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

Title page **1**

Abstract **3**

Acknowledgments **5**

List of Figures **9**

List of Tables **11**

1 Introduction & Objectives **13**
1.0.1 Impact . . . 14
1.0.2 Outline . . . 14

2 Background and Significance **15**
2.0.1 Safety verification . . . 15
2.0.2 Programs as mathematical models . . . 18

3 General-purpose design optimization with statistical robustness certificates **23**
3.1 Related Work . . . 25
3.2 Preliminaries and Assumptions . . . 26
3.3 Design Optimization . . . 28
3.4 Design Certification via Robustness Analysis . . . 30
3.4.1 Estimating the worst-case performance . . . 31
3.4.2 Estimating sensitivity . . . 32
3.4.3 Connections to Design Optimization . . . 33
3.5 Experimental Results . . . 33
3.5.1 Case study: optimal sensor placement for navigation . . . 34
3.5.2 Case study: collaborative multi-robot manipulation . . . 35
3.5.3 Design optimization ablation study . . . 38
3.5.4 Accuracy of robustness analysis . . . 40
3.6 Discussion . . . 41

4 Improving robustness using formal specifications and counterexample-guided optimization **43**
4.1 Background on signal temporal logic . . . 45
4.1.1 Syntax of signal temporal logic . . . 46
4.1.2 Semantics of signal temporal logic . . . 46
4.2 Problem statement . . . 47
4.3 Approach . . . 48
4.3.1 Differentiable signal temporal logic . . . 49
4.3.2 Counterexample-guided optimization . . . 49
4.4 Experiments . . . 52
4.4.1 Satellite rendezvous problems . . . 52
4.4.2 Baselines . . . 54
4.4.3 Results . . . 54

A Appendix to Chapter 3 **57**
A.1 Sensor Placement Design Problem Statement . . . 57
A.2 Multi-agent Manipulation Design Problem Statement . . . 58

References **61**

List of Figures

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. 27

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. 28

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. 37

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. 38

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. 39

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). 42

4.1 TODO An illustration of robot behaviors as continuous-time signals. 45

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. 53
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. 55
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. 56
4.5 Optimized trajectories found using our method for the two satellite rendezvous missions. 56

List of Tables

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. 40
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. 41

A.1 Formal statement of the sensor placement design problem with T discrete timesteps. 59
A.2 Formal statement of the collaborative manipulation design problem using a planning network with n_p total parameters (weights and biases). 60

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 ξ, and a specification $\psi \colon \Xi \to \mathbb{R}$ that determines whether a trace satisfies ($\psi(\xi) \geq 0$) or does not satisfy ($\psi(\xi) < 0$) the safety constraints. In this context, the falsification problem involves solving an optimization for a counterexample

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

which is commonly reparameterized as a minimization problem to find the worst 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 δx 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 δx. As the function is computed and each primitive operation is carried out, the tangent value is updated using hand-derived derivative rules for each operation. The benefit of forward-mode AD is that its memory usage is constant with respect to the number of operations used to define f (roughly double the memory usage of 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 δy 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

General-purpose 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

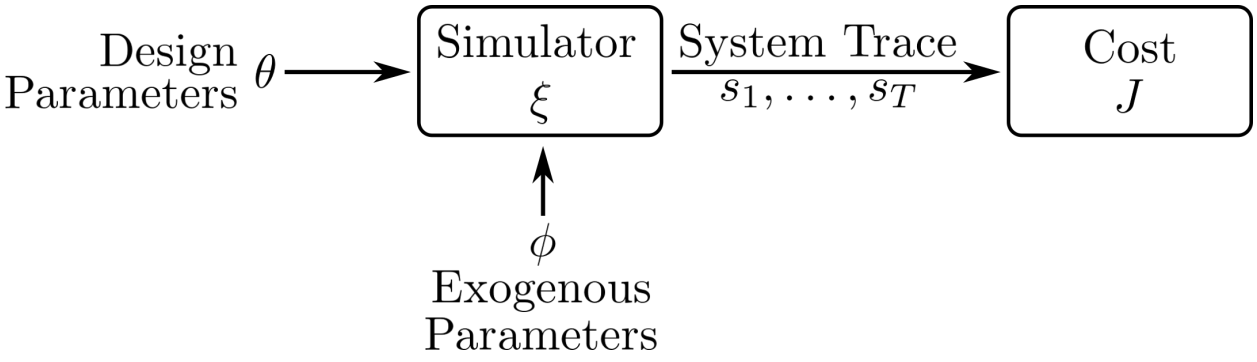


Figure 3.1: A glass-box model of a generic robotic system. Design optimization involves finding a set of design parameters so that the simulated cost is minimized, while robustness analysis involves quantifying how changes in the exogenous parameters affect the simulated cost.

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

Simulator

Given particular choices for θ and ϕ, the system's state $s \in \mathcal{S}$ evolves in discrete time according to a known simulator $S : \Theta \times \Phi \mapsto \mathcal{S}^T$. 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 θ are feasible if $c_i(\theta) \geq 0 \; \forall i \in \mathcal{I}_c$. Here, we consider constraints as functions of θ only; we leave the extension to robust constraints involving ϕ to future work.

