DESIGN AND VERIFICATION OF AUTONOMOUS SYSTEMS IN THE PRESENCE OF UNCERTAINTIES

Bineet Ghosh

A dissertation submitted to the faculty at the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2023

Approved by:

Ron Alterovitz

James H. Anderson

Étienne André

Samarjit Chakraborty

Sandeep Chinchali

Parasara Sridhar Duggirala

Sriram Sankaranarayanan

ABSTRACT

Bineet Ghosh: Design and Verification of Autonomous Systems in the Presence of Uncertainties
(Under the direction of Parasara Sridhar Duggirala)

Autonomous Systems offer hope towards moving away from mechanized, unsafe, manual, often inefficient practices. The last decade has seen several small, but important, steps towards making this dream into reality. These advancements have helped us to achieve limited autonomy in several places, such as, driving, factory floors, surgeries, wearables, and home assistants, etc. Nevertheless, autonomous systems are required to operate in a wide range of environments with uncertainties (viz., sensor errors, timing errors, dynamic nature of the environment, etc.). Such environmental uncertainties, even when present in small amounts, can have drastic impact on the safety of the system—thus hampering the goal of achieving higher degree of autonomy, especially in safety critical domains. To this end, the dissertation shall discuss formal techniques that are able to verify and design autonomous systems for safety, even under the presence of such uncertainties, allowing for their trustworthy deployment in the real world.

Specifically, the dissertation shall discuss monitoring techniques for autonomous systems from available (noisy) logs, and safety-verification techniques of autonomous system controllers under timing uncertainties. Secondly, using heterogeneous learning-based cloud computing models that can balance uncertainty in output and computation cost, the dissertation will present techniques for designing safe and performance-optimal autonomous systems.

Dedicated to মা (Mother), বাবাই (Father), দিদিভাই (Sister), ছোটোপিসিমনি (Auntie), ছোটকা (Uncle), and my family ···

চিত্ত যেথা ভয়শূন্য, উচ্চ যেথা শির,
জ্ঞান যেথা মুক্ত, ...
যেথা বাক্য হৃদয়ের উৎসমুখ হতে
উচ্ছসিয়া উঠে, যেথা নির্বারিত স্রোতে
দেশে দেশে দিশে দিশে কর্মধারা ধায়
অজস্র সহস্রবিধ চরিতার্থতায়,
যেথা তুচ্ছ আচারের মরুবালুরাশি
বিচারের স্রোতঃপথ ফেলে নাই গ্রাসি . .
.

রবীন্দ্রনাথ ঠাকুর

Where the mind is without fear and the head is held high:
Where knowledge is free: . . .
Where words come out from the depth of truth:
Where tireless striving stretches its arms towards perfection:
Where the clear stream of reason has not lost its way into the dreary desert sand of dead habit: . . .

Rabindranath Tagore

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to the following individuals who have played a significant role in the completion of this thesis:

I extend my deepest appreciation to my advisor, Sridhar. His introduction to the field of Cyber-physical systems and unwavering support throughout my PhD journey have been invaluable. Not only did he guide me in my research endeavors, but our non-academic discussions have also been enlightening and enriching. I am truly grateful for his mentorship and the overall impact he has had on my academic and personal growth.

I would like to extend a profound thank you to Samarjit for his constant support, both professionally and personally. Samarjit's guidance went beyond research, as he mentored me in developing the essential skills required to be a successful scientist. I am indebted to him for providing me with the knowledge and wisdom that I would not have acquired otherwise. Both Sridhar and Samarjit have been instrumental in shaping not only my research acumen but also my overall understanding of the craft. Their mentorship has been invaluable to me.

I am immensely grateful to Étienne for the remarkable learning experience and joyful collaboration we shared. He introduced me to the field of monitoring, and I thoroughly enjoyed working with him. Throughout our collaboration, I learned a great deal from his expertise, and I appreciate the knowledge and insights he shared with me.

I would like to express my sincere appreciation to Sandeep for introducing me to the world of robotics. His guidance and expertise have greatly influenced the direction of my research in the latter part of this thesis. I am grateful for the learning opportunities he provided and the impact he had on my academic journey.

Although I did not have the opportunity to work directly with Ron, Jim, and Sriram, I am indebted to them for the valuable insights and research discussions we had. Their contributions and expertise have been instrumental in shaping my future research agenda. I am grateful for the knowledge and perspectives they shared with me.

I would like to extend special thanks to all those who have been my support system during this journey, tolerating my occasional tantrums and providing calm and solace during challenging moments. Your

unwavering support and understanding have been vital, and I cannot thank you enough for being there for me. Your presence has helped me maintain my sanity and navigate through the ups and downs of this academic pursuit.

Lastly, but certainly not least, I would like to express my deepest gratitude to সুমন দা (Suman Da) for sparking my interest in "Computers" and being the best teacher I have ever had. Your impact on my young mind will forever be forgotten, and I am forever grateful for your guidance and inspiration.

To all the individuals mentioned above, and to everyone else who has supported me in various ways throughout this journey, thank you from the bottom of my heart. This thesis would not have been possible without your invaluable contributions, guidance, and unwavering support.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

These advancements have helped us to achieve limited autonomy in several places, such as: Advanced Driver Assistance Systems (ADAS), ensuring safety and efficiency in factory floors, assisting doctors in surgeries, wearables and home assistants, *etc.* [4, 5, 6]. Autonomous systems, however, face the challenge of operating in diverse environments characterized by various uncertainties, including modeling errors, sensor errors, timing errors, implementation uncertainties, and the dynamic nature of the surroundings. Even if these uncertainties are present in small amounts, they can have a profound impact on the safety of the system, thereby hindering the pursuit of achieving a higher degree of autonomy, particularly in safety-critical domains. As a result, in order to instill trust and confidence in the deployment of autonomous systems, particularly in safety-critical domains, it is crucial to verify their provable safety in the presence of uncertainties. This verification process ensures that the system's behavior remains within acceptable bounds, even when faced with environmental uncertainties.

An equally significant aspect of verification involves the design of autonomous systems with provable performance and safety guarantees. Let us consider an autonomous system that runs computationally intensive perception models alongside multiple other processes on a scheduler. In the design of such systems, it is essential to carefully determine the order in which computations are performed to ensure optimal performance in terms of computational cost, scheduling delay, and perception accuracy [7]. This entails organizing the computational tasks in a manner that maximizes efficiency while maintaining the desired level of accuracy. Moreover, the design process should explicitly take scheduling choices into account, as these choices can have a significant impact on the overall stability of the system [8]. By carefully considering the scheduling approach, potential instabilities and conflicts can be mitigated, enhancing the reliability and robustness of the autonomous system.

In addition to the existing design challenges, modern robotic systems face the complexity of utilizing a wide array of perception models with varying computational costs and accuracies in order to achieve a desired task while minimizing control expenses. The main challenge in such a scenario is to select and utilize the appropriate models at the right moments to accomplish the task with the lowest control cost and

in the shortest possible compute time. As a result, this thesis investigates techniques for designing and verifying autonomous systems that are inherently heterogeneous in nature, encompassing factors such as scheduling choices, computation order, perception accuracy and cost, among others. These systems operate in uncertain environments, and this thesis aims to explore methods to ensure their provable performance and safety guarantees. By addressing the complexities arising from the heterogeneity of autonomous systems and the uncertainties in their operating environments, the research aims to contribute to the development of robust and reliable autonomous systems that can effectively handle various challenges while maintaining desired performance and safety standards.

The algorithmic core of most autonomous systems, be it in automotive or robotics, is a feedback control loop. Hence, in this thesis, the primary focus will be on exploring the closed-loop control aspect of autonomous systems and developing methodologies for their design and verification. The initial part of the thesis will involve creating artifacts to investigate and understand the characteristics and modeling of uncertainties. Building upon these artifacts, this thesis proposes algorithms that can be used to design and verify autonomous systems. These algorithms will take into account the availability of heterogeneous compute resources, the challenges posed by uncertain environments, and the objective of ensuring provable safety and performance guarantees. By delving into the closed-loop control aspect and addressing the uncertainties inherent in autonomous systems, this thesis seeks to contribute valuable insights and practical solutions for designing and verifying autonomous systems that can operate effectively and reliably in real-world scenarios.

Formal verification of dynamical systems, such as autonomous vehicles, drones, spacecrafts, medical devices *etc.*, often assumes that the system model is perfectly known, *i.e.*, the model is free from uncertainties. But in reality, this is seldom the case—a dynamical system model is dependent on several parameters. For instance, consider the model of an adaptive cruise controller (ACC), the model is dependent on parameters like the mass of the car, acceleration of the lead vehicle *etc.* [9]. These parameters are measured by sensors, whose accuracy is not perfect, but measures a value with some error. Such examples are plenty in medical devices—an anesthesia delivery model is dependent on various parameters that are often measured empirically [10]. Therefore, verification assuming a perfectly known model might not be very useful in an uncertain environment. Towards this, the first part of the thesis will propose several reachable set computation methods (symbolic and numerical) that can take into account uncertainties [11]. Further, this will be helpful in the second part of the thesis, which proposes algorithms, with provable safety and performance guarantees, to verify and design autonomous systems that can handle uncertainties. This thesis also develops tools to

compute reachable sets, using the proposed methods, that can take into account uncertainties. All such code is be made open source through public repositories. This shall constitute the first of my thesis. In the second part of thesis, we leverage some of the artifacts developed in the first part, to propose algorithms to design and verify autonomous systems with provable safety and performance guarantees that can operate in uncertain environments. In particular, to verify systems for safety, we investigate quantitative safety verification where traditional qualitative safety verification (*viz.*, stability) might fail, and also monitoring based techniques as a lightweight yet useful method for verification [12].

1.1 Role of Uncertainties: Understanding Their Impact and Implications

In today's rapidly advancing technological landscape, autonomous systems have emerged as powerful tools that play significant roles across various domains. These systems, encompassing drones, vehicles, robots, medical devices, and more, possess the ability to operate without direct human intervention. However, their effectiveness and reliability are often challenged by the presence of uncertainties inherent in their operational environments. Uncertainties in autonomous systems can arise from diverse sources, including modeling errors, dynamic environments, timing uncertainties, and perception errors of neural networks, among others. The accurate modeling of drones and vehicles, for instance, necessitates precise measurements of mass and acceleration, as any errors in these parameters can lead to inaccurate predictions and potentially compromise the system's performance. Similarly, in dynamic environments where humans and robots collaborate, uncertainties can arise due to the unpredictable nature of human behavior and interactions. Timing uncertainties also pose a significant challenge for autonomous systems. These uncertainties stem from factors such as delays in communication, sensor response times, or the synchronization of different components within the system. Timely decision-making is crucial for autonomous systems, and any uncertainty in timing can result in sub-optimal actions or even system failures. Furthermore, perception errors of neural networks, which serve as the backbone of many autonomous systems, can introduce uncertainties. Neural networks rely on training data to make sense of their surroundings, and any inconsistencies or limitations in the training data can lead to perceptual errors. These errors can impact the system's ability to accurately interpret its environment and make informed decisions. Addressing uncertainties in autonomous systems is a crucial research area, as it directly impacts their safety, efficiency, and overall performance. Robustness and adaptability become key attributes that need to be incorporated into the design and operation of these

systems to mitigate the effects of uncertainties. In this context, this thesis explores the challenges posed by uncertainties in autonomous systems and investigates various strategies, techniques, and approaches employed to handle and mitigate these uncertainties. By understanding and addressing these uncertainties, we can pave the way for the continued advancement and successful deployment of autonomous systems in diverse real-world applications.

Let us consider the example of anesthesia, which is a crucial safety-critical event preceding nearly all surgical procedures. To ensure its safety, it is necessary to comprehend the drug's metabolic processes within the body and its impact on the depth of hypnosis. A prevalent method to determine anesthesia safety involves acquiring a dynamical model of the system, which captures the rate of change of the system's state variables. These state variables correspond to diverse concentration levels that provide insights into the drug's metabolism and the depth of hypnosis. It is imperative for these concentration levels to consistently remain within predetermined limits, as higher levels can prolong unconsciousness, while lower levels may result in the patient being conscious during surgery, which can be traumatic. Consequently, having a dynamical model that accurately represents the rate of change of these state variables is crucial for analysis. Intuitively, the rate of change of state variables is influenced by an interaction between their previous values and the input, namely the anesthesia drug administered to the patient. This interaction is governed by several empirically determined parameters, such as lung capacity, weight, alveolar volumes, and others. Once we possess a dynamical model of the system, it becomes possible to compute the potential concentration levels after a specified time t given a particular anesthesia drug dosage. Subsequently, we can verify whether these concentration levels indeed fall within the safe limits. These concentration levels, represented as a set, are referred to as reachable sets. Computing reachable sets for such models is relatively straightforward in the absence of modeling uncertainty. However, in the presence of uncertainties, determining reachable sets can be exceedingly challenging. Nevertheless, it is still feasible to compute over-approximations of reachable sets that account for uncertainties. In the context of the anesthesia example, the rate of change of state variables is influenced by various parameters specific to individual patients, including weight, lung capacity, alveolar volumes, among others. Since these parameters are empirically determined, their associated values may exhibit some degree of error range. Furthermore, due to their patient-specific nature, employing a fixed set of values for all patients may not accurately capture the system's dynamics. Consequently, for a safety-critical event like anesthesia, explicit consideration of uncertainties in the computation of reachable sets is necessary to ensure the soundness of the safety analysis. To address this requirement, this thesis

proposes several symbolic and numerical approaches for computing reachable sets that can account for uncertainties, as detailed in Chapter 2. While computing reachable sets that account for uncertainties remains highly challenging, this thesis demonstrates that there are specific classes of uncertainties for which exact reachable sets can be computed, even with the inclusion of uncertainties (see Section 2.2).

Reachable sets play a vital role in the verification and design of autonomous systems, providing valuable insights into the cumulative impact of uncertainty on system behavior and safety. However, while reachable sets offer an understanding of the overall system's response to uncertainty, they do not provide detailed insight into the individual parameters' impact on system safety. Yet, certain parameters may have a more significant influence on safety than others. Therefore, gaining an understanding of parameter rankings based on their impact on safety can be highly beneficial in designing cost-effective systems that maintain safety standards. The ability to rank parameters based on their impact on safety allows for strategic decision-making in system design. By identifying parameters that are highly sensitive to uncertainty, designers can allocate resources to deploy highly accurate sensors or implement robust techniques in those specific areas. Conversely, parameters that demonstrate robustness to uncertainty can be approached with a more balanced trade-off between accuracy and cost. This ranking approach provides a systematic way to prioritize design considerations, ensuring that safety is not compromised while optimizing resource utilization. To address this need, this thesis proposes a novel ranking algorithm to assess the impact of parameters on system safety, as outlined in Section 2.2.3. The algorithm offers a comprehensive and objective method for determining the relative importance of different parameters in the context of safety. By employing this ranking algorithm, the thesis also introduces techniques to evaluate the permissible amount of uncertainty that a system can tolerate without violating safety requirements. Ultimately, this thesis contributes to the development of robust and reliable autonomous systems that can operate within predefined safety limits, even in the presence of uncertainties. Furthermore, the techniques introduced in Chapter 2, on understanding roles of uncertainties, will be used in the later chapters of this thesis to verify autonomous systems for safety, and design trustworthy autonomous systems.

1.2 Verification of Autonomous Systems

The deployment of autonomous systems in safety-critical domains necessitates a high level of trust and confidence in their reliable operation. These systems, however, encounter significant challenges when

operating in diverse environments characterized by a multitude of uncertainties. Uncertainties such as modeling errors, sensor errors, timing errors, implementation uncertainties, and the dynamic nature of the surroundings can have a profound impact on the safety of autonomous systems, even when present in small amounts. This hampers the pursuit of achieving a higher degree of autonomy, particularly in safety-critical domains where the consequences of system failures can be severe. To instill trust and confidence in the deployment of autonomous systems, it becomes crucial to provably verify their safety in the presence of uncertainties. The verification process ensures that the system's behavior remains within acceptable bounds, even when faced with environmental uncertainties that may affect its performance. By subjecting autonomous systems to rigorous safety verification, it becomes possible to mitigate risks associated with uncertain factors and enhance the system's overall reliability. The presence of environmental uncertainties highlights the need for proactive measures to guarantee the safety of autonomous systems. These uncertainties, when left unaccounted for, can propagate and amplify, leading to potentially catastrophic consequences. It is imperative to develop methodologies and techniques that enable the assessment and quantification of uncertainties, allowing for the establishment of safety bounds that autonomous systems should adhere to. In safety-critical domains, where the consequences of system failures can result in significant harm or loss of life, the verification of provable safety is of utmost importance. The ability to ensure that an autonomous system operates within predefined safety limits despite the presence of uncertainties can significantly enhance the trustworthiness of such systems. It not only assures stakeholders that the deployed autonomous systems are equipped to handle the challenges posed by uncertain environments but also instills confidence in the public, regulators, and end-users. This thesis aims to address the critical need for provable safety verification of autonomous systems in the presence of environmental uncertainties, particularly in safety-critical domains. By investigating and developing robust methodologies, algorithms, and frameworks, we seek to provide a comprehensive approach to verifying the safety of autonomous systems, accounting for various sources of uncertainty. The thesis will focus on understanding the impact of uncertainties on system behavior and developing techniques that enable the system to operate safely within predefined bounds despite these uncertainties. The complexity of autonomous system modules, including neural networks, presents unique challenges in ensuring provable safety. Formal safety verification techniques, such as those employing reachable sets, have been extensively used to verify the safety properties of dynamical systems. However, when applied to autonomous systems with intricate modules like neural networks, these techniques often encounter scalability issues. The computational burden and limitations in capturing the complex behavior of

neural networks hinder their effective application in safety verification. Consequently, new approaches are required to address these challenges and enhance trust in the deployment of autonomous systems. In light of these considerations, this thesis focuses on exploring statistical approaches for verifying the safety of complex autonomous systems. By leveraging statistical techniques, it becomes possible to account for uncertainties and intricacies in the behavior of autonomous system modules, such as neural networks. These approaches offer a promising avenue for assessing the safety of autonomous systems and addressing the limitations of traditional formal verification techniques. By advancing the field of statistical safety verification for complex autonomous systems, this research contributes to addressing the challenges associated with ensuring provable safety in the presence of uncertainties. The outcomes of this work will not only provide valuable insights into the design and deployment of autonomous systems but also hope to lay the groundwork for the adoption of statistical approaches as an alternate practice in safety-critical domains. Ultimately, the aim is to foster trust and confidence in the deployment of autonomous systems while ensuring their safe operation in complex and uncertain environments.

