1280$) 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. $\theta$ specifies generator setpoints and $\phi$ 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 $\theta_0$ for drone and intersection environments were pretrained using behavior cloning, and CNN-based policies for the highway environment were pretrained using PPO [115]. *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 [116]. 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 $\theta_0$, and the vision benchmarks were initialized using $\theta_0$ 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 $\theta_0$ 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 $\theta_0$, $GD_a$, 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 $\phi$ 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 $99^{\text{th}}$-percentile cost for non-vision benchmarks (some of which have unbounded cost, making the $99^{\text{th}}$ 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.

### 5.4.2 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 $\theta_0$ 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 $\theta_0$ and failure modes discovered using our method (sampling $\phi$ while holding $\theta_0$ 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

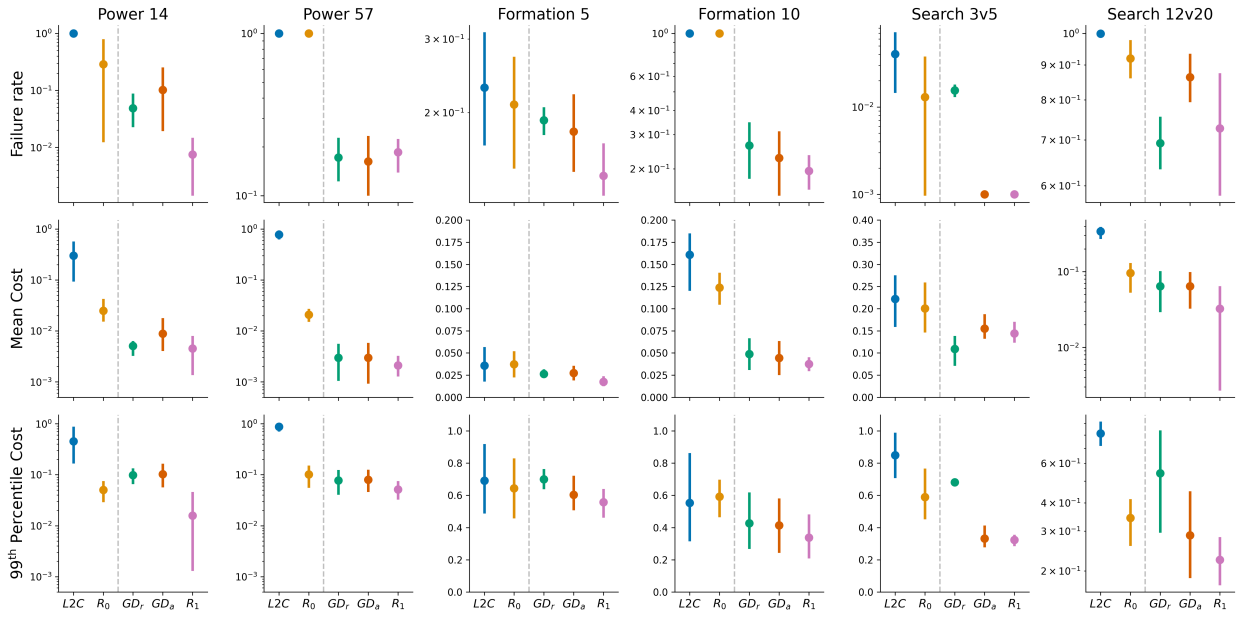



Figure 5.6: Comparison of our method (gradient-free and gradient-based variants $R_0$ and $R_1$, respectively) and baseline methods on benchmark problems without vision in the loop, showing failure rate, mean cost, and 99$^{\text{th}}$ percentile cost on a test set of 1,000 randomly sampled $\phi$. The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

maneuver (top row of Fig. 5.9).