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

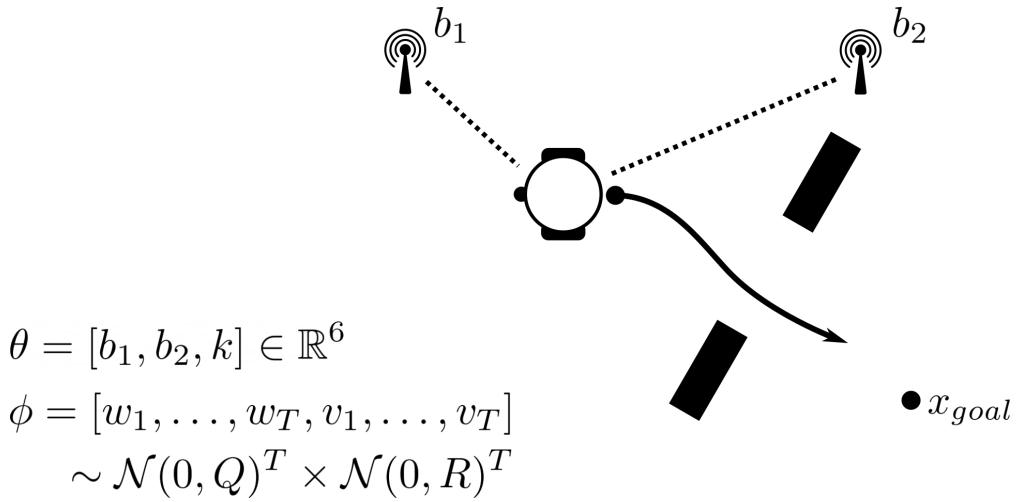


Figure 3.2: A design optimization problem for an AGV localization and navigation system. The goal is to find placements for two range sensors along with parameters for the navigation system that allow the robot to safely pass through the narrow doorway.

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 θ include the (x, y) locations of the two range beacons $b_1, b_2 \in \mathbb{R}^2$ and the feedback controller gains $k \in \mathbb{R}^2$. The exogenous parameters ϕ are the actuation and sensor noises at each timestep $w_t \in \mathbb{R}^3$ and $v_t \in \mathbb{R}^3$, drawn i.i.d. from Gaussian distributions $\mathcal{N}(0, Q)$ and $\mathcal{N}(0, R)$, respectively, as well as the initial state (also Gaussian). The simulator ξ integrates the AGV's dynamics using a fixed timestep, updating the EKF and evaluating the navigation controller at each step. The cost function J assigns a penalty to collisions with the environment, estimation errors, and deviations from the goal location. We will return to this example in more detail in Section 3.5.1; first, we discuss our approach to design optimization and robustness analysis in Sections 3.3 and 3.4, respectively.

3.3 Design Optimization

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 ϕ. Simply minimizing the expected value of the cost $\mathbb{E}_{\phi \sim \Phi}\left[J \circ S\left(\theta, \phi\right)\right]$ (where $\circ$ denotes composition) can lead to myopic behavior where

exceptional performance for some values of ϕ compensates for poor performance on other values; this is related to the phenomenon of "reward hacking" in reinforcement learning [68].

Ideally, we would like our designs to be robust to variations in exogenous parameters: changing ϕ should not cause the performance to change much. We can include this requirement as a heuristic by penalizing the variance of J. Intuitively, this heuristic "smooths" the cost function with respect to the exogenous parameters: regions of high variance (containing sharp local minima) are penalized, while regions of low variance are rewarded. We return to justify this connection to robustness in Section 3.4.3. 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 θ, 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 θ, then algorithms like L-BFGS-B may be used, but if the constraints are more complex then sequential quadratic programming or interior-point methods may be used. Our implementation provides an interface to a range of optimization back-ends through SciPy [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 μ, scale σ, and shape ξ,

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

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

In the special case $\xi = 0$, this distribution has a slightly different form (known as a Gumbel distribution), but the result holds. In practice, a_n and b_n are not estimated directly (this merely changes the fit values of μ and σ) and the GEVD is fit directly to M_N by either minimizing the log likelihood [72] or estimating the posterior distribution of (μ, σ, ξ) 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_∞. If $\xi \geq 0$, then we cannot estimate a strict upper bound, but we can provide for a confidence interval for M_∞ instead. In the following sections, we apply this theorem to analyze the robustness of an optimized design.

3.4.1 Estimating the worst-case performance

Our first robustness question concerns the worst-case performance of our design: given variation in ϕ, what is the maximum cost[1] we can expect for our choice of design parameters θ? Our insight is that the variation $\phi \sim \Phi$ induces an (unknown) distribution in $J \circ S(\theta, \phi)$, so $J \circ S(\theta, \phi)$ a random variable to which the extremal types theorem applies. Algorithm 3.4.1 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 μ, σ, and ξ using Markov Chain Monte Carlo sampling with the PyMC3 library [73]. From this posterior distribution, we take the 97% confidence level for each parameter (μ^*, σ^*, ξ^*). If $\xi^* < 0$, we have confidence that the corresponding GEVD has bounded support on the right and estimate the maximum cost $J_{max} \leq \mu - \sigma/\xi$. Otherwise, we can estimate the 97% confidence level for J_{max} using the GEVD described by (μ^*, σ^*, ξ^*).

[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 ϕ), then how much can we expect its performance to degrade if ϕ changes? Formally, we define the sensitivity L as the least constant such that for any two $\phi_1, \phi_2 \sim \Phi$,

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

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

In both cases, we can exploit the fact that L is an extreme value of the slope $|J \circ S(\theta, \phi_1) - J \circ S(\theta, \phi_2)|/||\phi_1 - \phi_2||$ and apply the extremal types theorem. Let $X = ||J \circ S(\theta, \phi_1) - J \circ S(\theta, \phi_2)||/||\phi_1 - \phi_2||$ be a random variable with $\phi_1, \phi_2 \sim \Phi$. The distribution of X is unknown, but the extremal types theorem lets us characterize the sample maximum $L_N = \max\{X_1, \ldots, X_N\}$ 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 ξ^* is negative, then $J \circ S$ is likely Lipschitz continuous with Lipschitz constant $L \leq \mu - \frac{\sigma}{\xi}$. If $\xi > 0$, then $J \circ S$ is likely not Lipschitz but we can estimate the 97% confidence level for L. As a result, this statistical approach allows us to avoid making prior assumptions about the continuity of our system.