Chapter 2 introduces a comprehensive artifact that utilizes reachable sets to perform safety verification in autonomous systems. However, the subsequent chapters of this thesis shift their focus to specific safety verification paradigms, namely the monitoring of system behavior and the handling of timing uncertainty in schedulers due to the sharing of processors by multiple controllers. These chapters, specifically Chapters 3 and 4, will present a range of safety verification methodologies that employ both formal and statistical approaches. The following are the specific paradigms that will be extensively discussed in the subsequent chapters of this thesis.

Monitoring of autonomous systems to detect crashes. Following a crash involving a plane or a vehicle, crucial questions arise from both law enforcement agencies and insurance companies: *"What caused the crash?"* and *"Who is at fault?"* To investigate these incidents, flights and vehicles are equipped with black boxes, devices that store critical information about the vehicle or plane leading up to the crash. These black boxes capture data such as controller inputs, voice recordings, and snapshots of state variables at specific time instants. However, accurately logging data at all time steps poses challenges for various reasons. The ability to log data accurately at every time step may be hindered by multiple factors. Systems designed on a tight budget often rely on uncertain or less accurate sensors for logging purposes. Additionally, logging data over an unreliable network can introduce uncertainties in the associated timestamps, leading to potential

inaccuracies. In intermittently powered devices, allocating energy for continuous logging may not always be feasible. Consequently, the collected logs can exhibit noise and contain missing samples, further complicating subsequent analysis. To address these challenges, Chapter 3 proposes an offline monitoring technique capable of analyzing noisy logs with missing samples while accounting for uncertainties in timestamps. This offline monitoring approach aims to identify safety violations by thoroughly analyzing a given log. Additionally, Chapter 3 introduces a safety-aware online logging approach that optimizes the logging process by selectively recording data only when necessary. By employing such an online monitoring approach, this thesis demonstrates the potential to enhance overall trust and energy efficiency in autonomous systems. Furthermore, this approach has been rigorously proven to be both sound and complete. By combining offline and online monitoring techniques, this thesis aims to contribute to improved crash investigations and the proactive implementation of safety measures. The outcomes of this thesis hopes to have implications for law enforcement agencies, insurance companies, and developers of autonomous systems, as they foster trust and confidence in the operation of these systems. Ultimately, the goal is to advance autonomous systems while ensuring their safe and efficient operation in real-world scenarios.

Verifying and designing autonomous system schedulers in the presence of timing uncertainties caused by sharing of processor by the controllers. Consider the concept of quantitative safety in the context of autonomous systems. Imagine a robot that is required to navigate along a predetermined path, avoiding obstacles. The robot employs multiple controllers on a single processor, including those responsible for perception, heat control, and a path follower that guides the robot along its predefined trajectory while avoiding obstacles. However, due to the simultaneous execution of multiple controllers on a shared processor, some controllers may fail to meet their deadlines, resulting in delayed computation of control inputs. In such a scenario, what happens if the path follower misses its deadline? The robot's trajectory can deviate from the planned path, potentially compromising safety by bringing it closer to obstacles. It becomes evident that not all patterns of deadline misses are equally safe. Deadline miss patterns that cause minimal deviation from the planned trajectory are considered safe, while patterns resulting in significant deviation and potential collisions are deemed unsafe. Chapter 4 of this thesis introduces techniques to prove the safety of autonomous system controllers in the presence of timing uncertainties. Specifically, given a pattern of hits and misses in meeting deadlines, we compute the maximum possible deviation that can occur for a controller. Furthermore, this chapter briefly explores available techniques that leverage the aforementioned verification methodologies

to design provably safe schedulers. These schedulers ensure that all controllers on a shared processor are scheduled in a manner that guarantees safety, with none of the controllers violating the established safety constraints. Through the proposed techniques and the incorporation of verified schedulers, this thesis aims to enhance the safety and reliability of autonomous systems by addressing timing uncertainties and preventing unsafe deviations from planned trajectories. The subsequent chapters delve into the detailed presentation and evaluation of these techniques, contributing to the overall advancement of autonomous system safety.

1.3 Designing of Autonomous Systems

In addition to verification, the design of autonomous systems with provable performance and safety guarantees is equally vital. However, this task is known to be computationally challenging, as it involves navigating through numerous design choices while ensuring desired properties. The design process can be seen as an iterative invocation of verification modules, evaluating different design options. This thesis introduces design paradigms that leverages on the unique characteristics of the systems and the uncertainties they encounter, leading to computationally scalable design algorithms. In Chapter 5, this thesis presents algorithms that facilitate the design of autonomous systems with provable safety and optimal performance. The proposed design paradigms take into account various aspects such as computational cost and control cost, aiming to achieve optimal performance while maintaining safety guarantees. The algorithms leverage the inherent properties of the systems and the nature of uncertainties in which they operate to develop efficient and scalable design solutions. The thesis addresses several key design challenges in autonomous systems, offering solutions to the following problems.

Choosing between two compute models. Consider the scenario of a robot engaged in autonomous driving, where it is required to perform a task based on its perception capabilities. To facilitate this task, the robot is equipped with a local model that is computationally efficient but possesses limited accuracy. Additionally, the robot has access to a cloud model that offers high accuracy but comes with a higher computational cost. The question that arises is whether the robot should invoke the cloud model at a given time step or not. If the robot were to invoke the cloud model at every time step, it would achieve a high cumulative accuracy (low loss) but at a significant cumulative computational cost. Conversely, relying solely on the local model at all time steps would result in a low cumulative computational cost but also a low cumulative accuracy (high loss). A smarter strategy in this context would involve invoking the cloud model selectively, particularly

during complex maneuvers, while utilizing the local model for other instances. The aim is to achieve a solution that strikes a balance between low cumulative computational cost and high cumulative accuracy. This is referred to as the model selection problem. In Section 5.1 of this thesis, we propose a strategy that can determine an optimal sequence for invoking the models, enabling the robot to achieve this desired balance in a safe and provably optimal manner. By analyzing the context and requirements of the task, the proposed strategy computes a sequence of model invocations that maximizes accuracy during crucial maneuvers while minimizing computational costs during other periods. Through this research, we aim to address the challenge of resource allocation in autonomous systems, specifically focusing on model invocations, to optimize the trade-off between computational cost and accuracy. Chapter 5 provides a detailed exploration of the proposed strategy, presenting its theoretical foundations, algorithmic implementation, and empirical evaluation. The results demonstrate the effectiveness of this strategy in achieving optimal performance while ensuring safety, contributing to the advancement of autonomous systems in real-world applications.

Decision making in perception-based robots. Fig. 5.13a depicts the wide array of perception networks currently available, offering a range of accuracy and cost trade-offs. The b7 perception model is derived from scaling up the parameters of the b0 model, resulting in a significant enhancement in accuracy. Numerous models can be obtained by smoothly adjusting the parameter count of b0 to achieve a reasonable increase in accuracy. Given this abundance of models, the question arises: Can we effectively perform model selection? It is evident that such a strategy must effectively handle the complex relationship between loss and cost, which can exhibit high degrees of non-linearity among the models. In Section 5.2, this question is answered affirmatively. Specifically, Section 5.2 introduces an optimal solution to the model selection problem, accommodating any number of perception models. Moreover, Section 5.2 demonstrates that a specific subset of this problem can be solved in polynomial time. This implies the existence of design paradigms for training perception models that can effectively balance accuracy and cost. Furthermore, the proposed approach was evaluated on a photo-realistic drone landing scenario, outperforming benchmark methods.

By tackling these design problems, the thesis aims to contribute to the advancement of autonomous systems by providing scalable design algorithms that guarantee safety and optimize performance. Chapter 5 delve into the detailed presentation, evaluation, and validation of these design solutions, offering valuable insights into the development of reliable and efficient autonomous systems. The aim is to enhance the

trustworthiness and effectiveness of autonomous systems in real-world applications, hoping to pave the way for their widespread adoption and integration into various domains.

1.4 Thesis

Autonomous systems have made significant progress in achieving limited autonomy in various domains. However, the pursuit of greater autonomy is often impeded by the presence of uncertainties. This thesis aims to address the challenge of designing and verifying autonomous systems in the face of uncertainties, with the goal of creating robust, reliable systems capable of handling unexpected situations. The research presented in this thesis can be broadly categorized into three main areas: understanding uncertainties, verification of autonomous systems, and designing autonomous systems. In the first part, which focuses on understanding uncertainties, the thesis proposes a safety verification approach for autonomous systems that incorporates reachable sets capable of accounting for uncertainties (to be discussed in Chapter 2). Moreover, not all uncertainties have the same impact on safety. To address this, an algorithm is introduced to rank uncertainties based on their safety implications. By leveraging this algorithm, autonomous systems can be designed to utilize resources efficiently while adhering to safety constraints. Additionally, the algorithm enables the computation of the level of uncertainty a system can tolerate without compromising safety. The second part of the thesis centers on the verification of autonomous systems. Here, the thesis presents techniques for monitoring autonomous systems to detect the causes of crashes. Both offline and online monitoring techniques are proposed, specifically tailored for autonomous systems with noisy logs and missing samples (to be discussed in Chapter 3). Furthermore, quantitative verification methodologies are introduced to verify the schedulers of autonomous systems in the presence of timing uncertainties (to be discussed in Chapter 4). These techniques are particularly valuable in the design of modern vehicles that require running multiple controllers on shared processors, eliminating the need for expensive high-bandwidth processors. Finally, the thesis focuses on algorithms for designing autonomous systems that are both safe and performance optimal. Specifically, it proposes a provably optimal solution to the model selection problem, initially considering two compute models and subsequently extending to N perception models (to be discussed in Chapter 5). The proposed methods have been evaluated through simulated navigation of a Mars rover on real Martian terrain, utilizing NASA's dataset to determine safety using reachable sets. Furthermore, they have been tested in a photo-realistic drone landing scenario, achieving safe landings with minimal control cost and compute

time. The proposed approaches not only provide provable safety and performance optimality but also surpass alternative methods in terms of effectiveness. Overall, this thesis offers novel insights and methodologies to address the challenges of designing and verifying autonomous systems in the presence of uncertainties. By considering uncertainties, autonomous systems can be equipped to handle unexpected events, ensuring robustness and reliability. The research contributes to the advancement of autonomous systems, opening new possibilities for their deployment in various domains.

Formally, I put forward the following thesis statement:

> *"Autonomous systems operating in uncertain environments are required to be performance optimal, without violating safety, under the presence of uncertainties. Leveraging the knowledge of environmental uncertainties, autonomous systems can be verified and designed to be provably safe with performance guarantees. Such guarantees can be obtained both formally and statistically"*

1.5 Contributions

The thesis has the following contribution.

1.5.1 Reachable set computation techniques that can account for uncertainties.

This thesis will propose reachable set computation techniques, both deterministic and probabilistic, that can account for model uncertainties of linear dynamical systems. The detailed techniques are discussed in Chapter 2. In particular, this thesis will explore the following.

1. Classes of uncertainties for which safety verification can be performed efficiently. In other words, we will study conditions that can sufficiently guarantee efficient verification. The details are discussed in Section 2.2.

2. Proposed generalized reachable set computation techniques that can handle uncertainties, both symbolic and numerical techniques. This is discussed in Section 2.3.1.

3. All uncertainties are not equal——some uncertainties have more impact on safety than others. Therefore, we will discuss an algorithm to order the uncertainties based on its impact on safety. This is discussed in Section 2.2.3.

4. Explore statistical methods to compute reachable sets, with probabilistic guarantees, to scale reachable set computation for various types of systems. This is discussed in Section 2.3.2.

1.5.2 Scalable techniques to perform safety verification of systems under uncertainties.

In this part of the thesis, we will discuss scalable verification techniques, of autonomous systems, under the presence of uncertainties. The detailed techniques are discussed in Chapters 3 and 4. In particular, this thesis has the following contributions:

1. Monitoring-based techniques to analyze safety and correctness of dynamical systems, where traditional analysis does not scale. This is discussed in Chapter 3.

2. A Python-based tool to perform monitoring of systems with a given log. The tool is available open-source. This is discussed in Section 3.4.

3. Quantitative verification (and designing) of embedded controllers under timing uncertainties, where qualitative analysis (*viz.*, stability) might fail. This is discussed in Chapter 4.

1.5.3 Designing autonomous systems that can leverage heterogeneous compute resources optimally.

In the final part of the thesis, we will discuss optimal design approaches to design autonomous systems that uses heterogeneous/distributed compute resources without violating safety and correctness. The detailed techniques are discussed in Chapter 5. In particular, we will explore the following problems:

1. Optimal solution to the model selection problem with two compute models. This is discussed in Section 5.1.

2. Optimal solution to the model selection problem with N perception models. This is discussed in Section 5.2.

CHAPTER 2: HOW ROBUST IS SAFETY?

The modeling of autonomous systems depends on various parameters that are influenced by the specific environment in which the system operates. These parameters include factors such as coefficient of friction and density of materials. Since autonomous systems are now being used in diverse environments with uncertainties, the task of modeling them becomes particularly challenging. In simpler terms, it is extremely difficult, if not impossible, to obtain precise values for all the parameters necessary to model the system accurately in such a wide range of environments. The fundamental algorithmic structure of most autonomous systems, whether they are in the automotive or robotic domain, is a dynamical system. Therefore, in this chapter, we introduce a formalism called the *"Uncertain Linear System"* which extends the capabilities of linear dynamical systems by incorporating representations of uncertainties (also referred as *linear uncertain system* or *uncertain linear dynamical system*).

In order to perform safety analysis on dynamical systems, it is necessary to have an accurate mathematical model of the system. However, due to inherent uncertainties, constructing a completely precise model can be challenging. *Hence, it is crucial to assess the robustness of the safety analysis, which means evaluating whether the safety specifications remain valid even when the model is subjected to certain perturbations.* Let us consider the example of a robot operating on a factory floor, depicted in Fig. 2.1. The robot's objective

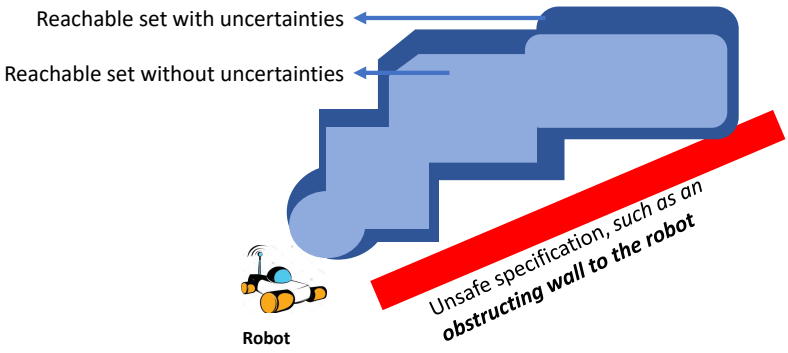


Figure 2.1: A robot, in a factory floor, trying to avoid collision with the wall (in red). The set of states reached by the robot on a clean factory floor is marked in light blue. The set of states reached by the robot when there is an oil spill on the floor is marked in dark blue (including the region in light blue).

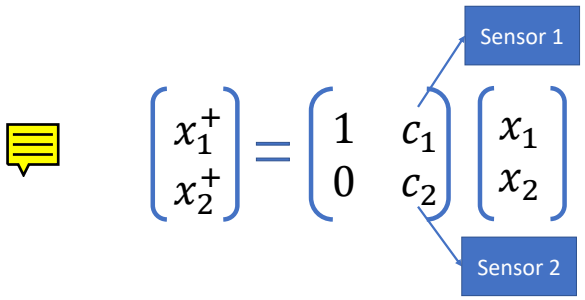


Figure 2.2: An example an uncertain linear system, where parameter c_1 is computed empirically using sensor 1, and c_2 is computed empirically using sensor 2. The main idea to prioritize one sensor over another based on the impact on safety its corresponding parameter has.

is to navigate through the floor while avoiding collisions with a red wall. The robot's model relies on a parameter called the coefficient of friction, which was assumed to have a fixed value during the modeling phase. However, due to an oil spill on the floor, the coefficient of friction has changed. During the traditional formal verification process, the robot's behavior was assessed based on the assumption that the coefficient of friction remained constant (as used in the model). However, with the parameter altered by the oil spill, the previously established safety guarantee no longer holds. In particular, the set of states that the robot can reach, assuming the fixed coefficient of friction, is represented by the light blue region in the figure. Within this region, the robot exhibits no unsafe behavior, meaning it avoids colliding with the wall. However, due to the presence of the oil spill, the actual set of states that the robot can reach is now represented by the dark blue region, which includes the previously considered light blue region. As a result, this expanded region introduces the potential for unsafe behavior, namely colliding with the wall. Throughout the remainder of the thesis, we refer to these modeling parameters as uncertainties since assigning a fixed value to them proves challenging.

A (continuous time) linear dynamical system without uncertainties is defined as $\dot{x} = Ax$, where the dynamics matrix A is assumed to be known accurately, *i.e.*, all the elements of the matrix are known accurately. Note the following Example 1 of a linear system.

Example 1. *Consider the continuous time linear dynamical system* $\dot{x} = Ax$, *where* A *is* $\begin{bmatrix} 1 & c \\ 0 & 2 \end{bmatrix}$ *and* c *represents a system parameter.*

Traditional methods of reachable set computation of linear systems assume a fixed value of c (say, -1) [13, 14, 15, 16]. However, in reality, the exact value of c is unknown and can only be estimated. Therefore, it is more appropriate to consider c as a parameter that can assume a range of values, denoted as $c \in [-1.1, -0.9]$

for instance, taking into account the uncertainties associated with sensing and actuation. It is important to note that traditional safety proofs based on reachable set techniques often fail to consider such uncertainties. *Further, the reachable set techniques do not inform us about the amount of uncertainties that can be tolerated while maintaining safety.* In this thesis, we consider a model with uncertainties—we define continuous time uncertain linear systems as $\dot{x} = \Lambda_u x$ (corresponding discrete time uncertain system will also be introduced later in this chapter), where the uncertainties in the model are considered as variables in Λ_u that can take values from a bounded range.