### 5.4.3 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 99$^{\text{th}}$ percentile costs relative to gradient-free methods ($L2C$ and $R_0$) 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

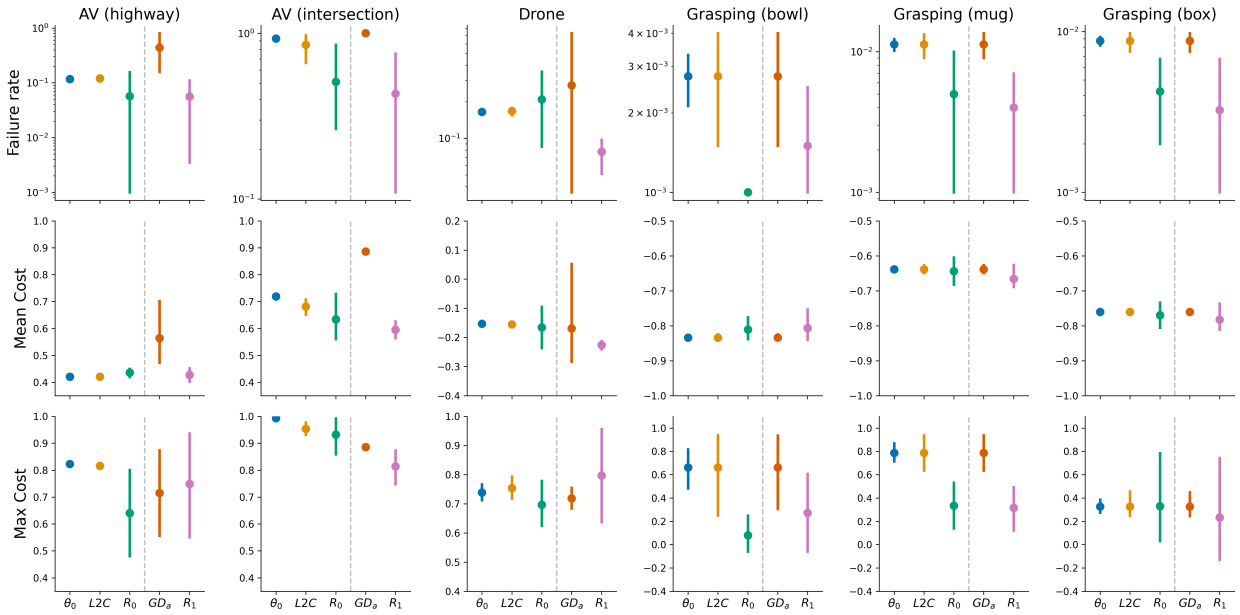



Figure 5.7: Comparison of our method (gradient-free and gradient-based variants $R_0$ and $R_1$, respectively) and baseline methods on benchmark problems with vision in the loop, showing failure rate, mean cost, and max cost on a test set of 1,000 randomly sampled $\phi$. The dashed gray lines separate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

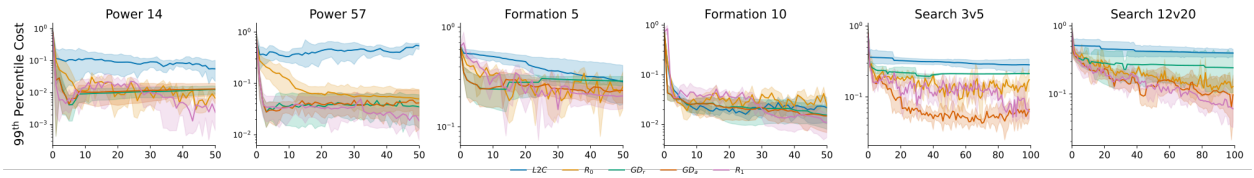
better performance on some tasks because the Metropolis-Hastings adjustment on line 5 of Algorithm 5.2.1 allows it to reject large steps caused by poorly conditioned gradients.