3.4.3 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 α 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 ϕ 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 ϕ. 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.

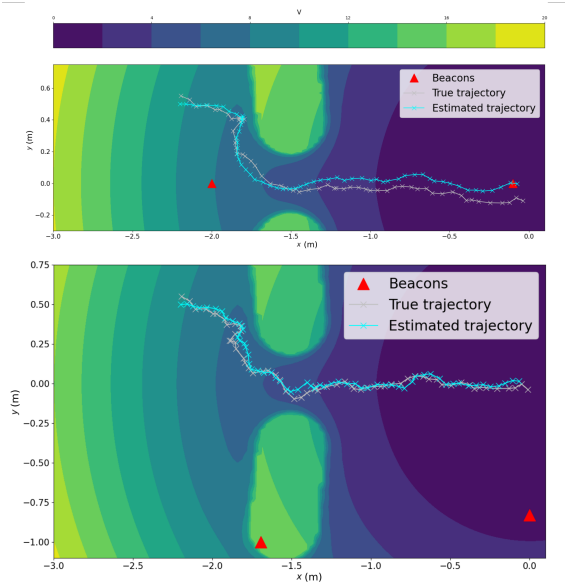
3.5.1 Case study: optimal sensor placement for navigation

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.

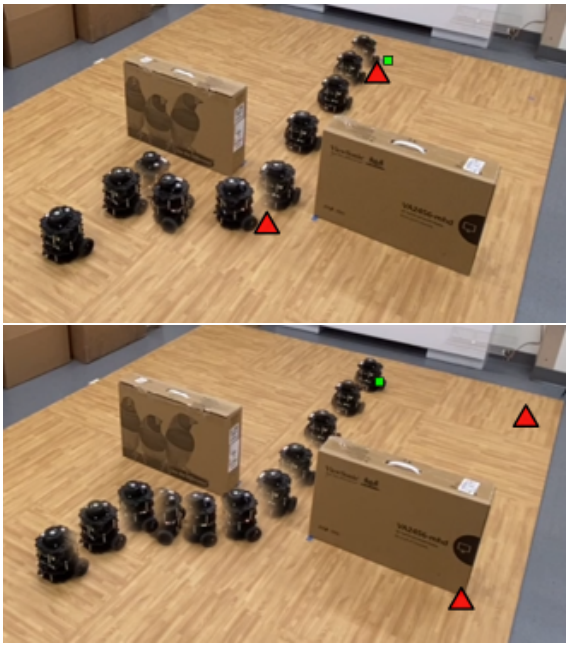
The design parameters are the (x, y) locations of two range beacons and two feedback controller gains (6 total design parameters). The exogenous parameters include uncertainty in the robot's initial state along with actuation and sensing noise at each of T timesteps ($3 + 6T$ total exogenous parameters). 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.



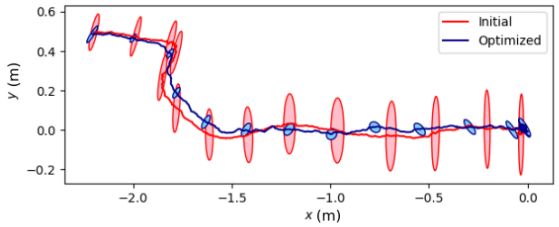
(b) Hardware performance of initial (top) and optimized (bottom) AGV designs. Square (green) shows the goal; triangles (red) show beacon locations. The optimized design eliminates drift relative to goal.

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.

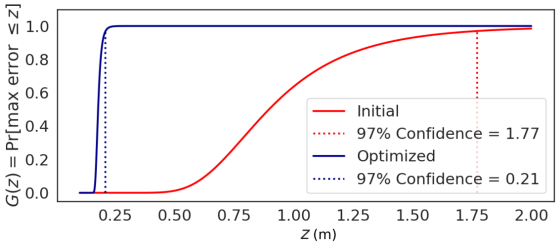
Finally, we apply the robustness analysis from Section 3.4 to certify the maximum absolute estimation error $\|x_t - \hat{x}_t\|$ in the optimized design (in meters, projected into the xy plane). Note that this error is different from the cost used during optimization, but we can still apply Algorithm 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

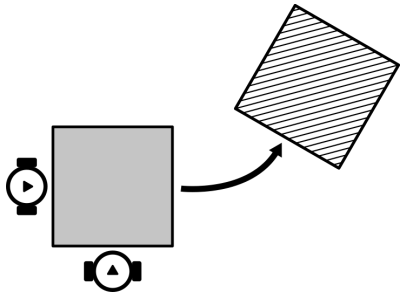
Our second example involves finding a control strategy for multi-agent collaborative manipulation. In this setting, two ground robots must collaborate to push a box from its current location to a target pose (as in Fig. 3.5a). Given the desired box pose and the current loca-



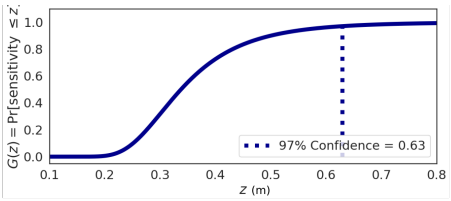
(a) Hardware results for EKF state estimates and posterior error covariance 3σ ellipse for initial and optimized AGV designs.