As previously mentioned, the reachable set computation techniques do not offer insights into the level of uncertainty that a system can tolerate, nor do they provide an ordering of uncertainties based on their impact on safety. These types of information become valuable when modeling systems with budget constraints. To illustrate this point, consider the uncertain linear system depicted in Fig. 2.2. The parameter c_1 is empirically measured using sensor 1, while c_2 is measured using sensor 2. When modeling a system with budget constraints, it becomes essential to understand which parameter, either c_1 or c_2, has a more significant impact on safety. If c_2 is determined to have a more substantial effect on safety, it would be logical to allocate more energy and resources towards improving sensor 2, and vice versa. Once the uncertainties are appropriately ordered, this information can be leveraged to determine the range of values that these parameters (c_1 and c_2) can assume without violating safety. In other words, it becomes possible to compute the range of values within which these parameters can vary while still maintaining safety. This calculation constitutes the robustness threshold. As a second major contribution, we introduce an ordering algorithm that ranks the parameters (cells of the dynamics matrix) based on their impact on safety. By utilizing this ordering, we propose heuristics that can compute the robustness threshold, which establishes the range of values that these parameters can tolerate without compromising safety. We also discuss the special subclass of uncertainties for which the verification can be performed very efficitenly and precisely, *i.e.*, this chapter will propose a *sound and complete* verification technique for a special class of uncertainties.

About the chapter. The contents of this chapter is taken from [17, 18, 19]

2.1 Preliminaries

Dynamical systems evolve in a state space. In this chapter, our domain of state space is $\mathbb{R}^n$. The state of the system is denoted as x and vectors in $\mathbb{R}^n$ are denoted by v. Given a matrix $M \in \mathbb{R}^{m\times n}$, the $(i,j)^{th}$ element is

denoted as $M[i,j]$. The domain of Boolean matrices of dimension $m \times n$ are denoted with $\mathbb{B}^{m\times n}$ Given $\delta > 0$, $B_\delta(x) = \{y \mid ||y-x|| \leq \delta\}$. For a set $S \subset \mathbb{R}^n$, $B_\delta(S) = \cup_{x\in S} B_\delta(x)$. This operation over S is called *bloating* the set S by δ. Given two sets S_1, S_2, its Minkowski sum is $S_1 \oplus S_2 = \{x_1 + x_2 \mid x_1 \in S_1, x_2 \in S_2\}$. Given $i, j \in \mathbb{Z}$, with $i \leq j$, $[\![i,j]\!]$ denotes the set $\{i, i+1, \ldots, j\}$. A hyper-sphere with radius $k \in \mathbb{R}_{\geq 0}$ is denoted as $\mathcal{S}(k) \subset \mathbb{R}^n$; thus, a unit sphere is denoted as $\mathcal{S}(1)$. We denote the usual notion of absolute value of $a \in \mathbb{R}$ as $|a| \in \mathbb{R}_{\geq 0}$. We overload the operator $|.|$ for matrices as well: given a matrix $M \in \mathbb{R}^{n\times m}$, $|M|$ is a matrix such that $\forall_{i\in[\![1,n]\!], j\in[\![1,m]\!]} |M|[i,j] = |M[i,j]|$. Given a matrix $M \in \mathbb{R}^{n\times n}$, the maximum singular value (Max SV) of M is denoted as $\sigma(M)$, the determinant of M is denoted as $det(M)$, and $e^M = I + \frac{M}{1!} + \frac{M^2}{2!} + \ldots$.

Definition 1 (Discrete Linear Dynamical Systems)**.** *Given a matrix $A \in \mathbb{R}^{n\times n}$, a discrete linear dynamical system is denoted as:*

$$x^+ = Ax \tag{2.1}$$

Definition 2 (Trajectories)**.** *A trajectory of the discrete linear dynamical system, denoted as $\xi_A : \mathbb{R}^n \times N \to \mathbb{R}^n$, describes the evolution of the system in time. Given an initial state $x_0 \in \mathbb{R}^n$, the trajectory is defined as*

$$\xi_A(x_0, 0) = x_0. \tag{2.2}$$

$$\xi_A(x_0, t) = A \times \xi(x_0, t-1). \tag{2.3}$$

Therefore, $\xi_A(x_0,t) = A^t x_0$ where $A^t = \underbrace{A \times A \times \ldots \times A}_{t\text{-times}}$.

We drop A from the subscript of ξ when it is clear from the context.

Definition 3 (Reachable Set)**.** *Given a linear dynamical system $x^+ = Ax$, initial set of states Θ, and a time step $t \in N$, the reachable set of states*

$$RS(A, \Theta, t) = \{\, x \mid \exists x_0 \in \Theta, x = \xi_A(x_0, t)\}\tag{2.4}$$

.

Definition 4 (Safety Specification)**.** *An uncertain linear dynamical system $x^+ = \Lambda x$ with initial set $\Theta \in \mathbb{R}^n$ and time bound $T_b \in N$ is said to be safe with respect to an unsafe set $U \subseteq \mathbb{R}^n$ if and only if*

$$\forall t \in N,\ 0 \leq t \leq T_b,\ RS(\Lambda, \Theta, t) \cap U = \emptyset. \tag{2.5}$$

Definition 5. *A generalized star S is defined as a tuple* $\langle c, V, P\rangle$ *where* $c \in \mathbb{R}^n$ *is called the* center, $V = \{v_1, v_2, \ldots, v_m\}$ *where* $\forall i, 1 \leq i \leq m, v_i \in \mathbb{R}^n$ *are called a set of* basis vectors *(that span* $\mathbb{R}^n$*), and* $P : \mathbb{R}^m \to \{\top, \bot\}$ *is called the* predicate. *The set of states represented by a generalized star is defined as:*

$$\begin{aligned}[\![S]\!] = \{\, x \mid \exists \alpha_1, \alpha_2, \ldots, \alpha_m \text{ such that} \\ x = c + \Sigma_{i=1}^{m} \alpha_i v_i \text{ and } P(\alpha_1, \alpha_2, \ldots, \alpha_m) = \top\}\end{aligned} \tag{2.6}$$

We abuse notation and use S to refer to both $[\![S]\!]$ and S.

Lemma 1 (Reachable Set Computation Using Generalized Stars)**.** *Given an initial set* Θ *in a generalized star representation as* $\langle c, V, P\rangle$*, the reachable set,* $RS(A, \Theta, t) = \langle c', V', P\rangle$ *where* $c' = A^t c$ *and* $V' = \{v'_1, v'_2, \ldots, v'_m\}$ *and* $\forall i, 1 \leq i \leq m$*,* $v'_i = A^t v_i$*.*

Definition 6 (Linear Matrix Expressions)**.** *Let,* $Vars = \{y_1, y_2, \cdots, y_k\}$ *denote a set of variables and* $\mathsf{LE} = \{\sum_{i=1}^{k} \beta_i y_i, \beta_i \in \mathbb{R}\}$ *denote the set of all linear expressions over variables in Vars. A matrix M is called a* Linear Matrix Expression *(LME) if* $M \in \{\mathbb{R} \cup \mathsf{LE}\}^{n \times n}$*. An LME M over* $Vars = \{y_1, y_2, \ldots, y_k\}$ *can also be written as* $M = N_0 + N_1 y_1 + \ldots + N_k y_k$ *where* $\forall i, 0 \leq i \leq k, N_i \in \mathbb{R}^{n \times n}$ *and* N_i *represents the coefficients of the variable* y_i*. We represent the LME M as the tuple* $\langle N_0, N_1, \ldots, N_k\rangle$*. If the perturbations are time varying we represent the variables as* $Vars_T = \{y_1(t), y_2(t), \cdots, y_k(t)\}$*.* $y_i(t+1)$ *denotes the valuation of (perturbation) variable* y_i *at time* $t+1$*.*

Informally, the elements of the matrix M can be either real numbers or linear expressions of variables in *Vars*.

Definition 7 (Uncertain Linear Systems)**.** *An uncertain linear system* Λ *is defined as a pair* (M, Γ) *where M is a linear matrix expression over Vars and* $\Gamma : \mathbb{R}^k \to \{\top, \bot\}$ *is a predicate. The set of matrices denoted by* Λ *are given as*

$$\begin{aligned}\Lambda = \{A \mid \exists \gamma_1, \gamma_2, \ldots, \gamma_k, \text{ such that } \Gamma(\gamma_1, \gamma_2, \ldots, \gamma_k) = \top \\ \text{and } A = \underbrace{N_0 + N_1\gamma_1 + \ldots + N_k\gamma_k}_{M}\}\end{aligned} \tag{2.7}$$

Informally, Λ represents the set of matrices obtained by assigning different valuations to the variables *Vars* that satisfy the predicate Γ. While in general, Γ can be any semi-algebraic set, in this paper, we restrict Γ to represent polytopes. In this chapter we restrict Γ to represent a bounded polytope.

Definition 8. ***Continuous Time Linear System***. *Given a matrix $A \in \mathbb{R}^{n\times n}$, a continuous time linear dynamical system is denoted as $\dot{x} = Ax$. The trajectory of linear dynamical systems that gives the state of the system at time t starting from initial state x_0 is given as $\xi(x_0,t) = e^{At}x_0$.*

Definition 9. ***Interval Matrix*** *(as defined in Section 2 of [20]). Given two matrices $A_{min}, A_{max} \in \mathbb{R}^{n\times n}$, s.t. $\forall i,j\ A_{min}[i,j] \leq A_{max}[i,j]$: we define an interval matrix $\Lambda \subset \mathbb{R}^{n\times n}$, also denoted as $[A_{min}, A_{max}]$, where the $[i,j]^{th}$ element is the interval $[A_{min}[i,j], A_{max}[i,j]]$. The* center *of Λ (also denoted as C_Λ) is defined as $\frac{1}{2}(A_{min}+A_{max})$. Similarly, $\Delta_\Lambda = \frac{1}{2}(A_{max}-A_{min})$. A matrix $E \in \Lambda$ if and only if $\forall i,j, E[i,j] \in \Lambda[i,j]$.*

Given two interval matrices $\Lambda_1, \Lambda_2 \subset \mathbb{R}^{n\times n}$, operations like addition, subtraction and multiplication are performed according to usual matrix algebra, where each operation is performed according to interval arithmetic. The notations are usual $\Lambda_1 \star \Lambda_2$, where $\star = \{+, \times, -\}$. Given an interval matrix $\Lambda \subset \mathbb{R}^{n\times n}$, the singular value decomposition (SVD) of the interval matrix Λ is given as $\Lambda = \bigcup A = \bigcup U\Sigma V^*$, where U, V are unitary matrices and Σ is a diagonal matrix; we assume the decomposition is such that, the diagonal matrix Σ is sorted in descending order.

Definition 10. ***Matrix Norms***. *Given a matrix $M \in \mathbb{R}^{n\times n}$, its matrix 2 norm is denoted as $|| M ||_2$, and $|| M ||_F$ is the matrix Frobenius norm. We also use $|| M ||$ to denote $|| M ||_p$, where p can be anything in $\{2, F\}$. For any matrix M, $|| M ||_2 \leq || M ||_F$.*

Definition 11. ***Interval Matrix Norm*** *(as defined in Theorem 1 of [20]). Given interval matrix Λ, $|| \Lambda ||_p = sup\{|| A ||_p \mid A \in \Lambda\}$, $p \in \{2, F\}$. We also use $|| \Lambda ||$ to denote $|| \Lambda ||_p$, where p is obvious from the context.*

Definition 12. ***Uncertain Linear System*** *[11]. A uncertain linear system is denoted as $\dot{x} = (A+\Lambda)x$, where $A \in \mathbb{R}^{n\times n}$ is a matrix and $\Lambda \subset \mathbb{R}^{n\times n}$ is an interval matrix. The uncertainties considered in the dynamics, represented by the matrix $\Lambda \subset \mathbb{R}^{n\times n}$, are time time-invariant, unlike the formulation considered in [21] or more general formulation as in [22]. The set of states reached by a uncertain linear system after time t is $\xi_{A+\Lambda}(x_0,t) = \{e^{(A+M)t}x_0 \mid M \in \Lambda\}$.*

Definition 13. ***Max SV Candidate Matrix***. *Given an interval matrix $\Lambda \subset \mathbb{R}^{n\times n}$, the Max SV Candidate A_{max} is defined as $\Xi(\Lambda) = A_{max}$, where $\forall_{A\in\Lambda}\ \sigma(A) \leq \sigma(A_{max})$. Note that Max SV Candidate need not be unique. Let $\sigma(A_{max}) = \sigma_{max}$.*

Definition 14. ***Interval Matrix Transformation***. *Given a transformation $f(\theta) = \Lambda\theta$, where $\theta \subset \mathbb{R}^n$ and $\Lambda \subset \mathbb{R}^{n\times n}$, the transformation $f(\theta)$ is defined as $f(\theta) = \bigcup_{A\in\Lambda} A\theta$.*

2.2 All Uncertainties Are Not Equal: Classifying Uncertainties Based on Their Impact on Safety

This section addresses the following question: Are there a class of uncertainties, for which, computing the reachable set while accounting for uncertainties is computationally inexpensive? If so, how to characterize such uncertainties? The answer to the above questions is an affirmative. We present a class of uncertainties for which reachable set can be computed efficiently and safety verification can be performed using bi-linear optimization. We present sufficient conditions for which our algorithm is applicable.

This algorithm relies on the observation that matrix multiplication is the key operation for computing reachable set using generalized stars (Lemma 1). In this section, we identify a sub-class of linear systems with uncertainties for which matrix multiplication can be performed using symbolic techniques. We illustrate this in Examples 2 and 3.

Example 2. *Consider the discrete linear dynamical system $x^+ = Ax$ where A is $\begin{bmatrix} 1 & \alpha \\ 0 & 2 \end{bmatrix}$ and α represents either the modeling uncertainty or a parameter. Observe that $A^2 = \begin{bmatrix} 1 & 3\alpha \\ 0 & 4 \end{bmatrix}$ and does not contain any quadratic terms or higher order terms of α. Additionally, for all k, A^k does not contain any quadratic or higher order terms of α.*

Example 3. *Consider the linear dynamical system $x^+ = Ax$ where A is an $n\times n$ matrix given as $A = \begin{bmatrix} Q & R \\ \mathbf{0} & T \end{bmatrix}$ where Q ($\in \mathbb{R}^{m\times m}$) is a block matrix of size $m\times m$ (where $m < n$), R is a block matrix of size $m\times(n-m)$, $\mathbf{0}$, is a block matrix of size $(m-n)\times m$ containing only 0s, and T $\left(\in \mathbb{R}^{(n-m)\times(n-m)}\right)$ is a block matrix of size $(n-m)\times(n-m)$. Observe that A^2 is $\begin{bmatrix} Q\times Q & Q\times R + R\times T \\ \mathbf{0} & T\times T \end{bmatrix}$. If all the uncertainties are in block matrix R, then, similar to Example 2, A^2 does not have any nonlinear terms in $Q\times R + R\times T$. Additionally, this property holds for all A^k, $k \geq 2$. Example 2 is a special case of this structure.*

The properties of matrices given in Example 2 and 3 are structural in nature. Consider Example 3; irrespective of the values of elements in matrices Q ($\in \mathbb{R}^{m\times m}$) and T $\left(\in \mathbb{R}^{(n-m)\times(n-m)}\right)$, the matrix A^k will not have any nonlinear terms. We demonstrate that, reachable set for such systems can be represented using bi-linear constraints. Hence, checking for safety specification can be performed using bi-linear programming. In this section, we introduce a framework for checking if the uncertainties satisfy such structural properties. We provide sufficient conditions under which such structural properties hold and present an algorithm for safety verification using bi-linear optimization. Using these sufficient conditions, we construct an artifact called *robust reachable set* that accounts for uncertainties. Given a dynamical system, we identify all the uncertainties for which the sufficient condition is applicable and introduce uncertainties as suggested by the user. The resulting reachable set also accounts for the effect of uncertainties over the reachable set. As a result of the sufficient conditions, the robust reachable set can be computed symbolically and is not an over-approximation. Experimental evaluation on several benchmarks demonstrates that this approach is indeed computationally feasible.

Before we propose the main algorithm of this section, wish to layout some of the preliminaries.

As observed before, matrix-matrix multiplications is an important operation for computing reachable set using generalized stars. To compute reachable set of uncertain linear systems represented as LMEs, one need to compute products of two LMEs.

Definition 15. *Given two LMEs $A = \langle N_0, N_1, \ldots, N_k\rangle$ and $B = \langle M_0, M_1, \ldots, M_k\rangle$ over the same set of variables $Vars = \{y_1, y_2,$*

$\cdots, y_k\}$, the product of two LMEs (similar to product of two linear expressions) is defined as

$$
\begin{aligned}
A \times B = (N_0 + N_1 y_1 + \ldots + N_k y_k) \times \qquad & (2.8)\\
(M_0 + M_1 y_1 + \ldots + M_k y_k)\\
= N_0 M_0 + N_0 M_1 y_1 + \ldots + N_0 M_k y_k\\
+ N_1 M_0 y_1 + N_1 M_1 y_1^2 + \ldots + N_1 M_k y_1 y_k\\
+ \ \vdots\\
+ N_k M_0 y_k + N_k M_1 y_1 y_k + \ldots + N_k M_k y_k^2. \qquad & (2.9)
\end{aligned}
$$

Observe that product of two LMEs can have higher degree terms. However, in this paper, we focus our attention on LMEs for which the product is closed *i.e.* does not have any non-linear terms.

Definition 16. *LMEs $A = \langle N_0, \ldots, N_k \rangle$ and $B = \langle M_0, \ldots, M_k \rangle$ are said to be closed under multiplication if the following two conditions are satisfied:*

1. $\forall_{1 \leq i \leq k}$, $N_i \times M_i = 0$
2. $\forall_{1 \leq i,j \leq k}$ $(N_i \times M_j) + (N_j \times M_i) = 0$

Therefore, $A \times B = \langle L_0, \ldots, L_k \rangle$ where,

1. $L_0 = N_0 \times M_0$.
2. $\forall_i$ $1 \leq i \leq k$, $L_i = (N_i \times M_0) + (N_0 \times M_i)$.