## 5.5 Case studies

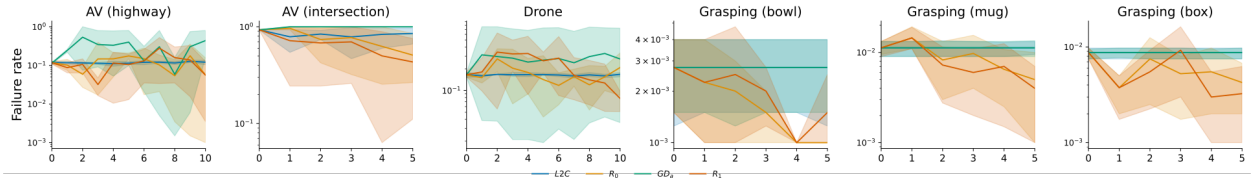
In this next section, we present three case studies to illustrate the practical use of our method. We first demonstrate how RADIUM can be used to solve a challenging optimization problem arising in the control of electrical power systems. We then provide two case studies showing how RADIUM can be applied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

### 5.5.1 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




(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 | $1.72_{0.003}$ | $6.65_{0.14}$ | $1.77_{0.06}$ | $0.0049_{3.8\times10^{-5}}$ | | |

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 [117]) 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

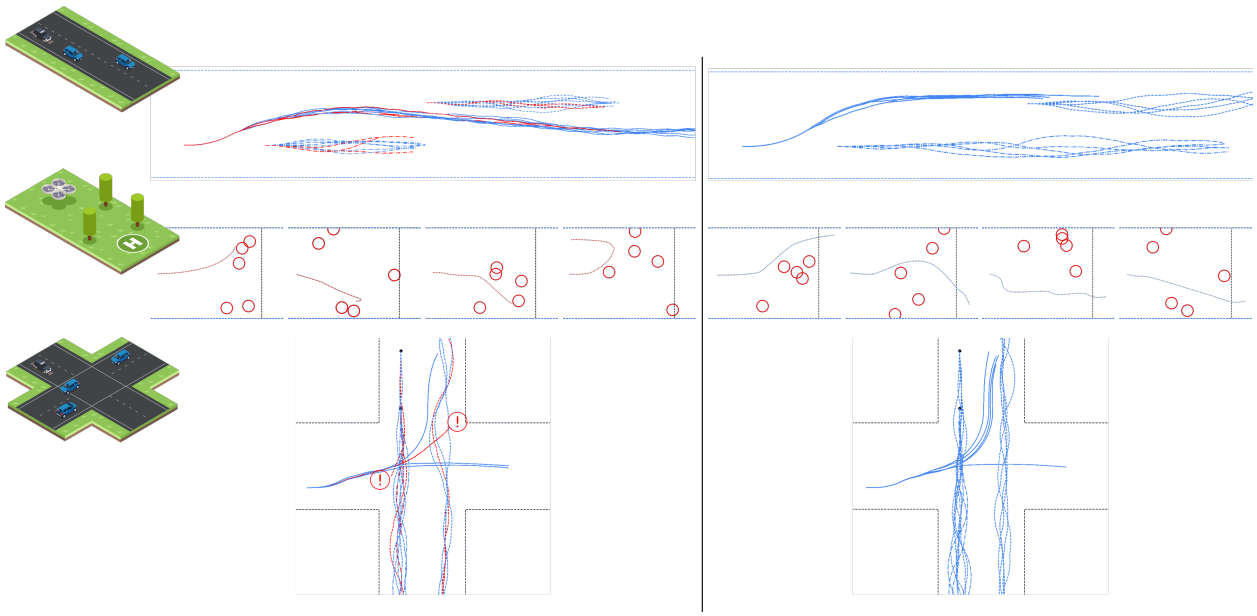


Figure 5.9: Examples of failure cases (left) and repaired policies (right) generated using our method. Failed trajectories are shown in red.

given by $\sigma(y_i)Y_{i,nom}$ where $\sigma$ 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 [118] 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 [117] and an ongoing competition run by the U.S. Department of Energy [119]. In addition to its potential economic and environmental impact [118], 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