(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 (θ includes both the neural network parameters and the tracking controller gains, with a total of 454 design parameters). The exogenous parameters include the coefficient of friction for each contact pair, the mass of the box, the desired pose of the box, and the initial pose for each robot (a total of 13 exogenous parameters; we vary the desired box pose and initial robot poses to prevent over-fitting during optimization). The cost function is simply the squared error between the desired box pose (including 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

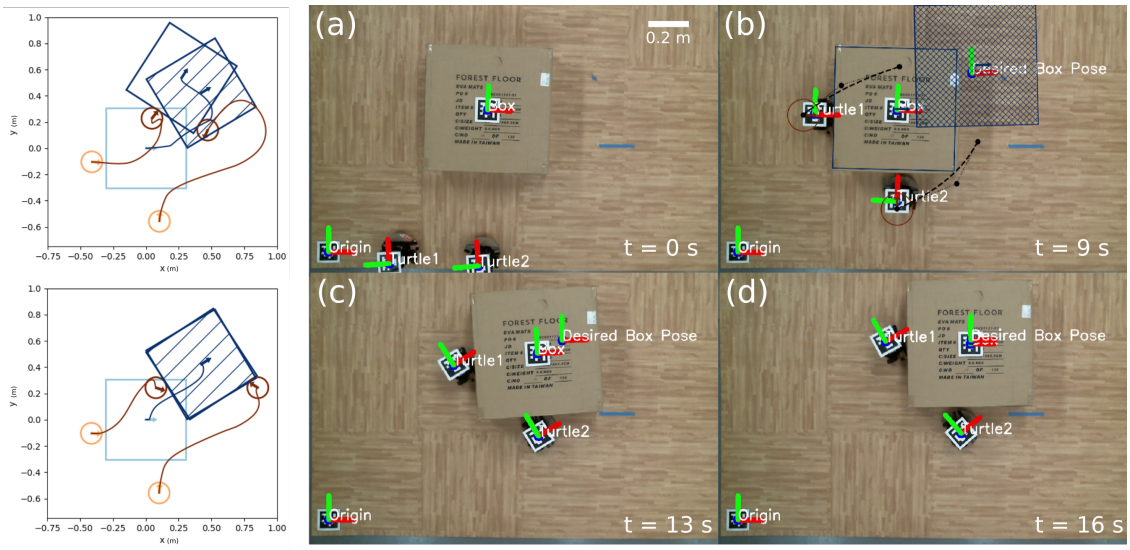


Figure 3.6: Left: Initial (top) and optimized (bottom) manipulation strategies in simulation (light/dark colors indicate initial/final positions, stripes indicate desired position). Right: Optimized manipulation strategy deployed in hardware (video included in the supplementary materials). (a) The robots 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 ϕ. As a result, minimizing the expected cost with respect to variation in ϕ yields a solution that achieves good performance for many different target poses, enabling the user to select one at run-time and be confident that the design will perform well.

To solve this design problem, the neural network parameters are initialized i.i.d. according to a Gaussian distribution, and the tracking controller gains are set to nominal values. We then optimize the parameters using $N = 512$, $\lambda = 0.1$, and L-BFGS-B back-end. This optimization took 45 minutes 32 s on a laptop computer (8 GB of RAM and a 1.8 GHz 8-core processor). Fig. 3.6 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