If the uncertain linear system is represented as an LME, such that A^{t-1} and A are both LME and closed under multiplication, then the representation of the reachable set at time t using generalized stars will have only second degree terms.

Definition 17 (Matrix Support)**.** *Given a matrix $M \in \mathbb{R}^{m \times n}$, $\mathsf{supp}(M) = B$ where $B \in B^{m \times n}$ such that for all $i, 1 \leq i \leq m, 1 \leq j \leq n$, $B[i,j] = 0$ if and only if $M[i,j] = 0$.*

Informally, the support of a matrix identifies the elements of a matrix that are zero and the elements that are non-zero.

Definition 18 (Block Boolean Matrix)**.** *A matrix $B \in B^{n \times n}$ is said to be a block Boolean matrix, denoted as $block((r_1,c_1),(r_2,c_2))$ if and only if for all i, j where $r_1 \leq i \leq r_2$ and $c_1 \leq j \leq c_2$, $B[i,j] = 1$ and $B[i,j] = 0$ otherwise.*

Fig. 2.3 shows a pictorial representation of a Block Boolean Matrix.

Definition 19 (Addition and Multiplication of Boolean Matrices)**.** *Given Boolean matrices $B_1, B_2 \in B^{m \times n}$, we define the addition operation on Boolean matrices as $B_1 \oplus B_2 = B_3$ where $B_3 \in B^{m \times n}$ and*

$$\forall i, j, 1 \leq i \leq m, 1 \leq j \leq n, B_3[i,j] = B_1[i,j] \vee B_2[i,j].$$

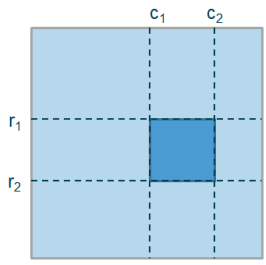


Figure 2.3: Pictorial representation of a Block Boolean Matrix $block((r_1,c_1),(r_2,c_2))$. Light Blue represents 0, and Dark Blue represents 1

Given Boolean matrices $B_1 \in B^{m\times k}$ and $B_2 \in B^{k\times n}$, we define the multiplication operation on Boolean matrices as $B_1 \otimes B_2 = B_3$ where $B_3 \in B^{m\times n}$ and

$$\forall i,j, 1 \leq i \leq m, 1 \leq j \leq n, B_3[i,j] = \bigvee_{l=1}^{k} B_1[i,l] \wedge B_2[l,j].$$

Informally, the addition and multiplication operations on
Boolean matrices are similar to that of matrices with real numbers. We also extend the matrix difference operation as follows.

Definition 20. *Given Boolean matrices B and B_1, $[\![B_1]\!]B$ is the difference matrix where $[\![B_1]\!]B[i,j] = 1$ if and only if $B[i,j] = 1$ and $B_1[i,j] = 0$.*

Informally, in $[\![B_1]\!]B$, we remove all the 1s from B that are also 1s in B_1. Given a set of matrices $B_1, B_2, \ldots, B_k$, $[\![B_1, B_2, \ldots, B_k]\!]B = [\![B_1 \oplus \ldots \oplus B_k]\!]B$.

Definition 21 (Sub-Support and Super-Support)**.** *Given Boolean matrices $B_1, B_2 \in B^{m\times n}$, we say that B_1 is a* sub-support *of B_2, denoted as $B_1 \leq B_2$ if and only if for all i,j, if $B_1[i,j] = 1$ then $B_2[i,j] = 1$. An equivalent formulation is for all i,j, if $B_2[i,j] = 0$ then $B_1[i,j] = 0$. We also say that B_2 is a* super-support *of B_1.*

Definition 22 (Intersection of Support Matrices)**.** *Given Boolean matrices $B_1, B_2 \in B^{m\times n}$, we denote intersection $B_1 \cap B_2 = B$, where $B \in B^{m\times n}$ and $B[i,j] = B_1[i,j] \wedge B_2[i,j]$*

Lemma 2. *Given $M_1, M_2 \in \mathbb{R}^{m\times n}$, $\mathsf{supp}(M_1 + M_2) \leq \mathsf{supp}(M_1) \oplus \mathsf{supp}(M_2)$.*

Proof. Let us denote $\mathsf{supp}(M_1 + M_2)$ as B_1 and $\mathsf{supp}(M_1) \oplus \mathsf{supp}(M_2)$ as B_2. Consider an i,j such that $B_2[i,j] = 0$. From Definition 19, it follows that $\mathsf{supp}(M_1)[i,j] = 0$ and $\mathsf{supp}(M_2)[i,j] = 0$. Therefore,

$M_1[i,j] = 0$ and $M_2[i,j] = 0$. Therefore $(M_1 + M_2)[i,j] = 0$. Therefore $\mathsf{supp}(M_1 + M_2)[i,j] = 0$. Therefore, for all i, j if $B_2[i,j] = 0$ then $B_1[i,j] = 0$. Therefore $B_1 \leq B_2$. □

Lemma 3. *Given $M_1 \in \mathbb{R}^{m \times n}$ and $M_2 \in \mathbb{R}^{n \times l}$, $\mathsf{supp}(M_1 \times M_2) \leq \mathsf{supp}(M_1) \otimes \mathsf{supp}(M_2)$.*

Proof. Let us denote $\mathsf{supp}(M_1 \times M_2)$ as B_1 and $\mathsf{supp}(M_1) \otimes \mathsf{supp}(M_2)$ as B_2. Consider an i, j such that $B_2[i,j] = 0$. From Definition 19, it follows that for all k, either $\mathsf{supp}(M_1)[i,k] = 0$ or $\mathsf{supp}(M_2)[k,j] = 0$ or both. Hence, for all k, either $M_1[i,k] = 0$ or $M_2[k,j] = 0$ or both. Therefore $(M_1 \times M_2)[i,j] = 0$. Therefore $\mathsf{supp}(M_1 \times M_2)[i,j] = 0$, that is, $B_1[i,j] = 0$. Therefore $B_1 \leq B_2$. □

Corollary 1. *Given matrices M_1 and M_2, if $\mathsf{supp}(M_1) \otimes \mathsf{supp}(M_2) = \boldsymbol{0}$, then $M_1 \times M_2 = \boldsymbol{0}$*

2.2.1 Sufficient Conditions for Representing Reachable Set of Uncertain Linear Systems using Bi-linear Inequalities

In this section, we present sufficient conditions for product of LMEs to be closed under product (Definition 16). This allows us to compute the symbolic representation of the reachable set of uncertain linear systems using bi-linear inequalities.

Lemma 4. *Given LMEs $A = \langle N_0, N_1, \ldots, N_k \rangle$ and $B = \langle M_0, M_1, \ldots, M_k \rangle$ over the same set of variables Vars. If $\forall i, j$, $1 \leq i \leq k$ and $1 \leq j \leq k$, $\mathsf{supp}(N_i) \otimes \mathsf{supp}(M_j) = \boldsymbol{0}$, then $A \times B$ results in an LME.*

Proof. From Corollary 1, if for all i, j, $\mathsf{supp}(N_i) \otimes \mathsf{supp}(M_j) = \mathbf{0}$, then for all i, $N_i \times M_i = \mathbf{0}$ and for all i, j, with $i \neq j$, $N_i \times M_j = \mathbf{0}$ and $N_j \times M_i = \mathbf{0}$. Therefore $N_i \times M_j + N_j \times M_i = 0$.

Therefore, both the conditions specified in Definition 16 for closure of products over LMEs are satisfied. Hence $A \times B$ results in an LME. □

Lemma 4 provides a sufficient condition for product of two LMEs to result in an LME. We now present sufficient conditions for A^k to be an LME.

Theorem 1. *Given an LME $A = \langle N_0, N_1, \ldots, N_k \rangle$ if*

$$\forall i, j, 1 \leq i, j \leq k, \mathsf{supp}(N_i) \otimes \mathsf{supp}(N_j) = \boldsymbol{0} \tag{2.10}$$

$$\forall i, 0 \leq i \leq k, \mathsf{supp}(N_0) \otimes \mathsf{supp}(N_i) \leq \mathsf{supp}(N_i),$$

$$\textit{and } \mathsf{supp}(N_i) \otimes \mathsf{supp}(N_0) \leq \mathsf{supp}(N_i). \tag{2.11}$$

then for all $m \geq 2$, A^m is an LME.

Proof. The proof is by induction. We strengthen the inductive hypothesis to prove the above property.

Inductive Hypothesis: Given an LME A that satisfies Equations (2.10) and (2.11), for all $m \geq 2$, A^m is an LME $\langle L_0^m, L_1^m, \ldots, L_k^m \rangle$, and additionally, for all $i, 0 \leq i \leq k$, $\mathsf{supp}(L_i^m) \leq \mathsf{supp}(N_i)$.

Base Case (m = 2): Given Equations 2.10 and 2.11, using Lemma 4, we know that A^2 is an LME. Let us denote $A^2 = \langle L_0^2, L_1^2, \ldots, L_k^2 \rangle$.

Consider L_i^2 where $i > 0$. From Definition 16, we know that $L_i^2 = N_0 \times N_i + N_i \times N_0$. Therefore

$$\begin{aligned}
\mathsf{supp}(L_i^2) &= \mathsf{supp}(N_0 \times N_i + N_i \times N_0) \\
&\leq \mathsf{supp}(N_0 \times N_i) \oplus \mathsf{supp}(N_i \times N_0) \\
&\qquad \text{(from Lemma 2)} \\
&\leq \big(\mathsf{supp}(N_0) \otimes \mathsf{supp}(N_i)\big) \oplus \big(\mathsf{supp}(N_i) \otimes \mathsf{supp}(N_0)\big) \\
&\qquad \text{(from Lemma 3)} \\
&\leq \mathsf{supp}(N_i) \oplus \mathsf{supp}(N_i) \text{ (from Condition 2.11)} \\
&\leq \mathsf{supp}(N_i)
\end{aligned}$$

Hence, $\forall i > 0\ \mathsf{supp}(L_i^2) \leq \mathsf{supp}(N_i)$.

The reasoning for $\mathsf{supp}(L_0^2) \leq \mathsf{supp}(N_0)$ follows similarly.

Induction Step: Suppose that for an $m > 2$ the induction hypothesis is satisfied. We now prove that the inductive hypothesis holds for $m+1$.

Let us denote A^m as $\langle L_0^m, L_1^m, \ldots, L_k^m \rangle$. From inductive hypothesis, we know that, for all i, $0 \leq i \leq k$, $\mathsf{supp}(L_i^m) \leq \mathsf{supp}(N_i)$. Therefore, from Equation (2.10) and Corollary 1, we know that $\forall i, j\ i > 0, j > 0$, $N_j \times L_i^m = \mathbf{0}$. Therefore, from Definition 16, $A \times A^m$ results in an LME. Let us denote $A^{m+1} = \langle L_0^{m+1}, L_1^{m+1}, \ldots, L_k^{m+1} \rangle$.

To prove the inductive hypothesis, consider L_i^{m+1} where $i > 0$. From Definition 16, we know that $L_i^{m+1} = N_0 \times L_i^m + N_i \times L_0^m$. Therefore

$$\begin{aligned}
\mathsf{supp}(L_1^{m+1}) &= \mathsf{supp}(N_0 \times L_i^m + N_i \times L_0^m) \\
&\leq \mathsf{supp}(N_0 \times L_i^m) \oplus \mathsf{supp}(N_i \times L_0^m) \\
&\qquad \text{(from Lemma 2)}
\end{aligned}$$

$$
\begin{aligned}
&\leq \big(\mathsf{supp}(N_0) \otimes \mathsf{supp}(L_i^m)\big) \oplus \\
&\qquad \big(\mathsf{supp}(N_i) \otimes \mathsf{supp}(L_0^m)\big) \text{ (from Lemma 3)} \\
&\leq \big(\mathsf{supp}(N_0) \otimes \mathsf{supp}(N_i)\big) \oplus \\
&\qquad \big(\mathsf{supp}(N_i) \otimes \mathsf{supp}(N_0)\big) \text{ (from Induction)} \\
&\leq \mathsf{supp}(N_i) \oplus \mathsf{supp}(N_i) \text{ from Condition 2.11.} \\
&\leq \mathsf{supp}(N_i)
\end{aligned}
$$

Hence, $\forall i > 0\ \mathsf{supp}(L_i^{m+1}) \leq \mathsf{supp}(N_i)$. The reasoning for $\mathsf{supp}(L_0^{m+1}) \leq \mathsf{supp}(N_0)$ follows similarly. □

Theorem 2. *Let $\Lambda = (A, \Gamma)$ be an uncertain system where $A = \langle N_0, N_1, \ldots, N_k \rangle$ is an LME satisfying Equations (2.10) and (2.11), and $\Gamma : \mathbb{R}^k \to \{\top, \bot\}$. Since A satisfies Equations (2.10) and (2.11), from Theorem 1, it follows that, given t, A^t is also an LME, represented as $\langle L_0, L_1, \ldots, L_k \rangle$. Given an initial set $\Theta = \langle c, V, P \rangle$; the reachable set is given as:*

$$
\begin{aligned}
RS(\Lambda, \Theta, t) \quad = \quad & \{ x \,|\, \exists y_1, \ldots, \exists y_k, \exists \alpha_1, \ldots, \exists \alpha_m, && (2.12) \\
& \quad x = c' + \Sigma_{i=1}^m \alpha_i v_i' && (2.13) \\
& \quad c' = (L_0 + L_1 y_1 + \ldots L_k y_k) \times c, \\
& \quad \forall i, 1 \leq i \leq m, v_i' = (L_0 + L_1 y_1 + \ldots L_k y_k) v_i, \\
& \quad \textit{and } \Gamma(y_1, \ldots, y_k) = \top \wedge P(\alpha_1, \ldots, \alpha_m) = \top \}
\end{aligned}
$$

Proof. Consider an $M \in \Lambda$, therefore, $M = N_0 + N_1\gamma_1 + \ldots + N_k\gamma_k$ where $\Gamma(\gamma_1, \ldots, \gamma_k) = \top$. Since A^t is also an LME, we have $M^t = L_0 + L_1\gamma_1 + \ldots + L_k\gamma_k$. Hence, $RS(M, \Theta, t)$ is given as

$$
\begin{aligned}
RS(M, \Theta, t) \quad = \quad & \{ x \,|\, \exists \alpha_1, \ldots, \exists \alpha_m, x = c' + \Sigma_{i=1}^m \alpha_i v_i' && (2.14) \\
& \quad c' = (L_0 + L_1\gamma_1 + \ldots + L_k\gamma_k) \times c, \\
& \quad \forall 1 \leq i \leq m, v_i' = (L_0 + L_1\gamma_1 + \ldots + L_k\gamma_k) v_i, \\
& \quad \text{and } P(\alpha_1, \ldots, \alpha_m) = \top \}
\end{aligned}
$$

Since $RS(\Lambda, \Theta, k) = \bigcup_{M \in \Lambda} RS(M, \Theta, t)$. Hence, adding existential quantifiers for Equation (2.14) will yield Equation (2.13). □

Observe that $RS(\Lambda,\Theta,t)$ can be formulated as a semi-algebraic set with bi-linear constraints. The safety verification algorithm for uncertain linear systems first checks if Conditions 2.10 and 2.11 are satisfied by the uncertain linear system. If the conditions are satisfied, then A^k is computed as an LME. For checking $RS(\Lambda,\Theta,t)\cap U$ where the unsafe set U is given as a polyhedral or quadratic constraints, one can employ quadratically constrained quadratic programming tools (such as Gurobi).

Algorithm 1: Algorithm for performing safety verification of uncertain linear system when A satisfies Conditions 2.10 and 2.11.

input : Uncertain linear system $\Lambda = (A,\Gamma)$ where $A = \langle N_0, N_1, \ldots, N_k\rangle$; Initial set $\Theta = \langle c, V, P\rangle$; Time step t; Unsafe Set U

output : Safe or Unsafe or Invalid

1 **if** *Condition 2.10 or 2.11 is not satisfied by A* **then**
2 **return** Invalid;
3 Compute A^t as $\langle L_0, L_1, \ldots, L_k\rangle$;
4 Compute $RS(\Lambda,\Theta,t)$ according to Equation (2.13);
5 **if** $RS(\Lambda,\Theta,t)\cap U \neq \emptyset$ **then**
6 **return** Unsafe;
7 **else**
8 **return** Safe;

Algorithm 8 for performing safety verification of uncertain linear systems relies on Theorem 2. Lines 1-2 in the algorithm checks if the condition 2.10 and 2.11 are satisfied. At line 3, A^t is calculated. From Theorem 1 we know that A^t is also an LME as it satisfies conditions condition 2.10 and 2.11. At line 4 the reachable set is computed. Finally, lines 5-7 checks if the reachable set intersects with the unsafe set. This check can be formulated as a bi-linear programming.

There are several advantages of computing the reachable set of an uncertain linear system in this manner. First, this representation is exact and is not an over-approximation. Second, computing this symbolic representation requires only matrix-matrix multiplications. Standard libraries that perform these computations are very efficient. Third, one need not recompute the reachable set for changing the uncertainty. One can simply change the set of constraints Γ and re-use the results of reachable set computation. Finally, this technique can provide counterexamples. That is, if the safety property is violated, one can diagnose the problem and provide the values corresponding to the uncertainty that lead to the safety violation. This ability to generate counterexamples is generally not possible while computing over-approximations.

Example 4. *Conditions in Equations (2.10) and (2.11) are only sufficient conditions over an LME A such that A^k is also an LME. We provide an example to demonstrate that these conditions are not necessary.*

Consider $A = \begin{bmatrix} 1+x & x \\ -x & 1-x \end{bmatrix}$. *That is,* $A = N_0 + N_1 x$ *where* $N_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ *and* $N_1 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$. *It is easy to infer that not only A^2 is also an LME but for all $t > 2, A^t$ is an LME. Furthermore,* $\mathsf{supp}(N_1) \otimes \mathsf{supp}(N_1) \neq \boldsymbol{0}$ *and hence Condition 2.10 is not satisfied.*

Observe that $A^t = N_0 + tN_1x$. *Hence, even small perturbations over the model can grow over time and significantly change the reachable set.*

Note, if a given linear dynamics is already known to satisfy conditions 2.10 and 2.11, one need not check for the conditions (lines 1-2, Algorithm 1). Computing the power (line 3, Algorithm 1) will automatically preserve the required structural properties (*i.e.* we can execute lines 3-8 by skipping 1-2 of Algorithm 1). This paper have introduced the notion of structures for two reasons. First, to understand the general principle behind the empirical observations, and second, to determine the instances to introduce uncertainties while ensuring that the robust reachable set can still be represented using quadratic constraints.