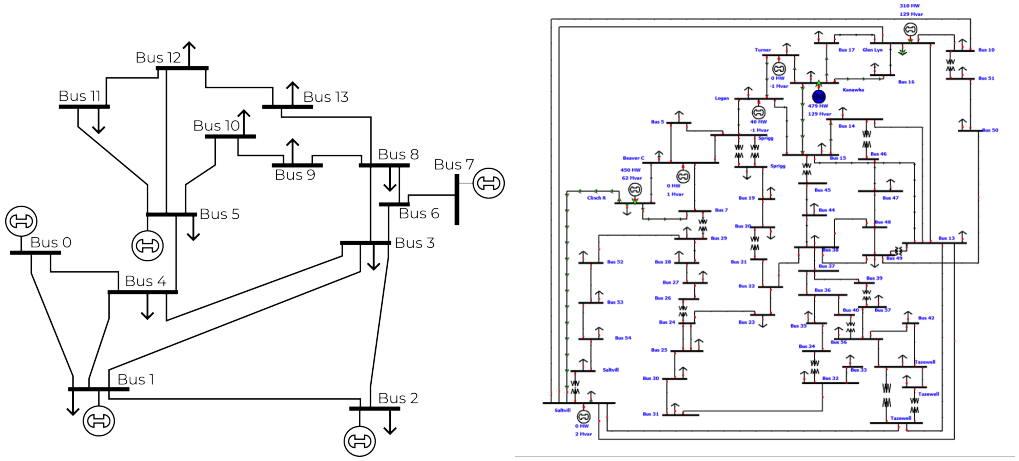


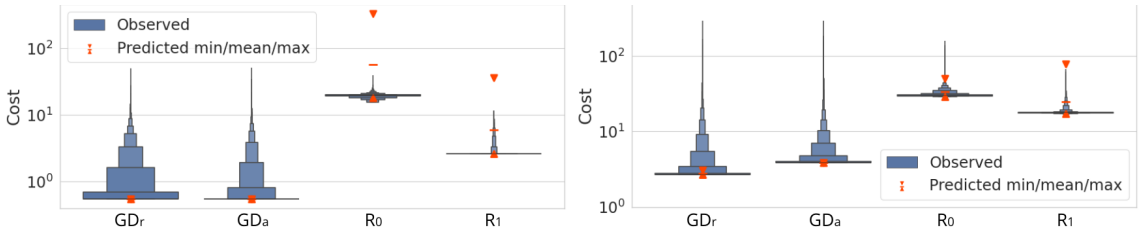

Figure 5.10: Example 14- and 57-bus electricity transmission networks [120].

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 is shown in Figs. 5.6 and 5.8a; in this section, we provide a more detailed comparison between RADIUM and prior optimization-based methods that are the state-of-the-art for this SCOPF problem [101] (the comparison with *L2C* is not shown in thi 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 5.3.1. 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 took 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 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) 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.

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 10 failure modes predicted by $R_1$, 4 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 4 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 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

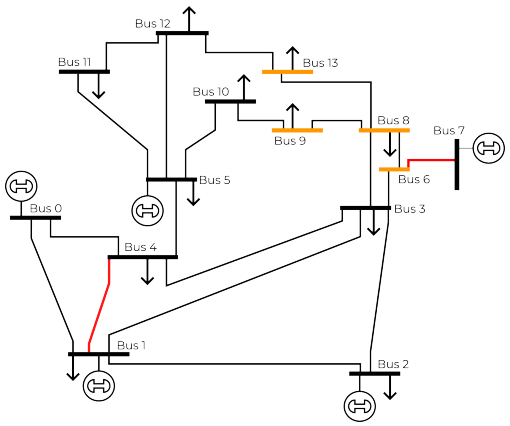



Figure 5.12: The only predicted failure modes (of 10 candidates) that causes violation of voltage constraints on the 14-bus transmission network, using the optimized design found using $R_1$. Fully disconnecting the generator at bus 7 (dotted red line) is not enough; other predicted failures include 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.

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 $99^{\text{th}}$ 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.