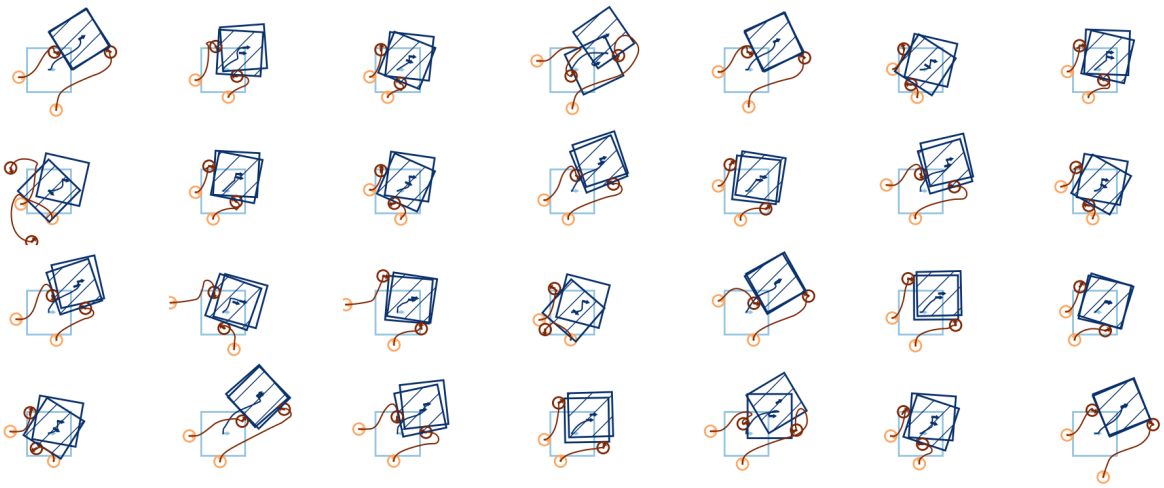


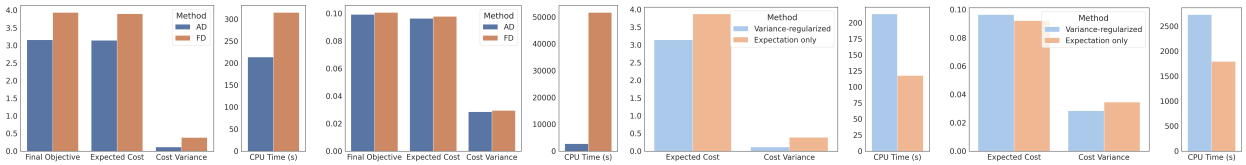
Figure 3.7: Additional examples of optimized multi-agent manipulation behavior in simulation, showing that the optimized strategy reaches the goal in most cases. Each example shows the results of executing the optimized pushing strategy for 4 s with a randomly selected set of friction coefficients, random 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 ϕ at nominal values (box mass 1 kg and target pose $[0.3, 0.3, 0.3]$). The fit distribution is shown in Fig. 3.5b. 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 μ, σ, and ξ, allowing us to construct an upper-bound and lower-bound GEVD using the 97% and 3% confidence level parameter estimates. Using these distributions, we can measure false optimism and conservatism using a one-sided Kolmogorov-Smirnov (KS) test [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	6.75×10^{-5}	Reject; 97% GEVD *does not* under-estimate sensitivity
Conservatism	3% GEVD over-estimates sensitivity	0.0618	1.03×10^{-10}	Reject; 3% GEVD *does not* over-estimate sensitivity

Table 3.2: Results of one-sided KS tests for the collaborative manipulation case study. These results indicate that Algorithm 3.4.2 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 Φ 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 Φ 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.

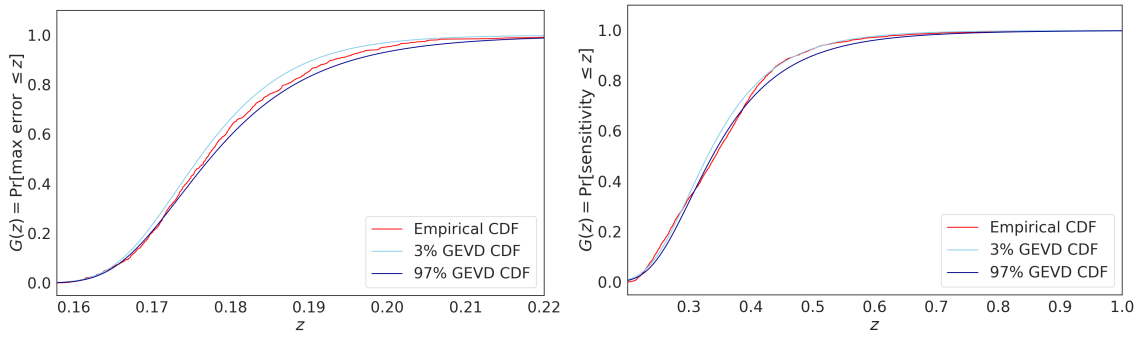


Figure 3.9: Comparison of fit GEVD CDFs and empirical CDF for worst-case estimation error in the sensor placement example (top) and sensitivity in the manipulation example (bottom).

Chapter 4

Improving robustness using formal specifications and counterexample-guided optimization

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 ϕ. 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 ϕ that lead to high cost). This chapter is based on the author's published work [79].

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 θ and a set of adversarial counterexamples ϕ. 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. There are different types of temporal logic, 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 ϕ) to guide the optimization of the design parameters θ. This method uses an iterative adversarial optimization scheme inspired by solution methods for multi-player games, alternating between searching for a design that performs well against a set of known counterexamples and searching for new counterexamples to guide the design process. This approach relies on differentiable simulation, as discussed in Chapter 3, with the addition of differentiable temporal logic 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

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

numerical experiments that demonstrate the improved performance of our method relative to state-of-the-art mixed-integer and nonlinear optimization methods. We find that our approach not only yields designs that satisfy the STL specification despite worst-case environmental uncertainty, but also that it requires less than half of the optimization time as the next-most-successful method. Our counterexample-guided method scales to handle long-horizon tasks that are not tractable for MIP-based methods, and the designs found using our approach are consistently more robust than those generated by competing methods.

4.1 Background on signal temporal logic

Recall from Chapter 3 that we described the behavior of the system by the discrete trace of states over a fixed horizon T: $s_1, \ldots, s_T$. 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 ψ recursively as:

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

with closed (but potentially unbounded) time interval I, predicate $\mu : \mathcal{S} \mapsto \mathbb{R}$ 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, ψ_1 must be true until ψ_2 becomes true". For convenience, we assume $I = [0, \infty)$ when not explicitly specified. Additional temporal operators can be defined in terms of these basic building blocks:

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

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

4.1.2 Semantics of signal temporal logic

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

that time [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 ψ at time t. The robustness is a real number that quantifies how well the signal satisfies the formula at that time; the formula is satisfied precisely when $\rho \geq 0$. The robustness function is defined recursively as:

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

where $\top$ is a constant defined to be greater than all other real values. In practice, linear-time algorithms exist to compute ρ from a given piecewise-affine signal s [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 θ and ϕ with respect to STL specification ψ. We subtract ρ from a generic cost function J (weighted by $\lambda > 0$) to balance maximizing STL robustness against minimizing other costs (e.g. fuel use). The scaling factor λ is typically small to prioritize satisfying the STL specification. For convenience, we denote