2.2.1.1 Generalizing Sufficient Conditions for Closure of LME products

Conditions 2.10 and 2.11 are restrictive and require $\mathsf{supp}(N_0) \otimes \mathsf{supp}(N_0) \leq \mathsf{supp}(N_0)$. We present a less restrictive conditions where given an LME A, A^k is also an LME.

Lemma 5. *Given LME* $A = \langle N_0, N_1, \ldots, N_k \rangle$, *if* $\exists E_0, \exists E_1, \ldots, \exists E_k$ *where, for all* $i, E_i \in B^{n \times n}$ *such that*

$$\forall i, \mathsf{supp}(N_i) \leq E_i. \tag{2.15}$$

$$\forall i, j, 1 \leq i, j \leq k, E_i \otimes E_j = \boldsymbol{0}. \tag{2.16}$$

$$\forall i, 0 \leq i \leq k, E_0 \otimes E_i \leq E_i \text{ and } E_i \otimes E_0 \leq E_i. \tag{2.17}$$

Then for all $n \geq 2$, A^n is also an LME.

Proof. This proof, similar to the proof of Theorem 1, is a proof by induction where inductive hypothesis is strengthened.

Inductive Hypothesis: Given an LME A that satisfies Equations (2.15), (2.16) and (2.17), for all $m \geq 2$, A^m is an LME $\langle L_0^m, L_1^m, \ldots, L_k^m \rangle$, and additionally, for all $i, 0 \leq i \leq k$, $\mathsf{supp}(L_i^m) \leq \mathsf{supp}(E_i)$.

The proof for the base case and the inductive step are very similar to that of proof of Theorem 1. The proof is given as a part of Appendix. □

Conditions 2.16 and 2.17 are less restrictive versions of conditions 2.10 and 2.11 respectively, because of condition 2.15. Informally, any support matrix U_i, which is a super support of N_i and satisfies conditions 2.16 and 2.17 will also satisfy conditions 2.10 and 2.11.

According to Lemma 5, if conditions 2.15, 2.16 and 2.17 are satisfied, then 2.10 can be generalized to condition 2.16 and condition 2.11 can be generalized to $\forall i, 0 \leq i \leq k,$ if $\mathsf{supp}(N_0) \otimes \mathsf{supp}(N_i) \leq \mathsf{supp}(U_i),$ and $\mathsf{supp}(N_i) \otimes \mathsf{supp}(N_0) \leq \mathsf{supp}(U_i)$

2.2.1.2 Sufficient Conditions for Time-Varying Perturbations

In this section, we extend the sufficient conditions to time-varying uncertainties. Let, the given dynamics be represented by the *LME* $A = \langle N_0, N_1, \ldots, N_k \rangle$ and the time-varying uncertainties are denoted as $Vars_T = \{y_1(t), y_2(t), \cdots, y_k(t)\}$. That is, at time $t = 1$, the LME is given as $N_0 + N_1 y_1(1) + \cdots + N_k y_k(1)$. At $t = 2$, the LME is given as $N_0 + N_1 y_1(2) + \cdots + N_k y_k(2)$.

Theorem 3. *Given an LME* $A = \langle N_0, N_1, \ldots, N_k \rangle$ *over time-varying variables* $Vars_T = \{y_1(t), y_2(t), \cdots, y_k(t)\}$*, if*

$$\forall i, j, 1 \leq i, j \leq k, \mathsf{supp}(N_i) \otimes \mathsf{supp}(N_j) = \boldsymbol{0} \tag{2.18}$$

$$\forall i, 0 \leq i \leq k, \mathsf{supp}(N_0) \otimes \mathsf{supp}(N_i) \leq \mathsf{supp}(N_i),$$

$$\textit{and } \mathsf{supp}(N_i) \otimes \mathsf{supp}(N_0) \leq \mathsf{supp}(N_i). \tag{2.19}$$

then for all $m \geq 2$*,* A^m *is an LME.*

Proof. The proof is by induction. We prove a stronger inductive property of the LMEs.

Base Case (m = 2): Due to conditions 2.18 and 2.19, A^2 will also be an *LME* (Similar proof as lemma 4). And the corresponding *LME* is: N_0^2 + $\big(N_0 N_1 y_1(2) + N_1 N_0 y_1(1)\big)$ + $\big(N_0 N_2 y_2(2) + N_2 N_0 y_2(1)\big)$ + $\cdots$ + $\big(N_0 N_k y_k(2) + N_k N_0 y_k(1)\big)$.

Rearranging the terms we get: $N_0^2 + \big(M_1^1 y_1(1) + M_2^1 y_1(2)\big) + \big(M_1^2 y_2(1) + M_2^2 y_2(2)\big) + \cdots + \big(M_1^k y_k(1) + M_2^k y_k(2)\big)$. From Equation 2.19, it follows that $\forall 1 \leq i \leq k, \forall 1 \leq j \leq 2$, $\mathsf{supp}(M_j^i) \leq \mathsf{supp}(N_i)$.

Inductive Hypothesis: Let, the LME representation be as follows: $N_0^m + \big(O_1^1 y_1(1) + O_2^1 y_1(2) + \cdots + O_m^1 y_1(m)\big) + \big(O_1^2 y_2(1) + O_2^2 y_2(2) + \cdots + O_m^2 y_2(m)\big) + \cdots + \big(O_1^k y_k(1) + O_2^k y_k(2) + \cdots + O_m^k y_k(m)\big)$. From stronger inductive hypothesis, we also have, $\forall i, 1 \leq i \leq k, \forall j, 1 \leq j \leq m$, $\mathsf{supp}(O_j^i) \leq \mathsf{supp}(N_i)$ and $\mathsf{supp}(N_0^m) \leq \mathsf{supp}(N_0)$).

Induction Step: At $m+1$ step, we will have the following representation: $N_0^{m+1} + N_0\big(O_1^1 y_1(1) + O_2^1 y_1(2) + \cdots + O_m^1 y_1(m)\big) + N_1 N_0^m y_1(m+1) + N_0\big(O_1^2 y_2(1) + O_2^2 y_2(2) + \cdots + O_m^2 y_2(m)\big) + \cdots + N_0\big(O_1^k y_k(1) + O_2^k y_k(2) + \cdots + O_m^k y_k(m)\big) + N_k N_0^m y_k(m+1)$ Therefore, this is also an *LME*. Additionally, $\forall i, i \leq i \leq k \forall j, 1 \leq j \leq m$ $\mathsf{supp}(N_0 \times O_j^i) \leq \mathsf{supp}(N_i)$ [$\mathsf{supp}(O_j^i) \leq \mathsf{supp}(N_i)$]; and, $\forall i, 1 \leq i \leq k$ $\mathsf{supp}(N_i N_0^m) \leq \mathsf{supp}(N_i)$. □

2.2.1.3 Comparing with Interval Arithmetic

In this section, we provide an example where calculating reachable set using naive interval arithmetic results in an over approximation, whereas symbolic reachable set is exact. It should also be noted, in case of interval arithmetic, if the uncertainty changes (even for a single variable), the reachable set needs to be recomputed from scratch, whereas in case of symbolic reachable set computation no re-computation is required.

Following example illustrates the over-approximation caused by interval arithmetic in reachable set computation.

Example 5. *Consider the matrix* $A = \begin{bmatrix} 2 & 4\alpha \\ 0 & -1 \end{bmatrix}$, *where* $\alpha \in [2,4]$ *corresponds to a perturbation. The uncertain matrix after performing matrix multiplication using interval arithmetic will be* $A^2 = \begin{bmatrix} 2 & 4[2,4] \\ 0 & -1 \end{bmatrix} \times \begin{bmatrix} 2 & 4[2,4] \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 4 & [0,24] \\ 0 & 1 \end{bmatrix}$ *Using symbolic computation, it will be* $A^2 = \begin{bmatrix} 4 & 4\alpha \\ 0 & 1 \end{bmatrix}$

It is easy to verify that example 5 satisfies the sufficient conditions 2.10 and 2.11.

2.2.2 Robust Reachable Set: Introducing Perturbations in the Linear Dynamics

In Section 2.2.1, we presented a set of sufficient conditions for which the reachable set of uncertain system can be represented using bi-linear inequalities. In this section, we will apply these sufficient conditions to compute the *robust reachable set*. Our procedure is as follows: Given a linear dynamical system $x^+ = Ax$, we identify the set Ψ of all indices $[i, j]$ such that, the LME obtained after replacing the indices in Ψ with a variable, satisfies the conditions presented in Section 2.2.1.

Once the set of indices Ψ is identified, we construct the uncertain linear system by introducing perturbations in the numerical values in the matrix A at indices in Ψ by a value determined by the user. That is, the user would be interested to check whether the safety property is satisfied after changing the numerical value at index i, j by $\pm 5\%$ or $\pm 10\%$.

To discover the set Ψ, we search for all Block Boolean matrices $H \in B^{n\times n}$ such that the generalized sufficient conditions are satisfied. Given H, we first construct a matrix U_H such that, for all matrices G where $\mathsf{supp}(G) \leq U_H$, we have $\big(\mathsf{supp}(G) \otimes H\big) \oplus \big(H \otimes \mathsf{supp}(G)\big) \leq H$.

Lemma 6. *Given a block matrix $N = block((r_1, c_1), (r_2, c_2))$, the corresponding $U_N \in B^{n\times n}$ is given as*

$$U_N[i,j] = \begin{cases} 0 & \text{if } ((i < r_1 \vee i > r_2) \wedge (r_1 \leq j \leq r_2)) \\ & \quad \text{or } ((j < c_1 \vee j > c_2) \wedge (c_1 \leq i \leq c_2)) \\ 1 & \text{otherwise.} \end{cases}$$

Proof. This proof has two parts. First, for the above U_N, we prove that $(U_N \otimes N) \oplus (N \otimes U_N) \leq N$. Second, for any U'_N such that $U_N \leq U'_N$ and $U_N \neq U'_N$, we show that $(U'_N \otimes N) \oplus (N \otimes U'_N) \not\leq N$.

Part 1: Given the above U_N, we first prove that $U_N \otimes N \leq N$. The proof for $N \otimes U_N \leq N$ follows similarly. To prove that $U_N \otimes N \leq N$, it suffices to prove the two parts. First, $\forall j, j < c_1$ or $j > c_2$, $(U_N \otimes N)[i, j] = 0$. Second, when $c_1 \leq j \leq c_2$, and $i < r_1$ or $i > r_2$, $(U_N \otimes N)[i, j] = 0$. Consider the element at $[i, j]$ of $U_N \otimes N$.

$$(U_N \otimes N)[i,j] = \bigvee_{k=1}^{n} U_N[i,k] \wedge N[k,j].$$

1) Since N is a block matrix with corners (r_1, c_1) and (r_2, c_2), all the elements in columns less than c_1 and greater than c_2 in N are zeros. Hence, for all $j < c_1$ or $j > c_2$, $(U_N \otimes N)[i, j] = 0$.

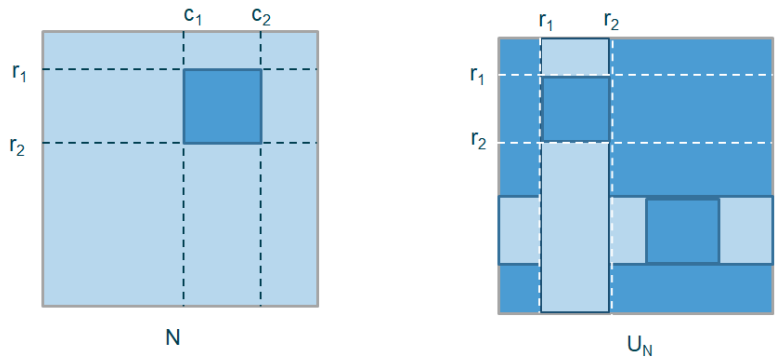


Figure 2.4: Pictorial representation of a block matrix N and its corresponding U_N. Light Blue represents 0, and Dark Blue represents 1

2) Consider $c_1 \leq j \leq c_2$ and $i < r_1$ or $i > r_2$. If $k < r_1$ or $k > r_2$ then $N[k, j] = 0$ and if $r_1 \leq k \leq r_2$, $U_N[i,k] = 0$. Therefore, the disjunction $\bigvee_{k=1}^{n} U_N[i,k] \wedge N[k, j]$ will result in 0. That is, when $i < r_1$ or $i > r_2$ and $c_1 \leq j \leq c_2$, $(U_N \otimes N)[i, j] = 0$.

Therefore, $U_N \otimes N \leq N$.

The proof for $N \otimes U_N \leq N$ follows similarly.

Part 2: Consider U'_N (different from U_N) such that $U_N \leq U'_N$, then there exists at least one i, j such that $U'_N[i, j] = 1$ and $U_N[i, j] = 0$. Let us consider the case where $(j < c_1 \vee j > c_2) \wedge (c_1 \leq i \leq c_2)$. For such an i, j, we have that $(U'_N \otimes N)[i, j] = \bigvee_{k=1}^{n} U'_N[i,k] \wedge N[k, j]$. By definition, it follows that $(U'_N \otimes N)[i, j] = 1$. However $N[i, j] = 0$. Therefore $U'_N \otimes N \not\leq N$. The proof for the case where $(i < r_1 \vee i > r_2) \wedge (r_1 \leq j \leq r_2)$ follows similarly. □

Fig. 2.4 shows a pictorial view of U_N, given N. Given a Boolean matrix N that represents the support of the system dynamics and a set of Block Boolean matrices $\{N_1, N_2, \cdots, N_k\}$ representing the support of uncertainties, we construct $[\![N_i]\!]N$ and check if $[\![N_i]\!]N \leq U_{N_i}$. If not, we can infer that introducing uncertainties at N_i might not preserve the closure under multiplication of the resulting LMEs.

Theorem 4. *Let, $N_1, N_2, \cdots, N_k$ be a set of Block Boolean Matrices, representing uncertainties in a given dynamics. If $\exists_{1 \leq i \leq k} N_i$, such that, it violates either of the following conditions:*

$$N_i \otimes N_j = \boldsymbol{0} \tag{2.20}$$

$$[\![N_i]\!]N_0 \otimes N_i \leq N_i,$$

$$\text{and } N_i \otimes [\![N_i]\!]N_0 \leq N_i. \tag{2.21}$$

then $[\![N_1, N_2, \cdots, N_k]\!]N_0$ *with the uncertainties* $N_1, \cdots, N_k$ *violates either Condition 2.10 or 2.11, or both.*

Proof. Case 1: $\exists_{1 \leq p,r \leq k}\ N_p \times N_r \neq 0$. Trivial.

Case 2: Let $N_i = block((r_1, c_1), (r_2, c_2))$. and it violates Condition 2.21 with $[\![N_i]\!]N_0$. It implies columns r_1 to r_2 excluding the rows r_1 to r_2 has a 1 in $[\![N_i]\!]N_0$, at-least in one cell; OR rows c_1 to c_2 excluding columns c_1 to c_2 has 1 in $[\![N_i]\!]N_0$, at-least in one cell (from Lemma 6).

Let us assume $[\![N_1, N_2, \cdots, N_k]\!]N_0$ along with the set of uncertainties $N_1, N_2, \cdots, N_i, \cdots, N_k$ satisfies Conditions 2.10 and 2.11.

To satisfy Condition 2.11, the 1 in columns r_1 to r_2 excluding the rows r_1 to r_2 OR rows c_1 to c_2 excluding columns c_1 to c_2 must not be present in $[\![N_1, N_2, \cdots, N_k]\!]N_0$. The only way to achieve that is, if $\exists_{j \neq i}\ 1 \leq j \leq k$ N_j intersects with columns r_1 to r_2 excluding the rows r_1 to r_2 OR rows c_1 to c_2 excluding columns c_1 to c_2 where there is a 1 in $[\![N_1, N_2, \cdots, N_k]\!]N_0$; in such a case $N_i \times N_j \neq 0$, violating condition 2.10.

Therefore, $[\![N_1, N_2, \cdots, N_k]\!]N_0$ along with the set of uncertainties $N_1, N_2, \cdots, N_i, \cdots, N_k$ does not satisfy Conditions 2.10 or 2.11 □

Corollary 2. *If* $[\![N_1, N_2, \cdots, N_k]\!]N_0$ *along with the set of uncertainties* $N_1, N_2, \cdots, N_k$ *satisfy Conditions 2.10 and 2.11 then* $\forall_{1 \leq i \leq k} N_i$ *will satisfy Conditions 2.20 and 2.21 with* $[\![N_i]\!]N_0$

It is easy to observe that converse of Theorem 4 does not hold true *i.e.* if $N_1, N_2, \cdots, N_k$ are a set of Block Boolean Matrix, representing uncertainties in a given dynamics and $\forall_i\ N_i$ satisfies Conditions 2.20 and 2.21 with $[\![N_i]\!]N_0$ then it does not imply $[\![N_1, N_2, \cdots, N_k]\!]N_0$ along with the set of uncertainties $N_1, N_2,$ $\cdots, N_k$ will satisfy Conditions 2.10 and 2.11.

We leverage Theorem 4 to search for block uncertainties in a given dynamics, such that Conditions 2.10 and 2.11 are satisfied. Given a block size of $p \times r$, we look for all block uncertainties N_i of size $p \times r$ such that they satisfy Conditions 2.20 and 2.21 with $[\![N_i]\!]N_0$ and put it in a set κ. The maximal subset of κ is the maximum number of uncertain blocks of size $p \times r$ that can be induced in A so that conditions 2.10 and 2.11 are satisfied. From Theorem 4 we know if a block N_i of size $m \times n$ is not in κ, it can never be in any of the sets of block uncertainties that satisfies condition 2.10 and 2.11.