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 [121], and gradient descent-based methods have been shown to work well for finding local optima [101], [122], 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 $99_{\text{th}}$ 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 $\theta$ towards the end of the repair process likely helps drive constraint violations to zero by converging to the local minima

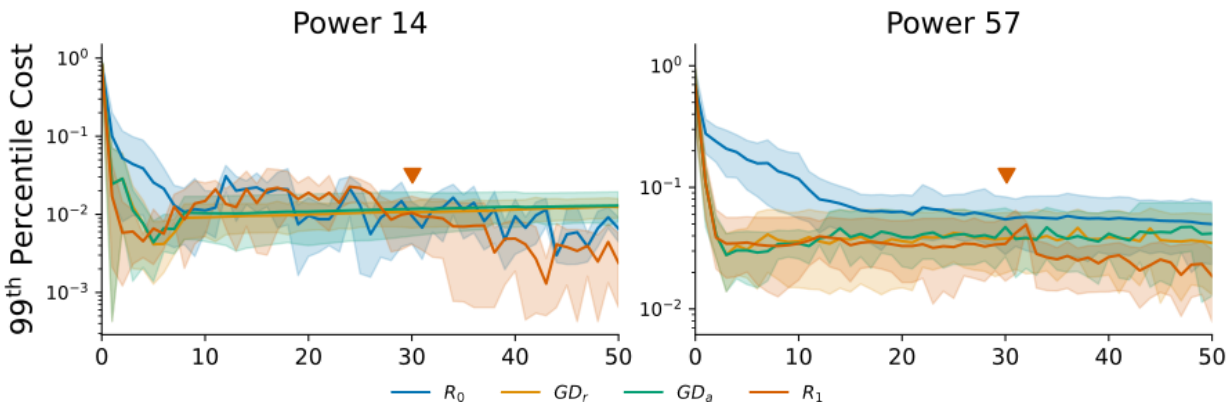



Figure 5.13: Comparison of convergence rates of different methods on 14- (left) and 57-bus (right) power networks, showing the 99$^{\text{th}}$ percentile cost of the best design after each round. The ▼ symbol indicates the start of quenching for $R_1$.

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.

### 5.5.2 Case study on multi-agent path planning

We return to the search problem used as a baseline in Section 5.4, 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 [123]. 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 × 2 m).

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

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

where $r$ is the sensing range of the seekers (0.5 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); $t_0, \ldots, t_n$ 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 [123] (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.