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

In this adversarial formulation, we 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 θ and corresponding adversarial exogenous parameters ϕ 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 ρ with respect to θ and ϕ; 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 θ to a particular adversarial ϕ so that the design performs poorly when ϕ 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 ρ, 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, ρ is a continuous but non-smooth function of s, t, and all parameters of the formula ψ. Smooth approximations to ρ can be derived by replacing the min and max operators with smooth approximations:

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

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

where $\gamma > 0$ controls the degree of smoothing; $\lim_{\gamma \to \infty} \widetilde{\min}_\gamma(a, b) = \min_\gamma(a, b)$. This approximation was introduced in [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 θ should perform well against a wide range of adversarial exogenous parameters ϕ. A common approach to finding Nash equilibria in two-player games like (4.2) is the family of nonlinear Gauss-Seidel-type methods [95], which solve min-max problems alternating between optimizing θ and ϕ. These methods tune each set of parameters in turn while holding the other constant; i.e. solving a sequence of optimization

problems:

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

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

If this sequence converges (which is not guaranteed), then the stationary point $()\theta^*, \phi^*)$ will be a local Nash equilibrium [95].

The risk in using this simple iterative algorithm is that the optimization for both θ and ϕ can get stuck in local minima. These local minima not only lead to suboptimal performance of the design θ^*, but also "overconfidence" when θ^* is overfit to a particular value of ϕ^*, which might cause the designer to believe that the design is robust because it performs well against a single adversarial example when it may fail for nearby values of ϕ. 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 θ against a single ϕ, we can maintain a dataset $\mathcal{D}_N = \{\phi_i\}_{i=1}^N$ of exogenous parameters, and optimize θ against all of them simultaneously.

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

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

This approach encourages the design to be robust to a wide range of adversarial exogenous parameters, rather than overfitting to a single value of ϕ. 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 ϕ^*_i from successive iterations of the alternating Gauss-Seidel process as adversarial counterexamples to guide the optimization of θ.

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. ϕ sampled randomly from Φ, then alternates between

solving the two optimization problems in (4.7a). At each step, the adversary's current best counterexample ϕ^* is added to the dataset, and the process continues until either reaching a fixed point where the best counterexample does not change between iterations or a maximum number of steps is reached. As we demonstrate empirically in Section 4.4, this counterexample-guided optimization achieves substantially better sample efficiency than simple domain randomization, in that it finds designs θ^* that are more robust to disturbances despite using fewer samples of ϕ.

An important note is that Algorithm 4.3.1 is enabled by the automatic differentiation approach described in Section 4.3.1. Without access to the gradients of J_ψ, solving the subproblems on lines 4 and 5 would be much more difficult. Although previous works have used gradients of STL robustness with respect to θ, we are not aware of any prior work using gradients with respect to disturbance parameters ϕ for robust optimization. Some prior works have used counterexamples to guide mixed-integer solvers to plan from STL specifications [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 θ^*
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 ϕ^* to $\mathcal{D}_N$
10 **return** θ^*, $\mathcal{D}_N$

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 Satellite rendezvous 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

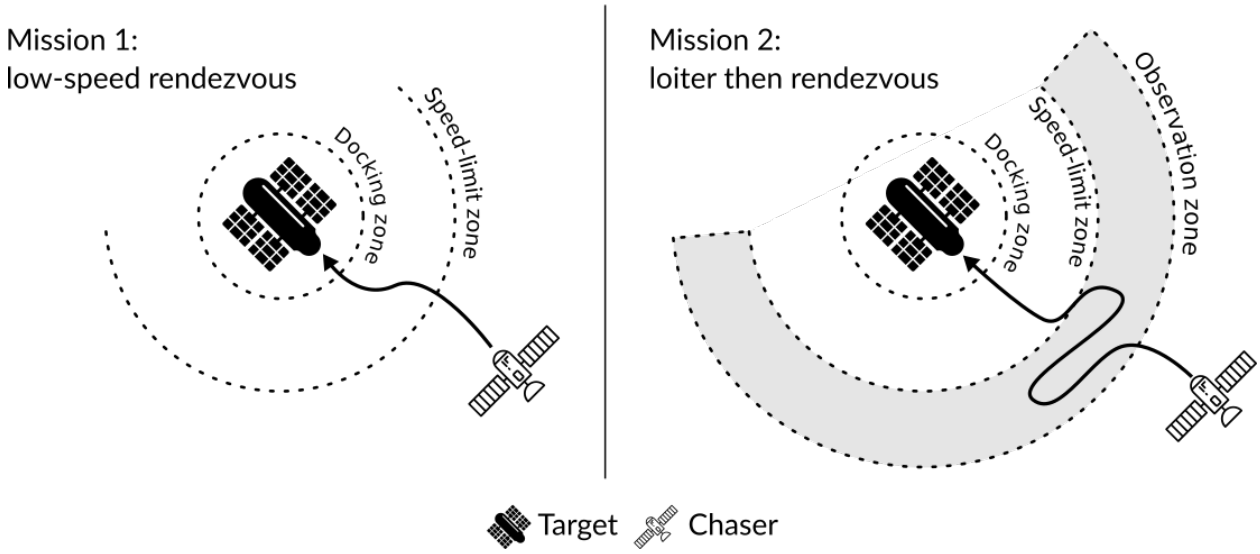


Figure 4.2: Two satellite rendezvous missions used in our experiments. In the first mission, the chaser satellite must eventually reach the target while respecting a maximum speed constraint in the region immediately around the target. In the second mission, the chaser must still reach the target and 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, ψ_1 and ψ_2:

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

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

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

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

4.4.2 Baselines

We compare our approach against two baselines: an MIP planner based on that in [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 ϕ. 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.