We use Lemma 6 and Theorem 4 for introducing uncertainties in the given linear dynamics $x^+ = Ax$ such that it satisfies the sufficient conditions in Lemma 5. We search for all block matrices M such that $N \times N = \mathbf{0}$. Let, $[\![N]\!]N_0$ be a matrix representing the constants coefficients after excluding all the block matrices N. Then we check if $\mathsf{supp}([\![N]\!]N_0) \leq U_N$ and add them to a set κ. We consider all possible subsets of κ and check

if the sufficient conditions in Lemma 5 are satisfied. We collect the subset with the maximum number of uncertainties that can be introduced and return the corresponding LME. The algorithm is formally given in Algorithm 2.

Algorithm 2: Algorithm that searches for all possible LMEs that satisfy the sufficient conditions in Lemma 5 and returns the LME with maximum number of uncertainties.

input : Linear dynamical system $x^+ = Ax$
output : Linear Matrix Expression $A' = \langle N_0, N_1, \ldots, N_k \rangle$ that satisfies conditions in Lemma 5.

```
1  κ ← ∅;
2  maxPerturbations ← 0;
3  maxS ← ∅;
4  N_0 ← supp(A);
5  for all blocks N = block((r_1,c_1),(r_2,c_2)) of size less than n × n in the matrix A do
6      if supp([[N]]N_0) ≤ U_N and N × N = 0 and U_N × U_N ≤ U_N then
7          κ ← κ ∪ {N};
8  if κ = ∅ then
9      return Cannot introduce uncertainties;
10 for all subsets S of κ do
11     S = {N_1, N_2, ..., N_k};
12     U_0 = ⋂ U_{N_i};
13     if ∀i, j, 1 ≤ i, j ≤ k, H_i × H_j = 0 then
14         if U_0 × U_0 ≤ U_0, ∧ supp(N_0) ≤ U_0 then
15             if size(S) > maxPerturbations then
16                 maxPerturbations ← size(S);
17                 maxS ← S;
18 return ⟨A, N_1, ..., N_k⟩ where maxS = {N_1, N_2, ..., N_k}.
```

Description of Algorithm 2: Lines 4-6, searches for all the blocks N in the given dynamics A such that it satisfies Conditions 2.20 and 2.21 with $[\![N]\!]N_0$.

Lines 7-8: If there are no blocks found in the previous step (Lines 4-6) that satisfies Conditions 2.20 and 2.21 then, no fault introduction is possible which satisfies Conditions 2.10 and 2.11; this follows from Theorem 4.

Lines 9-16: We look for the largest subset of κ that satisfies Conditions 2.10 and 2.11. From Theorem 4 this set is guaranteed to be the largest set which satisfies Conditions 2.10 and 2.11. The conditions checked in line 13 if satisfied implies the conditions mentioned in Lemma 5. Therefore, the subset S satisfies all the sufficient conditions.

Line 17: Returns the maximal subset $\{N_1, N_2, \cdots, N_k\}$ that satisfies the Conditions 2.10 and 2.11.

For a particular subset S: line 12, has a run time of $O(n^2 \cdot k)$, where A is of size $n \times n$; the check in line 13

takes $O(k^2)$, as checking $H_i \times H_j$ $O(1)$, and there are $O(k^2)$ of them; checks in line 14 takes $O(n^2)$.
This shows that if the uncertain blocks are given by the user *i.e.* uncertain blocks are already known, we can check if they satisfy Conditions 2.10 and 2.11 very efficiently. Moreover this check needs to be performed only once at the start. And then the safety verification can be performed using bi-linear optimization as discussed.

Observations: Notice that the Algorithm 2 searches through all possible block matrices (potentially n^4) to search for matrices that satisfy sufficient conditions and then searches for all possible subsets of the set κ (potentially 2^{n^4}). For each subset $s = \{N_1, N_2, \cdots, N_k\}$, we check if $\mathsf{supp}(N_0) \leq U_0$. Where N_0 is the coefficient matrix, excluding all N_i; $U_0 = \bigcap U_{N_i}$ and U_{N_i} are obtained according to Lemma 6. However, due to the restriction that $N \times N = 0$ and the condition that $\mathsf{supp}(N_0) \leq U_0$, we observe that, in practice, the set κ often contains the subsets of the order $O(n)$. Hence, we found that our approach to be useful in several benchmarks with dimensions ranging from 5 to 20.

2.2.2.1 Illustration

In this section we show how the experiments were performed with the help of a toy example. In the rest of the section we evaluate the applicability of our approach on several benchmarks. In each benchmark we provide all the set of inputs required to test the applicability of our approach. Let us consider the following example:

$$\begin{bmatrix} x_1^+ \\ x_2^+ \\ x_3^+ \\ x_4^+ \end{bmatrix} = \begin{bmatrix} 3 & 2.9 & 3.9 & 2 \\ 0 & 7 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

In the above example, the values in magenta (2.9) and brown (3.9) are uncertain. The different colors symbolize that these uncertainties are independent. Let, the initial set $\Theta \triangleq [1,2] \times [2,2] \times [3,3] \times [1,1]$. Suppose that the uncertainty associated with the value in magenta be $\pm 10\%$ and the value in brown be $\pm 20\%$. The system is considered to be safe if at every step the value of $x_1 < 100$. For performing safety verification of the uncertain linear system, the following steps are performed.

- A is represented as the following LME; the variables y and z correspond to the uncertainties.

$$\underbrace{\begin{bmatrix} 3 & 0 & 0 & 2 \\ 0 & 7 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{N_0} + \underbrace{\begin{bmatrix} 0 & 2.9 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{N_1} y + \underbrace{\begin{bmatrix} 0 & 0 & 3.9 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{N_2} z$$

- We now check the sufficient conditions in Equations (2.10) and (2.11). That is, the following conditions are checked:
  - Check $\mathsf{supp}(N_1) \times \mathsf{supp}(N_2) = 0$, $\mathsf{supp}(N_2) \times \mathsf{supp}(N_1) = 0$, $\mathsf{supp}(N_1) \times \mathsf{supp}(N_1) = 0$, and $\mathsf{supp}(N_2) \times \mathsf{supp}(N_2) = 0$.
  - Compute $U_N = U_{N_1} \cap U_{N_2}$. U_{N_1} and U_{N_2} are obtained based on Lemma 6. Check if $\mathsf{supp}(N_0) \leq U_N$ and $U_N \times U_N \leq U_N$

  All the above conditions are satisfied for this example. Hence, from Theorem 1, we know that for all t, A^t is an LME.

- We now compute A^2 as

$$\begin{bmatrix} 9 & 0 & 0 & 8 \\ 0 & 49 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 29 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} y + \begin{bmatrix} 0 & 0 & 19.5 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} z$$

- For checking the safety property of $x_1 < 100$, we give the following constraints from the reachable set to Gurobi.

$$x_1 - (9\alpha_1 + 29y\alpha_2 + 19.5z\alpha_3 + 8\alpha_4) = 0$$
$$x_1 \geq 100$$
$$1 \leq \alpha_1 \leq 2,\ 2 \leq \alpha_2 \leq 2,\ 3 \leq \alpha_3 \leq 3,\ 1 \leq \alpha_4 \leq 1$$
$$0.9 \leq y \leq 1.1,\ 0.8 \leq z \leq 1.2$$

where α_i represents the constraints on x_i in the initial set and y and z represents the uncertainties.

- If Gurobi returns that the above set of equations are feasible, then we report that the system is unsafe. Else, the system is safe. Since most of these conditions are linear and contain only a few quadratic constraints (that too only product of variable terms), Gurobi can quickly solve such instances.

2.2.3 Ordering Uncertainties Based on their Impact on Safety

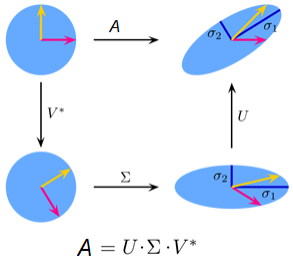
In this section, we discuss several other artifacts that provide more understanding on types and effects of various classes of uncertainties on a linear system. In particular, we propose an algorithm to order uncertainties based on its impact on safety. Further, leveraging this ordering, we propose heuristics to compute robustness threshold (*i.e.*, amount of uncertainty that can be tolerated by the system without violating safety).

2.2.3.1 Relationship between Reachable Sets and Singular Values

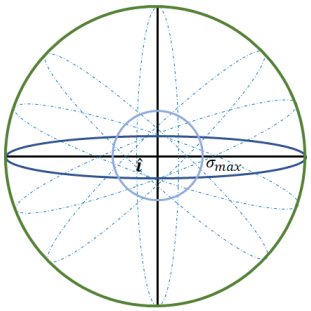
In this section, we demonstrate the relationship between the reachable set of a linear system and its corresponding singular values. We first show that the widening of the reachable set (along particular directions), when viewed as a linear transformation, is dependent on the singular values. We then provide a template based overapproximation of the reachable set that is dependent on the maximum singular values. This relationship between the widening of the reachable set and the singular values would help us understand the effect of specific uncertainties on the reachable set.

Remark. Given a set $\theta = \mathcal{S}(1)$ (hyper-sphere of unit radius) and a linear transformation A, the change in the width of the set θ (in directions determined by the SVD of A) after the linear transformation, is equal to the singular values of A. That is, the width of the set $A\theta$, in evaluated directions, is equal to the singular values.

Consider the singular value decomposition $U\Sigma V^*$ of the matrix A. The linear transformation can be understood as a sequence of three linear transformations V^*, followed by Σ, followed by U. Since U and V^* are unitary matrices, the transformation does not cause the set to widen in any direction. Whereas, the widening of the set $A\theta$, in evaluated directions from U and V^*, to the set θ is equal to the singular values of A. More specifically, the transformation $A\theta$ is an ordered sequence of rotation, scaling, and rotation. That is, V^* is a rotation, Σ is a scaling, and U is a rotation. This is illustrated in Fig. 2.5a.



(a) Geometric interpretation of SVD. [23]



(b) Illustration of the overapproximation being contained in $\mathcal{S}(\sigma^A_{max})$. Purple hypersphere: $\mathcal{S}(1)$ the initial set that is invariant to transformation by unitary matrix. Blue ellipse: $\mathcal{S}(1)$ after applying the transformation Σ causes scaling along various dimensions. Green hypersphere: $\mathcal{S}(\sigma^A_{max})$ that contains all possible transformations U of the blue ellipse.

Figure 2.5: Reachable sets and singular values.

We now show that a template based overapproximation of the set $A\theta$ is proportional to the maximum singular value of A. We illustrate this by performing a linear transformation on a unit sphere $\mathcal{S}(1)$ and show that the overapproximation of $A\mathcal{S}(1)$ is contained within a sphere of radius σ^A_{max}, the maximum singular value of A.

Lemma 7. *Given the unit sphere $\mathcal{S}(1)$ and a matrix A, we have $A \times \mathcal{S}(1) \subseteq \mathcal{S}(\sigma^A_{max})$, where σ^A_{max} is the maximum singular value of the matrix A.*

Proof. This proof hinges on the geometric interpretation of the singular value decomposition. Consider the SVD of A as $U\Sigma V^*$. Since a unit sphere is symmetric, the linear transformation of $V^*\mathcal{S}(1)$ (*i.e.* a rotation) is same as $\mathcal{S}(1)$. The transformation of Σ, a diagonal matrix with singular values, scales (widens) the i^{th} axis by the corresponding singular value σ_i, resulting in an ellipsoid whose axes are defined by the singular values. The transformation of the ellipsoid with U (*i.e.* another rotation) will reorient the axes but not cause any further widening. Therefore, the maximum distance of any point on $A\mathcal{S}(1)$ is bounded by σ^A_{max}, the maximum singular value of A. Hence, $A\mathcal{S}(1) \subseteq \mathcal{S}(\sigma^A_{max})$. This has been illustrated in Fig. 2.5b. □

In this section, we are interested in understanding the effect of matrix uncertainties on the reachable set. Given an uncertain system $\dot{x} = \Lambda x$, from Lemma 7, we know that the matrix with the maximum singular value of the matrix exponential would have the largest effect vis-à-vis widening on the reachable set—this has been formalized in Theorem 5. *Therefore, the uncertainties that will increase the maximum singular value would cause maximum increase in the width of the overapproximation of the reachable set.* Therefore, for evaluating the robustness of safety of a linear dynamical system, we introduce perturbations that cause largest change in the maximum singular value. In the following theorem, we show that the reachable set, viewed

as a transformation, can be computed using just the maximum singular value of the Max SV Candidate (Definition 13).

Theorem 5. *Given a transformation $f(\theta) = \Lambda\theta$ where $\Lambda \subset \mathbb{R}^{n\times n}$; and $\theta = \mathcal{S}(1)$, then $f(\theta) \subseteq \mathcal{S}(\sigma_{max})$.*

Proof. Follows from Lemma 7. □

2.2.3.2 Ordering the cells of the matrix based on its sensitivity to perturbation

In this section, we study the effect of introducing uncertainties in different cells of a linear dynamical system. From Theorem 5 we know that by studying the effect of an uncertainty on the singular value, we can understand the effect of that uncertainty on the magnitude of change in the reachable set.

The effect of perturbations on the singular values of a matrix has, fortunately, already been studied in [24]. In [24], given matrices A and B, the singular values of the matrices $A + \varepsilon B + O(\varepsilon^2)$ are shown to have the form $\sigma_i + k_i\varepsilon + O(\varepsilon^2)$. Additionally, [24] also provides the closed form expression to compute k_i as a function of A and B. Therefore, the matrix with largest k_i causes the largest effect in the singular values as a result of perturbation.

Using the closed form expression provided in [24], we can introduce a perturbation in each cell of A and study its effect on the singular value. To introduce a perturbation of ε in the $[i,j]^{th}$ element of the matrix A and observe the change in singular values, we use the expression in [24] with matrix B as a zero matrix except for the $[i,j]^{th}$ element being 1. A similar technique for constructing B where $B[i,j] = A[i,j]$ can be used to study the effect of relative perturbation on singular values. If perturbation in a cell (i,j) has more impact on the singular value *vis-à-vis* the reachable set than a cell (k,l), then Algorithm 3 orders (i,j) before (k,l).

Algorithm 3 ranks the cells of the matrix in decreasing order of relative change in the singular values by perturbation. That is, a perturbation in the first element of *SingularOrder* causes the maximum change in the singular values of the matrix A (thus on the reachable set). Since we are interested in *relative perturbation*, *i.e.*, the elements of the matrix A are subjected to relative uncertainty (such as $1\%, 5\%$, etc.), we initiate $B[i,j] = A[i,j]$ (and rest of the elements as 0) in Algorithm 3.

For each cell $[i,j]$ of the matrix, Algorithm 3 constructs the matrix B that is all zeros, except for the cell $[i,j]$ and invokes the closed form expression provided in [24] with the pair (A,B). This expression will determine the change in maximum singular value of A after applying ε relative perturbation. The magnitude

Algorithm 3: Algorithm to find an ordering on the cells of the matrix A with decreasing sensitivity to perturbation

$B \leftarrow zeros[n][n]$
for $1 \leq i, j \leq n$ **do**
    $B[i,j] = A[i,j]$;
    $Ord[i,j] \leftarrow$ maximumSingularValueChange(A,B) ▷ (Eq (3.8) in [24]) ;
    $B[i,j] = 0$;
$SingularOrder \leftarrow$ Sort (i,j) based on decreasing value of $Ord[i,j]$;
return *SingularOrder*;

of this change is stored in $Ord[i,j]$ (Algorithm 4). All the cells of the matrix are sorted according to the decreasing order of the effect on maximum singular value.

2.2.3.3 Computing Robustness Threshold

We define *robustness threshold* of a linear dynamical system as: *given a set of cells of the dynamics matrix and an unsafe set, the maximum amount of perturbation that can be introduced to the dynamics while being safe*.

Definition 23. *Given a dynamical system $\dot{x} = Ax$, initial set Θ, unsafe set U, and time bound T, the robustness threshold r is given as:*

$$r = max_{||\Lambda||}\{\Lambda \subseteq \mathbb{R}^{n\times n} \mid \forall A', A' \in A + \Lambda, \forall t, t \in [0,T], e^{A't}\Theta \cap U = \emptyset\} \tag{2.22}$$

Given that matrix exponential is a nonlinear function, performing the quantifier elimination and computing the exact value of robustness threshold *exactly* is impractical. In this section, we provide empirical techniques to approximate the value of robustness threshold. Our procedure follows a numerical search, i.e., we first determine an uncertainty Λ such that the uncertain linear system is safe and then progressively increase the uncertainties, until a safety violation is discovered. Instead of computing the matrix exponential for all the matrices in the uncertain linear system, we compute the overapproximation of the reachable set.

We provide two heuristics for numerically estimating the value of robustness threshold in this section. Both the proposed heuristics rely on the effect of a perturbation on the maximum singular value as computed in Algorithm 3. We first pick an initial value for the budget of perturbation (say 1% relative perturbation to be distributed in the given cells). Given a bound on the total perturbation that can be introduced in a dynamics,

the perturbation should be distributed such that: the cells that have larger effect on the maximum singular value receive a smaller proportion of the total perturbation and vice-versa. Recall that *Ord* matrix, computed in Algorithm 3, measures the effect of the cell $[i, j]$ on the maximum singular value. Therefore, the higher the value of $Ord[i, j]$, the lower is the amount of perturbation that can be tolerated while keeping the system safe.

We distribute this budget among the various cells according to two heuristics. The first heuristic, `Proportional Distribution` is such that the perturbation introduced in cell $[i, j]$ is proportional to the values of $Ord[i', j']$ that are less than $Ord[i, j]$. The second heuristic, `Harmonic Distribution` is such that the perturbation introduced in cell $[i, j]$ is proportional to $\frac{1}{Ord[i][j]}$. We use the baseline of `Equal Distribution` where the same amount of perturbation is introduced in all cells, irrespective of their effect on maximum singular value.

Once the perturbations for each cell is allocated, we compute the overapproximation of the reachable set for the uncertain linear system and infer safety. If the uncertain system is safe, we increase the budget by a constant value (say addition of 10%) for perturbation and continue. We terminate when any further increase in the perturbation budget will cause the system to be unsafe. This procedure is formally described in Algorithm 4.