### 5.5.3 Case study on vision-in-the-loop control of a model racecar

TODO

## 5.6 Conclusion & Limitations

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 distri-

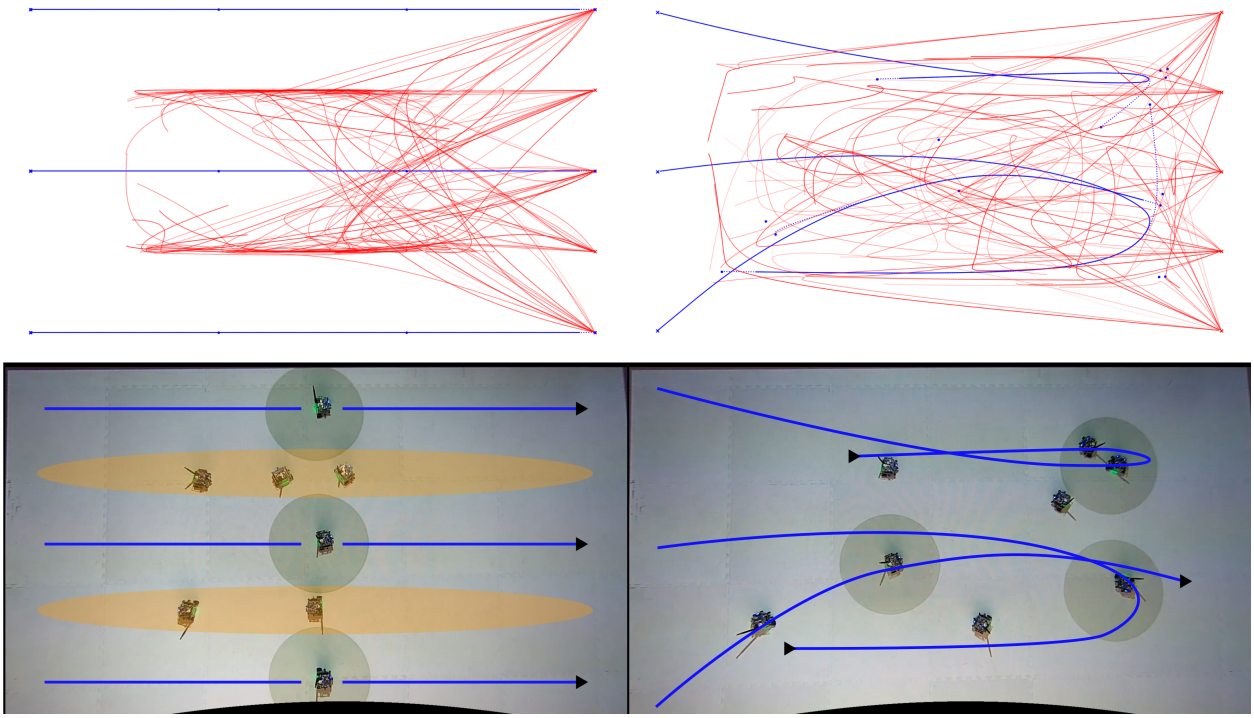


Figure 5.14: (Left) HW results for search-evasion with 5 hiders and 3 seekers, showing an initial search pattern (seeker trajectories; blue) and predicted failure modes (hider trajectories; red). (Right) HW results for an optimized search pattern leaves fewer hiding places. The top row shows the predicted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

butions. This reformulation brings three key advantages. First, it allows us to use gradient-based sampling algorithms that are robust to high-variance or poorly conditioned gradients that would otherwise cause a gradient-based optimization algorithm to diverge. Second, our stochastic sampling-based methods is able to avoid local minima that cause gradient-based optimizers to get stuck. Finally, since our method focused on sampling a range of high-severity failures, rather than finding a single most-severe failure, we are able to find a much more diverse set of counterexamples than prior optimization-based methods. Ultimately, these diverse counterexamples allow us to optimize against a more challenging curriculum of failure modes, leading to more 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 [31] and rendering with occlusion [124]) 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 $R_1$ will fall to zero, which provides 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.