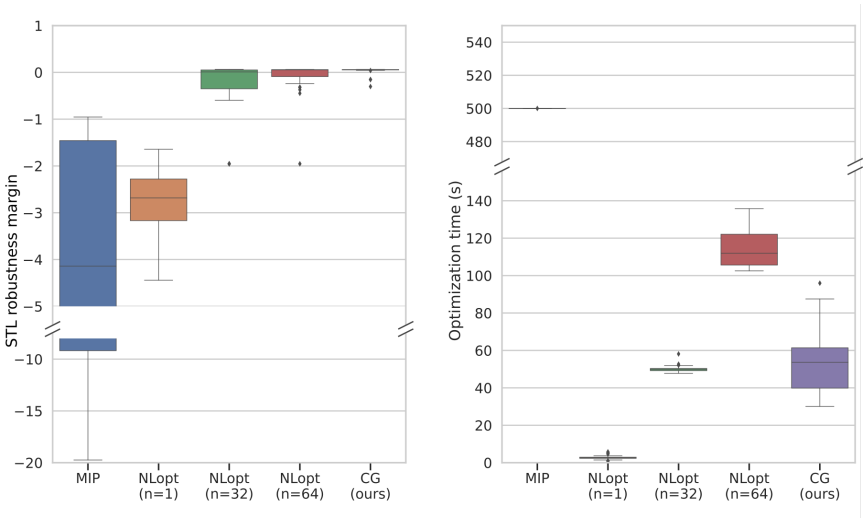


Figure 4.3: Comparison of different STL planning methods on the first example mission, averaged over 50 random seeds. Left: the robustness margin $\rho(\psi_1)$ computed for the optimized design parameters and worst-case exogenous parameters. Right: the planning time required by each method. Our method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

We find that

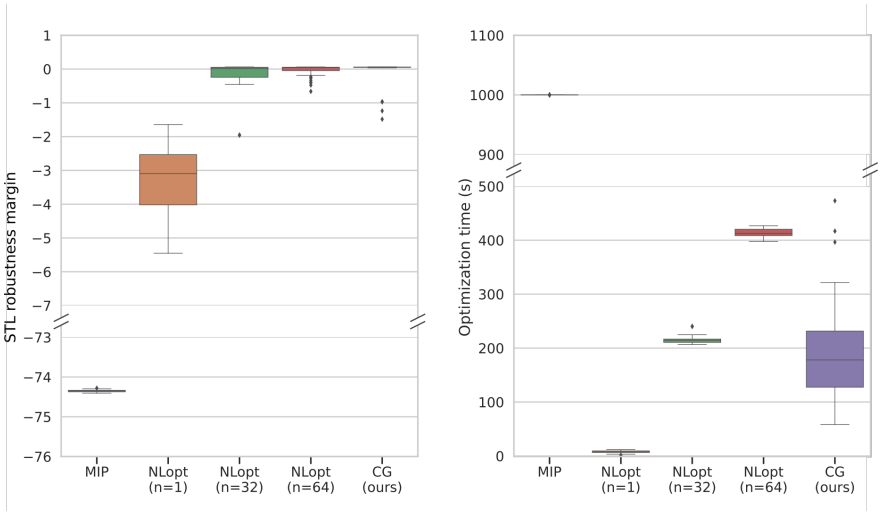
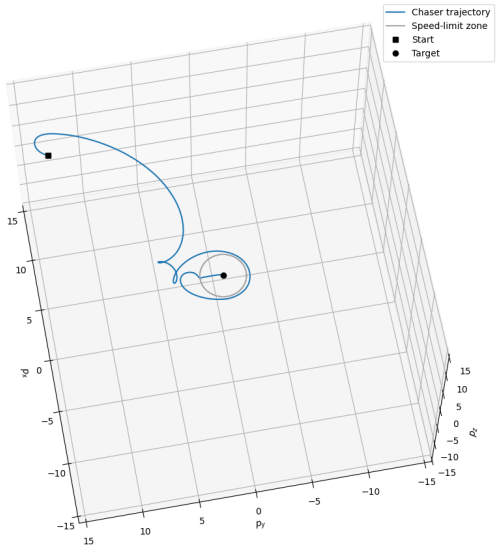
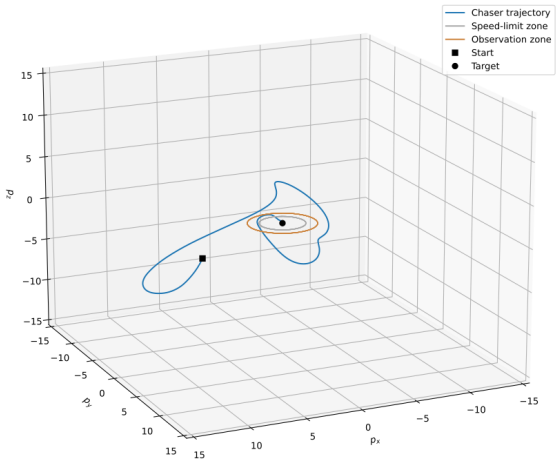


Figure 4.4: Comparison of different STL planning methods on the second example mission, averaged over 50 random seeds. Left: the robustness margin $\rho(\psi_2)$ computed for the optimized design parameters and worst-case exogenous parameters. Right: time required by each method to find a plan. Our method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



(a) Mission 1 (rendezvous with speed constraint).



(b) Mission 2 (loiter then rendezvous with speed constraint).

Figure 4.5: Optimized trajectories found using our method for the two satellite rendezvous missions.