Algorithm 4: Heuristic to compute robustness metric of linear dynamics

```
p ← 0 ▷ (perturbation budget) step ← user chosen parameter ▷ (increase in perturbation after each
  iteration) Λ_old={};
while True do
    Λ ← Perturb cells A according to budget p and user determined distribution;
    if U ∩ ORS_t(A + Λ) ≠ ∅ then
        return || Λ_old ||
    p = p + step  Λ_old = Λ
```

2.3 Safety Verification of Autonomous Systems in the Presence of Uncertainties

2.3.1 Deterministic Approaches

In this section, we consider a model with uncertainties—we define continuous time uncertain linear systems as $\dot{x} = \Lambda_u x$, where the uncertainties in the model are considered as variables in Λ_u that can take values from a bounded range. Interval matrix, a matrix where elements are intervals, is a convenient representation of such a matrix Λ_u. The uncertain linear system $\dot{x} = \Lambda_u x$ can equivalently be represented by separating

the nominal and the perturbed dynamics as $\dot{x} = (A + \Lambda)x$, where $\Lambda_u = A + \Lambda$. *In this section, we discuss two reachable set computation techniques of uncertain linear systems: one using perturbation theory (by extending results from classical perturbation theory, and interval matrix analysis), and the other using set based numerical method.* In particular, the contributions of this section are follows:

1. A symbolic approach, using classical approaches from perturbation theory, which is computationally very fast.

2. A set based representation (numeric) approach, where the reachable set is represented as a Minkowski sum of its unperturbed part with the effect of perturbation. We further discuss two methods to improve the performance of our set based representation method.

3. We provide two extensive case studies, from two different domains, to demonstrate the broad-spectrum applicability of the proposed artifacts.

2.3.1.1 Reachability Using Perturbation Analysis

In the early days of studying numerical techniques for solving ordinary differential equations, there was a significant interest in bounding the errors introduced by using finite precision representation of real numbers. Seminal works in this domain, such as [25] and [26], investigated analytical methods for computing upper bounds on the sensitivity of matrix exponential. Given a linear dynamical system $\dot{x} = Ax$ and its perturbation $\dot{x} = (A + E)x$, these methods presented expressions to upper bound the relative distance between the trajectories of the nominal (unperturbed) and the perturbed system. The relative distance, denoted by $\phi_{(A,E)}(t)$, is given as:

$$\phi_{(A,E)}(t) = \frac{|| \, e^{(A+E)t} - e^{At} \, ||}{|| \, e^{At} \, ||}. \tag{2.23}$$

Notice that [25] and [26] were applicable for specific matrices A and E, since we deal with bounded model uncertainties, we extend the same definition from a matrix E to an interval matrix Λ. We extend the notation to include interval matrices as follows:

$$\phi_{(A,\Lambda)}(t) = sup_{E \in \Lambda} \frac{|| \, e^{(A+E)t} - e^{At} \, ||}{|| \, e^{At} \, ||}. \tag{2.24}$$

Using the analytical expressions for $\phi_{(A,E)}(t)$ that were presented in [25] and [26], we provide analytical expressions for $\phi_{(A,\Lambda)}(t)$, involving interval matrices. To compute $\phi_{(A,\Lambda)}(t)$, we leverage some of the results on interval matrix norms that are provided in [20].

Lemma 8. ***[From [25], Table 4.1 (4.14)]*** *Given a matrix A and perturbation $E \in \mathbb{R}^{n\times n}$,*

$$\phi_{(A,E)}(t) \leq p_{n-1}(||A||_2 t) \times \Big(exp\big(p_{n-1}(||\ A\ ||_2\ t)\ ||\ E\ ||_2\ t\big) - 1\Big). \tag{2.25}$$

where $p_{n-1}(x) = \Sigma_{k=0}^{n-1} \frac{x^k}{k!}$,

Notice from Lemma 8 that $\phi_{(A,E)}(t)$ is monotonically increasing with $||\ \ E\ \ ||_2$. Therefore, to extend this analysis to interval matrices, one merely needs to compute the supremum of the set of 2-norms of all the matrices in the interval matrix.

Lemma 9. $\sup\limits_{E\in\Lambda} \{||\ E\ ||_2\} = ||\ \Lambda\ ||_2$.

As $||\ \Lambda\ ||_2$ is obtained using Theorem 7 of [20], where $||\Lambda||_2$ is defined in Definition 11, the above lemma holds true.

Theorem 6. $\phi_{(A,\Lambda)}(t) \leq p_{n-1}(||\ A\ ||_2\ t)(exp(p_{n-1}(||\ A\ ||_2\ t)\ ||\ \Lambda\ ||_2\ t) - 1)$.

Proof. Observe that $\phi_{(A,E)}(t)$ (in Lemma 8) monotonically increases with $||\ E\ ||_2$, and Lemma 9 states that the 2 norm of the interval matrix Λ is the supremum of 2-norms of all the matrices in the interval matrix. Therefore, we have $\phi_{(A,\Lambda)}(t) \leq p_{n-1}(||\ A\ ||_2\ t)(exp(p_{n-1}(||\ A\ ||_2\ t)\ ||\ \Lambda\ ||_2\ t) - 1)$. □

We denote the upper bound provided in Theorem 6 as `Kagstrom1`.

In the next two Theorems, we provide two different upper bounds of $\phi_{(A,\Lambda)}(t)$ than in Theorem 6.

Lemma 10. ***[From [25], Table 4.1 (4.12)]*** *Given a matrix A and perturbation $E \in \mathbb{R}^{n\times n}$,*

$$\phi_{(A,E)}(t) \leq K(SD)e^{\varepsilon t}(e^{K(SD)\cdot||E||_2 t} - 1), \tag{2.26}$$

where (a) for a matrix M, $K(M)$ is its condition number $||\ M\ ||\cdot||\ M^{-1}\ ||$, (b) SJS^{-1} is the Jordan form of the matrix A, and (c) D is a diagonal matrix s.t. $||\ D^{-1}JD\ ||_2 \leq \varepsilon$.

Theorem 7. $\phi_{(A,\Lambda)}(t) \leq K(SD)e^{\varepsilon t}(e^{K(SD)\cdot||\Lambda||_2 t} - 1)$*; where K, S, D, and ε are as defined in Lemma 10.*

Proof. Similar to the proof of Theorem 6—follows from monotonicity of $\phi_{(A,E)}(t)$ (Lemma 10) with respect to $|| E ||_2$, and Lemma 9. □

We denote the upper bound provided in Theorem 7 as `Kagstrom2`.

Lemma 11. ***[From [26], Theorem 1]*** *Given a matrix A and perturbation $E \in \mathbb{R}^{n\times n}$*

$$\phi_{(A,E)}(t) \leq t \; || \; E \; ||_2 \; e^{(||A||_2 - \alpha(A) + ||E||_2)t}, \tag{2.27}$$

where, for any square matrix M, $\alpha(M)$ is the largest real component of the eigenvalues of M.

Theorem 8. $\phi_{(A,\Lambda)}(t) \leq t \; || \; \Lambda \; ||_2 \; e^{(||A||_2 - \alpha(A) + ||\Lambda||_2)t}$.

Proof. Similar to the proof of Theorem 7. □

We denote the upper bound provided in Theorem 8 as `Loan`.

Note that [25] and [26] contain many such formulas for computing an upper bound on $\phi_{(A,E)}(t)$. This thesis focused on three of these specific formulas primarily because they yielded the best results on the standard verification benchmarks. *Theorem 6, Theorem 7 and Theorem 8 summarize the new contributions of this section in extending the perturbation theory based techniques.* Computing the reachable set of a linear system with uncertainty is, therefore, a two step process. First, we compute the reachable set of the dynamical system $\dot{x} = Ax$. Second, we compute an upper bound on the relative error for the bounded uncertainties (i.e. $\phi_{(A,\Lambda)}(t)$) and *bloat* the reachable set computed in the first step. It is easy to prove that this procedure results in an overapproximation of the reachable set for linear systems with uncertainties. The reachable set computation technique, proposed in this section, is computationally very efficient—one merely needs to compute the bloating factors, as given in Theorem 6, Theorem 7 and Theorem 8, and bloat the reachable set of the nominal dynamics to obtain the reachable set of the uncertain system. The evaluation on several (low and high dimensional) benchmarks show that the reachable set of a uncertain linear system can be computed in just few seconds (less than a second for most of the benchmarks). Next, we extend these results (Theorem 6, Theorem 7 and Theorem 8) to use Interval Matrix Frobenius Norm, as it is computationally more efficient [20], compensating the quality of overapproximation (see Section 2.4.1 for details).

Theorem 9. *(Re-writing Theorem 6)*

$\phi_{(A,\Lambda)}(t) \leq p_{n-1}(|| \; A \; ||_2 \; t)(exp(p_{n-1}(|| \; A \; ||_2 \; t) \; || \; \Lambda \; ||_F \; t) - 1)$.

Theorem 10. *(Re-writing Theorem 7)*

$\phi_{(A,\Lambda)}(t) \leq K(SD)e^{\varepsilon t}(e^{K(SD)\cdot||\Lambda||_F t} - 1)$ *where K, S, D, and* ε *are as defined in Lemma 10.*

Theorem 11. *(Re-writing Theorem 8)*

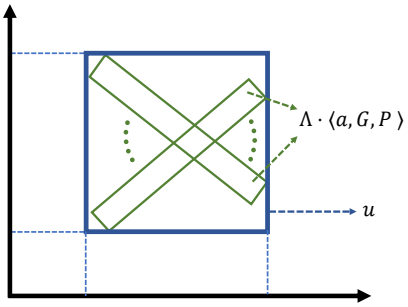
$\phi_{(A,\Lambda)}(t) \leq t \;||\; \Lambda \;||_F \; e^{(||A||_2 - \alpha(A) + ||\Lambda||_F)t}$.

In all our experiments, we compute 2-norm of interval matrices (Theorem 6, Theorem 7 and Theorem 8) when it is easy to compute (for low dimensional systems). However, for high dimensional systems, we use Frobenius norm of interval matrices (Theorem 9, Theorem 10, Theorem 11).

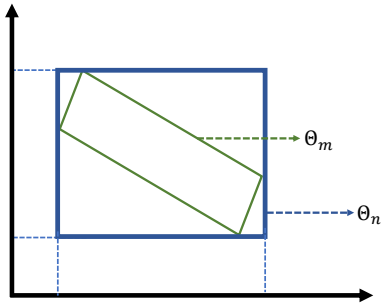
We finally conclude this section with the following discussion. Since the perturbation analysis based technique proposed in this paper is oblivious of the structure of the matrix and merely uses norm sensitivity of matrix exponentials, it bloats the nominal reachable set in all directions. Our experiments show (provided later) that for smaller values of t, the bloating factor obtained is reasonable but blows up for higher values of t (as is expected). Improving the tightness of this approach—in particular, considering direction specific bloating factors—is left as a future work. It might require us to move away from merely studying sensitivity of matrix exponentials with respect to norms, to a more structure aware metric. However, computing reachable sets using this technique, thanks to perturbation theory, is extremely computationally efficient—one merely needs to compute the bloating factor using the provided expressions, and bloat the nominal (unperturbed) reachable set. Such techniques might be useful in monitoring of systems [12, 27, 28, 29]—both for online and offline where time is of the essence or when timesteps between the logs are reasonably small. The approach we propose in the next section that relies on a discretized dynamics, does not suffer from such issues—enabling users to compute tighter reachable sets.

2.3.1.2 Reachability Using Set Representations

In reachable set computation techniques, a suitable representation (such as Zonotopes [30], Support functions [31], or Stars [32]) is chosen to represent the set of states reached by a dynamical system. For efficient computation of reachable set, a time step h is chosen, and the linear dynamical system $\dot{x} = Ax$ is converted into a discrete time dynamical system $x^+ = e^{Ah}x$. Given an initial set Θ in the representation, the linear transformation of Θ, *i.e.*, $e^{Ah}\Theta$ is computed in the same representation. For scenarios where the application is modelled using continuous time, reachable sets computed using discretized dynamics lack formal soundness. However, for systems where the behavior does not change faster than a given frequency



(a) Illustration of Theorem 12. The set of stars $\Lambda \cdot \langle a, G, P \rangle$ being approximated by a hyperbox u (represented as a star), by projecting it onto orthonormal axis.



(b) Illustration of Theorem 13. The stars Θ_m being approximated by a hyperbox Θ_n (represented as a star), by projecting it onto orthonormal axis.

Figure 2.6: Illustration of Theorem 12 and Theorem 13.

(*e.g.*, the processor clock), we can find a granularity small enough for the discrete time evolution. Moreover, discrete time systems are actively being used in research, such as in distributed embedded controllers [33, 34], studying cyber-attacks [35], modelling network forecasts [36].

In the case of uncertain linear systems, *i.e.*, $\dot{x} = (A + \Lambda)x$, we perform the same thing and compute a linear uncertain transformation as $x^+ = e^{(A+\Lambda)h}x$. Using widely known techniques for computing matrix exponential [37, 38], we compute a discrete time uncertain linear system, where $x^+ = (\bar{A} + \bar{\Lambda})x$. Since the reachable set computation technique, proposed in this section, only relies on discrete transformation, in this section, we abuse notation and represent the discrete time uncertain linear system as $x^+ = (A + \Lambda)x$. In this section, we use star representation [39] for computing reachable sets of uncertain linear systems. We overapproximate the effect of model uncertainties as time varying environmental inputs, and represent the reachable set as a Minkowski sum of the reachable set without inputs and the effect of inputs, as presented in [40]. We also provide two new methods to improve the scalability of this approach and evaluate this method on a wide variety of benchmarks.

Computation of Reachable Sets for Uncertain Linear Systems using Stars. To perform the linear uncertain transformation $x^+ = (A + \Lambda)x$ over a star $\langle a, G, P \rangle$, our approach considers Λ as an additional input. That is, $(A + \Lambda) \cdot \langle a, G, P \rangle$ is computed as $A \cdot \langle a, G, P \rangle \oplus \Lambda \cdot \langle a, G, P \rangle$. The former part of the summation $\big(A \cdot \langle a, G, P \rangle\big)$ is easier to compute. For the latter part, $\Lambda \cdot \langle a, G, P \rangle$, we impose a specific assumption on P, and compute a set u, *s.t* $\Lambda \cdot \langle a, G, P \rangle \subseteq u$. Note that, as Λ is an interval matrix, $\Lambda \cdot \langle a, G, P \rangle$ is a set of stars, *i.e.*, $\Lambda \cdot \langle a, G, P \rangle = \bigcup_{E \in \Lambda} \langle E \cdot a, E \cdot G, P \rangle$. Using Theorem 12, we overapproximate $\Lambda \cdot \langle a, G, P \rangle$ (a set of stars) with a star u, *i.e.*, $\Lambda \cdot \langle a, G, P \rangle \subseteq u$.

Theorem 12. *Given an interval matrix Λ and a star $\langle a, G, P\rangle$, let $\Lambda \cdot \langle a, G, P\rangle = \bigcup_{E \in \Lambda} E \cdot \langle a, G, P\rangle$. Also consider that $\forall \alpha_1, \ldots, \alpha_m$ such that $P(\alpha_1, \ldots, \alpha_m) = \top$, the upper and lower bounds on α_j are c^j_{low} and c^j_{high}. We have $\Lambda \cdot \langle a, G, P\rangle \subseteq u$, where $u = \langle \bar{0}, I_n, P'\rangle$: (a) $\bar{0}$ represents the origin, (b) I_n is the collection of n orthonormal unit vectors that are axis aligned, and (c) P' is a conjunction of box aligned constraints such that $d^i_{low} \leq \alpha'_i \leq d^i_{high}$, where $[d^i_{low}, d^i_{high}]$ is the interval obtained by performing $(\Lambda \cdot a)[i] + \Sigma^m_{j=1}(\Lambda \cdot G)[i][j] \cdot [c^j_{low}, c^j_{high}]$ using interval arithmetic.*

Proof. Using the expansion of the definition of stars and the provable overapproximation of interval arithmetic, we use contradiction to prove this theorem. Let there be an $1 \leq i \leq n$ for which $\Lambda \cdot \langle a, G, P\rangle \not\subseteq \langle \bar{0}, I_n, P'\rangle$. This would imply $[d^i_{low}, d^i_{high}] \not\subseteq [low(\alpha'_i), high(\alpha'_i)]$, which violates the assumption, where $low(\cdot)$ and $high(\cdot)$ denotes the lower and upper bound respectively. □

Intuitively, Theorem 12 overapproximates $\Lambda \cdot \langle a, G, P\rangle$ with an axis-aligned hyperbox u, *i.e.*, $\Lambda \cdot \langle a, G, P\rangle \subseteq u$—this is illustrated in Fig. 2.6a. Therefore, using Theorem 12, given an interval matrix Λ and a star $\langle a, G, P\rangle$, one can easily compute a u, *s.t* $\Lambda \cdot \langle a, G, P\rangle \subseteq u$. Using Theorem 12, we can overapproximate the linear uncertain transformation (*i.e.*, $A + \Lambda$) of a star θ (say) as a Minkowski sum of two stars. Let $A \cdot \theta \oplus u_0$ be the overapproximation of the reachable set after one transformation, where $\Lambda \cdot \theta \subseteq u_0$ (u_0 is obtained using Theorem 12). Applying the transformation of $(A + \Lambda)$ for the second time would yield $A^2 \cdot \theta \oplus A \cdot u_0 \oplus u_1$, where u_1 is obtained by applying Theorem 12 on $\Lambda(A \cdot \Theta \oplus u_0)$. The recursive relationship that represents the overapproximation of the reachable set, at time step k, is given as follows:

$$\begin{aligned} ORS_k &= \theta \text{ (initial set)} \qquad \text{if } k = 0 \\ &= A \cdot ORS_{k-1} \oplus u_k \quad \text{otherwise.} \end{aligned} \tag{2.28}$$

This is graphically represented in Fig. 2.7. It also follows from Theorem 12 that $(A + \Lambda)^k \cdot \theta \subseteq ORS_k$.