# 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 $Q \in \mathbb{R}^{3\times3}$). 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 $R \in \mathbb{R}^{3\times3}$), modeling range measurements from radio or acoustic beacons $b_1$ and $b_2$ and inertial or magnetic measurements of $\theta$. 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 $P_0 \in \mathbb{R}^{3\times3}$. The navigation function (shown in Fig. 3.3a) 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 $\omega_d$, 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 [37], where the normal force is given by $f_n = k_c \min(\phi, 0) - k_d \dot{\phi} \mathbb{1}_{\phi<0}$ ($\phi$ 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 ($\mu_d$ varies for each contact pair), $\psi$ 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 $\omega_d$ 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 | $\theta = [b_1, b_2, k] \in \mathbb{R}^6$<br>Beacon locations: $b_i = (x_{bi}, y_{bi}) \in \mathbb{R}^2$ for $i = 1, 2$<br>Feedback gains: $k \in \mathbb{R}^2$ |
| Exogenous parameters | $\phi = [q_0, w_0, \ldots, w_{T-1}, v_0, \ldots, v_{T-1}] \in \mathbb{R}^{3+6T}$<br>Initial state: $q_0 \in \mathbb{R}^3,\ q_0 \sim \mathcal{N}(\bar{q}_0, P_0)$;<br>$P_0 = 0.001 I_{3\times 3}$<br>Actuation noise: $w_t \in \mathbb{R}^3,\ w_t \sim \mathcal{N}(0, Q)$;<br>$Q = (\Delta t)^2 \text{diag}([0.001, 0.001, 0.01])$<br>Measurement noise: $v_t \in \mathbb{R}^3,\ v_t \sim \mathcal{N}(0, R)$<br>$R = \text{diag}([0.1, 0.01, 0.01])$ |
| 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<br>1. Evaluates the navigation function to find a collision-free path to the goal,<br>2. Uses a feedback controller to track that path,<br>3. Updates the state using forward Euler integration,<br>4. Performs an EKF prediction, obtains a measurement $z_t$, and performs an EKF update.<br>$S$ 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: $J = \frac{1}{T}\sum_{t=1}^{T}\left(100\|q_t - \hat{q}_t\|^2 + \|q_t\|^2 + 0.1V_t\right) + 0.1\max_t V_t$ |
| 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 | $\theta = [k_v, k_\omega, w_i, b_i] \in \mathbb{R}^{2+n_p}$<br>Trajectory tracking gains: $[k_v, k_w] \in \mathbb{R}^2$<br>Network weights and biases: $(w_i, b_i)$ for $i = 1, \ldots, n_p$ |
| Exogenous parameters | $\phi = [\mu_{rg}, \mu_{bg}, \mu_{br}, m_b, p_{bd}, p_{r1}, p_{r2}] \in \mathbb{R}^{13}$<br>Robot/ground, box/ground, box/robot coefficients of friction:<br>$[\mu_{rg}, \mu_{bg}, \mu_{br}] \in [0.6, 0.8] \times [0.4, 0.6] \times [0.1, 0.3]$<br>Box mass: $m_b \in [0.9, 1.1]$<br>Desired box pose:<br>$p_{bd} = [x_d, y_d, \theta_d] \in [0, 0.5]^2 \times [-\pi/4, \pi/4]$<br>(Above parameters are uniformly distributed)<br>Initial robot pose: $p_{ri} = [x_0, y_0, \theta_0] \sim \mathcal{N}(\bar{p}_{ri}, \Sigma)$;<br>$\Sigma = 0.01 I_{3\times 3}$, $i = 1, 2$. |
| Simulator | $S$ initializes the robots at the initial states in $\phi$ 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 $\sigma$ 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 $\mathcal{S}$ solves the nonlinear AC power flow equations [118] 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 $\xi$ is the concatenation of these values). We follow the 2-step method described in [122] 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 [125].

**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 $\phi$ and $10^{-6}$ for $\theta$, 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 [123]. 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 ($3.2\,\mathrm{m} \times 2\,\mathrm{m}$).

**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

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

where $r$ is the sensing range of the seekers (0.5 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); $t_0, \ldots, t_n$ 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.

**Hyperparameters** Prediction and repair experiments were run for $N = 100$ rounds of $K = 50$ steps with $n = 10$, learning rate $\lambda = 10^{-2}$ for $\phi$ and $\theta$, 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 3-layer fully-connected neural network with tanh 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 $\phi$ and $\theta$ 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 highlightling the "graspable" regions of the object (this is a common strategy in robot manipulation). $\theta$ includes all parameters of the autoencoder, and the environmental parameters $\phi$ 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-labelling 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 $\phi$ and $10^{-5}$ for $\theta$, 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). $\theta$ includes all parameters of the policy, and the environmental parameters $\phi$ 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 $\phi$ and $\theta$, 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, $\theta$ includes all trainable parameters of the policy network.

### B.1.7 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 $\phi$ 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 $\phi$ and $2 \times 10^{-5}$ for $\theta$, with tempering parameter $\alpha = 20$. Quenching was not used on this problem.

### B.1.8 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 straigt 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 $\phi$ and $10^{-4}$ for $\theta$, with tempering parameter $\alpha = 20$. Quenching was not used on this problem.

# References


[1] C. Belta and S. Sadraddini, "Formal Methods for Control Synthesis: An Optimization Perspective," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, no. 1, pp. 115–140, 2019. DOI: 10.1146/annurev-control-053018-023717. (visited on 02/07/2023).

[2] H. Kress-Gazit, M. Lahijanian, and V. Raman, "Synthesis for Robots: Guarantees and Feedback for Robot Behavior," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 211–236, 2018. DOI: 10.1146/annurev-control-060117-104838. (visited on 02/07/2023).

[3] 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).

[4] 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).