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 θ. The initial state of the robot is normally distributed $q_0 \sim \mathcal{N}(\bar{q}_0, P_0)$ for mean initial state $\bar{q}_0 \in \mathbb{R}^3$ and initial covariance $P_0 \in \mathbb{R}^{3\times3}$. 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 ω_d, the robot tracks those desired velocities by applying forces and torques subject to a friction cone constraint. The box is modeled as a rigid body with friction against the ground. Contact forces between the box and each robot are modeled using a penalty method described in [37], where the normal force is given by $f_n = k_c \min(\phi, 0) - k_d \dot{\phi} \mathbb{1}_{\phi<0}$ (ϕ is the signed distance between the robot and the box, $k_c = 300\,\mathrm{N/m}$ is the contact stiffness, k_d is a damping coefficient chosen to ensure critical damping, and $\mathbb{1}_{\phi<0}$ is the indicator function equal to 1 when the box and robot are in contact and 0 otherwise). Friction in the box/ground and box/robot contacts was modeled as Coulomb friction, resulting in a tangential force $f_t = \mu f_n$ with $\mu = c\psi$ if $\psi < \psi_s$ and $\mu = \mu_d$ otherwise, where mu_d is the coefficient of dynamic friction (μ_d varies for each contact pair), ψ is the tangential velocity at the point of contact, $\psi_s = 0.3\,\mathrm{m/s}$ is the tangential velocity where slipping begins, and $c = \mu_d/\psi_s$ was chosen to ensure a continuous friction model.

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

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

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

Design parameters	$\theta = [b_1, b_2, k] \in \mathbb{R}^6$ Beacon locations: $b_i = (x_{bi}, y_{bi}) \in \mathbb{R}^2$ for $i = 1, 2$ Feedback gains: $k \in \mathbb{R}^2$
Exogenous parameters	$\phi = [q_0, w_0, \ldots, w_{T-1}, v_0, \ldots, v_{T-1}] \in \mathbb{R}^{3+6T}$ Initial state: $q_0 \in \mathbb{R}^3,\ q_0 \sim \mathcal{N}(\bar{q}_0, P_0)$; $P_0 = 0.001 I_{3\times 3}$ Actuation noise: $w_t \in \mathbb{R}^3,\ w_t \sim \mathcal{N}(0, Q)$; $Q = (\Delta t)^2 \text{diag}([0.001, 0.001, 0.01])$ Measurement noise: $v_t \in \mathbb{R}^3,\ v_t \sim \mathcal{N}(0, R)$ $R = \text{diag}([0.1, 0.01, 0.01])$
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 1. Evaluates the navigation function to find a collision-free path to the goal, 2. Uses a feedback controller to track that path, 3. Updates the state using forward Euler integration, 4. Performs an EKF prediction, obtains a measurement z_t, and performs an EKF update. S returns a trace $s_t = [q, \hat{q}, P_{t\|t}, V_t]$ containing true states, estimated states, estimated posterior error covariance, and the value of the navigation function at each time step.
Cost	J has three components. The first ($\|q_t - \hat{q}_t\|^2$) minimizes the estimation error of the EKF, the second ($\|q_t\|$) guides the robot towards the goal, and the third (both V_t terms) avoids collision with the environment: $J = \frac{1}{T}\sum_{t=1}^{T}\left(100\|q_t - \hat{q}_t\|^2 + \|q_t\|^2 + 0.1V_t\right) + 0.1\max_t V_t$
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	$\theta = [k_v, k_\omega, w_i, b_i] \in \mathbb{R}^{2+n_p}$ Trajectory tracking gains: $[k_v, k_w] \in \mathbb{R}^2$ Network weights and biases: (w_i, b_i) for $i = 1, \ldots, n_p$
Exogenous parameters	$\phi = [\mu_{rg}, \mu_{bg}, \mu_{br}, m_b, p_{bd}, p_{r1}, p_{r2}] \in \mathbb{R}^{13}$ Robot/ground, box/ground, box/robot coefficients of friction: $[\mu_{rg}, \mu_{bg}, \mu_{br}] \in [0.6, 0.8] \times [0.4, 0.6] \times [0.1, 0.3]$ Box mass: $m_b \in [0.9, 1.1]$ Desired box pose: $p_{bd} = [x_d, y_d, \theta_d] \in [0, 0.5]^2 \times [-\pi/4, \pi/4]$ (Above parameters are uniformly distributed) Initial robot pose: $p_{ri} = [x_0, y_0, \theta_0] \sim \mathcal{N}(\bar{p}_{ri}, \Sigma)$; $\Sigma = 0.01 I_{3\times 3}$, $i = 1, 2$.
Simulator	S initializes the robots at the initial states in ϕ relative to the box. Since these initial states may be in contact, we simulate 0.5 s of settling time at a 0.01 s timestep, then re-index the robot positions and desired box pose relative to the settled box pose. We then evaluate the planning network and track the planned path for 4 s at a 0.01 s timestep. At each timestep, 1) evaluate the spline tracking controller, 2) evaluate contact dynamics between the box, robots, and ground, and 3) integrate forces and torques to obtain box and robot states at the next timestep. S returns a trace $s_t = [q_{r1}, q_{r2}, q_b]$ containing the states of each robot and the box over time (relative to the initial pose of the box after the settling period).
Cost	J is simply the squared distance between the final box pose and the desired box position $(x - x_d)^2 + (y - y_d)^2 + (\theta - \theta_d)^2$
Constraints	Network parameters were not constrained. k_v and k_w were constrained to be less than 10.

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. Kunčak, 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).