[5] 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).

[6] S. Posada, "EA 22-002: Autopilot & First Responder Scenes," *US DOT NHTSA*,

[7] 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).

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

[9] 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).

[10] 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.

[11] 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).

[12] 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).

[13] 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.

[14] 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).

[15] 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).

[16] 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.

[17] 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.

[18] 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).

[19] 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).

[20] 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).

[21] 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.

[22] 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).

[23] 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).

[24] 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.

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

[26] 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).

[27] 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.

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

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

[30] 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).

[31] 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).

[32] 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).

[33] 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.

[34] 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.

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

[36] 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.

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

[38] 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).

[39] 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).

[40] 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).

[41] 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).

[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] 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.

[44] 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).

[45] 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).

[46] 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).

[47] 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).

[48] M. Cusumano-Towner, A. K. Lew, and V. K. Mansinghka, *Automating Involutive MCMC using Probabilistic and Differentiable Programming*, Jul. 2020. DOI: 10.48550/arXiv.2007.09871. arXiv: 2007.09871 [stat]. (visited on 11/10/2022).

[49] 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).

[50] 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).

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] J. R. R. A. Martins and A. B. Lambe, "Multidisciplinary design optimization: A survey of architectures," *AIAA Journal*, vol. 51, no. 9, pp. 2049–2075, Sep. 2013. DOI: 10.2514/1.J051895.

[59] 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.

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

[61] 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.

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

[63] M. Althoff, G. Frehse, and A. Girard, "Set propagation techniques for reachability analysis," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 369–395, 2021. DOI: 10.1146/annurev-control-071420-081941. eprint: https://doi.org/10.1146/annurev-control-071420-081941.

[64] 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).

[65] C. Dawson, Z. Qin, S. Gao, and C. Fan, "Safe Nonlinear Control Using Robust Neural Lyapunov-Barrier Functions," in *Proceedings of the 5th Conference on Robot Learning*, PMLR, Jan. 2022, pp. 1724–1735. (visited on 05/23/2023).

[66] C. Fan, B. Qi, S. Mitra, and M. Viswanathan, "DryVR: Data-driven verification and compositional reasoning for automotive systems," in *Computer Aided Verification*, R. Majumdar and V. Kun**c**ak, Eds., Cham: Springer International Publishing, 2017, pp. 441–461, ISBN: 978-3-319-63387-9.

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

[68] 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.

[69] 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.

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

[71] 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.

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

[73] 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.

[74] 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).

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

[76] 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.

[77] 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.

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

[79] 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.

[80] 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).

[81] 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).

[82] 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).

[83] 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).

[84] 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).

[85] 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).

[86] 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).

[87] 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.

[88] 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.

[89] 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).

[90] 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).

[91] 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).

[92] 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.

[93] 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).

[94] 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.

[95] 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).

[96] 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.

[97] 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).

[98] 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).

[99] 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.

[100] L. Jain, V. Chandrasekaran, U. Jang, W. Wu, A. Lee, A. Yan, S. Chen, S. Jha, and S. A. Seshia, *Analyzing and Improving Neural Networks by Generating Semantic Counterexamples through Differentiable Rendering*, Jul. 2020. DOI: 10.48550/arXiv.1910.00727. arXiv: 1910.00727 [cs, stat]. (visited on 03/17/2023).

[101] 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).

[102] S. Yaghoubi and G. Fainekos, "Gray-box Adversarial Testing for Control Systems with Machine Learning Component," *HSCC 2019 - Proceedings of the 2019 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 179–184, Dec. 2018. DOI: 10.48550/arxiv.1812.11958. arXiv: 1812.11958. (visited on 10/18/2022).

[103] 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).

[104] 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).

[105] 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).

[106] 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).

[107] 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).

[108] 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).

[109] 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).

[110] 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).

[111] 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.

[112] 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).

[113] 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).

[114] 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).

[115] 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).

[116] 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.

[117] 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).

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

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

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

[121] 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).

[122] 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).

[123] 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.

[124] 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).

[125] 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.