Methods to improve performance The set representation based method performs Minkowski sum of two stars, at every step, to compute the overapproximate reachable set. The reachable set at time step $k+1$ is given by $ORS_{k+1} = A \cdot ORS_k \oplus u_k$. Due to the Minkowski sum of two stars, at every time step, we increase the number of basis vectors for representing the overapproximate reachable set by n (where n is the dimension of the system). The number of basis vectors increases by n at each step, because u_k is represented using n basis vectors (as it is computed using Theorem 12).

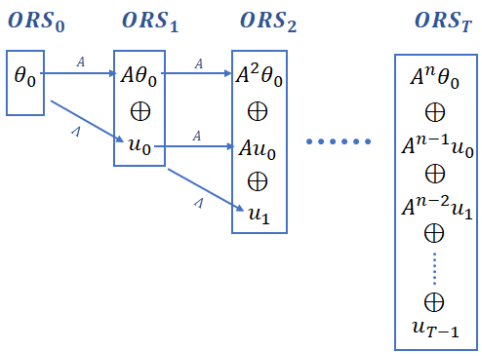


Figure 2.7: Recurrence relation to compute an overapproximate reachable set of a uncertain linear system, using Set Representation method.

Due to the increase in the number of basis vectors of the star representing the overapproximate reachable set at each step, the computation of u_k, using Theorem 12, becomes computationally expensive. Say, the star representing the overapproximate reachable set ORS_k, at time step k, has m basis vectors, where $m \gg n$: then the computation of u_k (using Theorem 12), *s.t* $\Lambda \cdot ORS_k \subseteq u_k$, will involve computing maximum and minimum value of $(\Lambda \cdot a)[i] + \Sigma_{j=1}^{m} (\Lambda \cdot G)[i][j] \cdot [c_{low}^j, c_{high}^j]$, where $j \in [\![1,m]\!]$, $i \in [\![1,n]\!]$, and Λ is an interval matrix. As the number of basis vectors m, representing the reachable set, increases at every step (increases by n at each step), the number of interval arithmetic operations needed to compute u_k also increases. Hence, reducing the number of generators would significantly improve the performance of the reachable set computation. We present the following two methods for this purpose.

Interval Arithmetic Based Reduction. In the first method, given ORS_k, we project the generators (along with the coefficients) of ORS_k on the orthonormal axis and compute n (n is the dimension of the system) axis aligned generators. That is, we compute an axis aligned hyperbox (represented as a star with n basis vectors) that overapproximates ORS_k. Given a star $\Theta_m = \langle a, G, P \rangle$ (representing a reachable set of an n dimensional system) with m basis vectors ($m \gg n$), we use Theorem 13 to compute a star Θ_n, with n basis vectors, *s.t* $\Theta_m \subseteq \Theta_n$.

Theorem 13. *Let a star be $\Theta_m = \langle a, G, P \rangle$; where G is a set of m basis vectors, and let P be of the form $c_{low}^j \leq \alpha_j \leq c_{high}^j$. Then $\Theta_m = \langle a, G, P \rangle \subseteq \Theta_n$, where $\Theta_n = \langle \bar{0}, I_n, P' \rangle$: (a) $\bar{0}$ represents the origin, (b) I_n is the collection of n orthonormal unit vectors that are axis aligned, and (c) P' is a conjunction of box aligned constraints such that $d_{low}^i \leq \alpha_i' \leq d_{high}^i$, where $[d_{low}^i, d_{high}^i]$ is the interval obtained by performing $a[i] + \Sigma_{j=1}^{m} [c_{low}^j, c_{high}^j] \cdot G[i][j]$ using interval arithmetic.*

Proof. Similar to the proof of Theorem 12. □

Intuitively, given a star Θ_m with m basis vectors, Theorem 13 computes an axis aligned hyperbox S_n, represented as a star with n basis vectors, $s.t$ $\Theta_m \subseteq \Theta_n$—this is illustrated in Fig. 2.6b. This operation is used to reduce the number of generators from m to n. Since the Minkowski sum increases the number of generators by n at each step, applying this technique at regular intervals reduces the (generator) size of the star. However, it also results in increased overapproximation of the reachable set.

Zonotope Based Reduction. The second method for reducing the number of generators uses a tool for performing operations on polytopes, called `pypolycontain` [41][1]. In [41], the authors propose a representation for polytopes that makes checking of polytope containment very efficient. As a side effect, it also provides an efficient Zonotope reduction method that uses linear programming. The details of the technique can be found in Section 2.4.2.

2.3.1.3 Case Studies

We demonstrate the applicability (and usability) of the artifacts, proposed in this section, on a variety of real life examples from robotics to medical devices. The algorithms and heuristics in a python-based prototype tool, named `CRULS`. The tool uses `numpy` [42], `scipy` [43], `mpmath` [44] for matrix multiplications, Gurobi [45] engine for visualization of the reachable sets, and `PythonRobotics` [46] for various robotics modules such as Spline Planner and Model Predictive Control (MPC). All of the experiments were performed on a Lenovo ThinkPad Mobile Workstation with i7-8750H CPU with 2.20 GHz and 32GiB memory on Ubuntu 18.04 operating system (64 bit).

Anaesthesia Delivery—A PK/PD Model. Over or under use of anaesthesia can be fatal to a patient—over-dose can cause long term detrimental effect, and under-dose might fail to maintain the state of hypnosis, causing a traumatic experience to the patient [10]. Though anaesthesia has traditionally been performed manually, with improvements in healthcare infrastructure, there is a push for automating the process; thereby raising a need to verify such safety critical systems. With the aforementioned motivation, in this case study, we study a commonly used anaesthetic drug, propofol, [10] vis-à-vis safety with perturbation in model parameters.

Model. The model considered in this case study has two components, PK and PD: the pharmacokinetics (PK) models the change in concentration of the drug as the body metabolizes it, and pharmacodynamics (PD)

[1] `https://github.com/sadraddini/pypolycontain`

models the effect of the drug on the body. The model has three compartments, and the state variables track the concentration of each compartment: (i) the first peripheral compartment c_1, (ii) the second peripheral compartment c_2, (iii) the plasma compartment c_p. The input to the model is the infusion rate of the drug (propofol). The evolution of the states are dependent on several parameters, such as: the weight of the patient (*weight*), the first order rate constants between the compartments, k_{10}, k_{12}, k_{13}, k_{21} and k_{31}. The aforementioned compartments—c_1, c_2 and c_p—comprise the pharmacokinetics (PK). The pharmacodynamics (PD), on the other hand, is tracked by the state variable c_e, which evolves relying on the parameter k_d (the rate constant between plasma and effect site) and other state variables. The detailed model can be found in Equation 5 of [10]. The system is considered safe—that is, no under/over dose occurs—if the following concentration levels are maintained: $c_1 \in [1,10]$, $c_2 \in [1,10]$, $c_p \in [1,6]$ and $c_e \in [1,8]$, where the input $u \in [0,200]$. In this case study, we study the following questions:

1. Impact on the concentration levels of the first and the second peripheral compartment, c_1 and c_2 respectively, due to small perturbations in parameters k_{21} and k_{31}.

2. Impact on the concentration level of the plasma compartment, c_p, due to small errors in the weight measurement of the patient.

3. Impact on the concentration level of the PD component, c_e, due to small perturbations in the parameter k_d. We answer questions (1) to (3) using the artifacts developed in Section 2.3.1.2.

4. Which parameter has more impact on the concentration level of the first peripheral compartment (c_1), k_{21} or k_{31}? We answer this question using Algorithm 3.

5. The amount of error in the weight measurement of the patient that can be tolerated without violating the safe concentration level of the plasma compartment c_p; we answer this question using Algorithm 4.

The answers to the above questions, (1) to (5), establish connections between the concentration levels of the various compartments and the model parameters with regards to perturbation; thus helping the practitioners/designers to be mindful of any unfortunate parameter or measurement errors that might occur during the process.

Results. The dynamics given in Equation 5 of [10] was discretized (with a step size 0.01) and the answers to questions (1)-(5) were obtained.

- *Answer to Question (1).* To answer this question, we introduced a small perturbation of $\pm 1.8\%$ in the parameters k_{21} and k_{31}. We observed that starting with an initial concentration level of $c_p \in [2,4]$, $c_1 \in [4,6]$, $c_2 \in [4,6]$, $c_e \in [3,5]$ and $u \in [0,10]$, the system reaches an unsafe concentration level in the first peripheral compartment c_1 in 20 time steps, whereas the unperturbed system (*i.e.*, with no perturbation) remains safe. The reachable sets, up-to 20 time steps, have been shown in Fig. 2.8: The x and the y axis represents the concentration levels c_1 and c_2 respectively. Green represents reachable sets of the unperturbed system, and blue represents the reachable set of the perturbed system. This analysis, using the reachable set computation method proposed in Section 2.3.1.2, took 0.32 seconds.

- *Answer to Question (2).* We introduced a small perturbation of $\pm 0.8\%$ in the weight measurement of the patient, and observed that starting with an initial concentration level of $c_p \in [2,4]$, $c_1 \in [3,6]$, $c_2 \in [3,6]$, $c_e \in [2,4]$ and $u \in [2,10]$, the system reaches an unsafe concentration level in the plasma compartment c_p in 20 time steps, whereas the unperturbed system remains safe. The reachable sets, up-to 20 time steps, have been shown in Fig. 2.9a: The x and the y axis represents the time steps and the concentration level c_p respectively. Cyan represents reachable sets of the unperturbed system, and blue represents the reachable set of the perturbed system. This analysis, using the reachable set computation method proposed in Section 2.3.1.2, took 0.32 seconds.

- *Answer to Question (3).* We introduced a perturbation of $\pm 5\%$ in the parameter k_d, and observed that starting with the same initial concentration level as before, the system reaches an unsafe concentration level in the PK component c_e in 20 time steps, whereas the unperturbed system remains safe. The reachable sets, up-to 20 time steps, have been shown in Fig. 2.9b: The x and the y axis represents the time steps and the concentration level c_p respectively. Cyan represents reachable sets of the unperturbed system, and blue represents the reachable set of the perturbed system. This analysis, using the reachable set computation method proposed in Section 2.3.1.2, took 0.32 seconds.

- *Answer to Question (4).* Since we are interested in the impact of k_{31} and k_{21} on concentration level of only the first peripheral compartment (c_1), we projected our model to the c_1 dimension for this analysis. After the projection, we applied Algorithm 3 and found that the parameter k_{31} has more impact than k_{21}: k_{31} is more sensitive to perturbation than k_{21} with regards to concentration level c_1. With an initial concentration level $c_p \in [2,4]$, $c_1 \in [4,6]$, $c_2 \in [4,6]$, $c_e \in [3,5]$ and $u \in [0,10]$, we separately studied the impact of $\pm 2\%$ on k_{31} and k_{21}—perturbation in k_{31} has more impact on the reachable set than

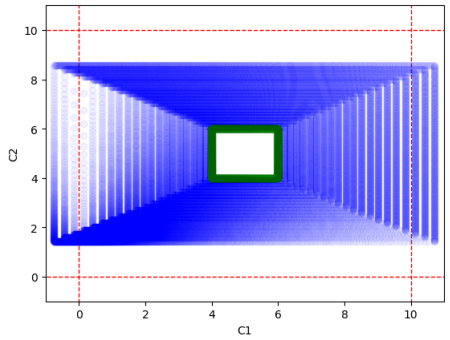
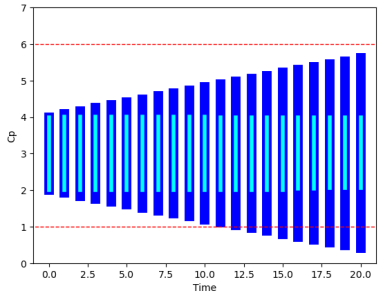


Figure 2.8: (Anaesthesia: Concentration level of c_1 and c_2 with perturbation in k_{21} and k_{31}) The x and the y axis represents the concentration levels c_1 and c_2 respectively. Green: Phase plots of the reachable sets of the unperturbed system. Blue: Reachable sets of the perturbed system. Red: Lines demarcating the safe region. That is, c_1 is considered safe if it is between 0 to 10. The safety of c_2 is interpreted similarly.

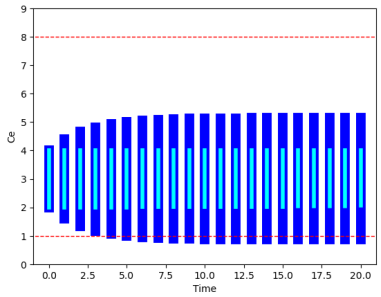
perturbation in k_{21}. We illustrate this in Fig. 2.9c: The x and the y axis represents the time steps and the concentration level of c_1 respectively. Cyan represents reachable sets with $\pm 2\%$ in k_{21}, and blue represents the reachable set with $\pm 2\%$ in k_{31}. This analysis, *i.e.*, applying Algorithm 3 took just 0.024 seconds.

- *Answer to Question (5).* With an initial concentration level of $c_p \in [2,4]$, $c_1 \in [3,6]$, $c_2 \in [3,6]$, $c_e \in [2,4]$ and $u \in [2,10]$, we applied Algorithm 4 to find out the amount of perturbation that can be introduced to the weight measurement of the patient without violating the safe concentration level of the plasma compartment c_p. In just 4.32 seconds, we found that an error of $\pm 0.4\%$ can be tolerated in the weight measurement without violating safety up-to 20 time steps. The reachable sets, up-to 20 time steps and $\pm 0.4\%$ perturbation in weight, have been shown in Fig. Fig. 2.9d: The x and the y axis represents the time steps and the concentration level c_p respectively. Cyan represents reachable sets of the unperturbed system, and blue represents the reachable set of the perturbed system.

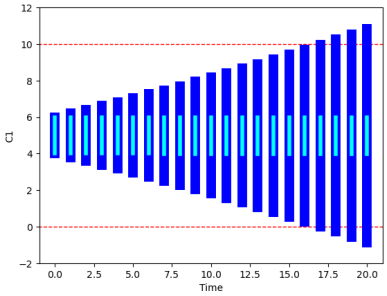
Mars Rover. In this case study, similar to the previous one, we illustrate various artifacts proposed in this paper on a Mars Rover simulated on real Martian Terrain. In the robotics community, reachable set computation with model uncertainties has been previously considered to compute safety of a Mars Rover [47]. Here, in this case study, we not just limit our attention to determine safety of a Mars Rover's path, but also find out the parameters that are more sensitive to perturbations and also compute the amount of perturbation the rover can tolerate without violating safety—to the best of the authors' knowledge, such questions have not been addressed before. In this case study, a Mars Rover is required to determine the safety of its computed path, in a real Martian Terrain, where the model of the rover has uncertainties in its parameters like velocity,



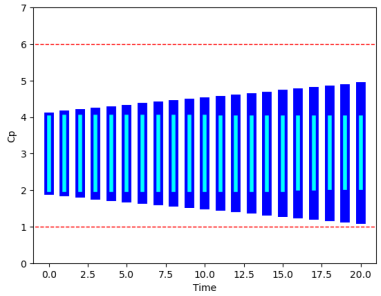
(a) (Anaesthesia: Concentration level of c_p with perturbation in the weight measurement of the patient) The x and the y axis represents time steps and the concentration level c_p respectively. Cyan: Phase plots of the reachable sets of the unperturbed system. Blue: Reachable sets of the perturbed system.



(b) (Anaesthesia: Concentration Level of c_e with perturbation in k_d) The x and the y axis represents time steps and the concentration level c_e respectively. Cyan: Phase plots of the reachable sets of the unperturbed system. Blue: Reachable sets of the perturbed system.



(c) (Anaesthesia: Comparing concentration level of c_1 with perturbation in k_{31} and k_{21}) The x and the y axis represents time steps and the concentration level c_1 respectively. Cyan: Phase plots of the reachable sets with perturbation of $\pm 2\%$ in k_{21}. Blue: Reachable sets with the same amount of perturbation in k_{31}.



(d) (Anaesthesia: Concentration level of c_p with safe amount of perturbation in the weight measurement of the patient) The x and the y axis represents time steps and the concentration level c_p respectively. Cyan: Phase plots of the reachable sets of the unperturbed system. Blue: Reachable sets with safe amount of perturbation in the weight measurement of the patient.

Figure 2.9: (Reachable Sets of Anaesthesia Delivery Model) Red: Lines demarcating the safe region. For example, in Fig. 2.9a, c_p is considered safe if it is between 1 to 6.

yaw angle, etc. We perform the safety analysis of a given path of the Mars Rover using the reachable set computation method proposed in Section 2.3.1.2, with uncertainties in its model. We simulate the above mentioned scenario using NASA's HiRise Dataset [2, 3], where the rover is required to safely maneuver in the Martian Terrain avoiding the steep obstacles. The rover is assumed to follow a linearized bicycle model with perturbations in the dynamics matrix. Similar to [47], the path of the Mars Rover is computed as follows: given a set of waypoints (including the start and end coordinates) in the Martian Terrain, we compute a reference trajectory using Cubic Spline Planner [46], which is then used to plan the Mars Rover's control inputs using Model Predictive Control (MPC) [46].

Model of the Rover. We assume a linearized bicycle model [46] for our rover dynamics, with state variables: (i) x-position x, (ii) y-position y, (iii) velocity v, (iv) yaw angle ϕ. The input to the rover dynamics are: (i) acceleration a, (ii) steering angle δ. The states of the rover dynamics evolve by relating itself to other state variables through complicated trigonometric functions of v, ϕ and δ. The detailed model can be found in [46]. Note that, since the given model is non-linear (it has trigonometric terms), we use the MPC path to assign value to the trigonometric terms and obtain a linear dynamics at each time step. Therefore, for this case study, we solve reachability at each time step.

Here, we not just limit our attention to compute reachable sets of the Mars Rover with model uncertainties, *but also use the artifacts, developed in this paper, to identify (rather) simpler trigonometric terms—that are sensitive to perturbation—relating it (the given state variable) to other state variables*. Specifically, we answer the following questions:

1. Impact of perturbation, in the yaw angle ϕ, on the Mars Rover's path.
2. Identify simpler trigonometric terms—that relate evolution of the x and y dimensions to other state variables, based on their sensitivity to perturbation.
3. Amount of perturbation that can be tolerated in the yaw angle ϕ without violating the safety of the Mars Rover.

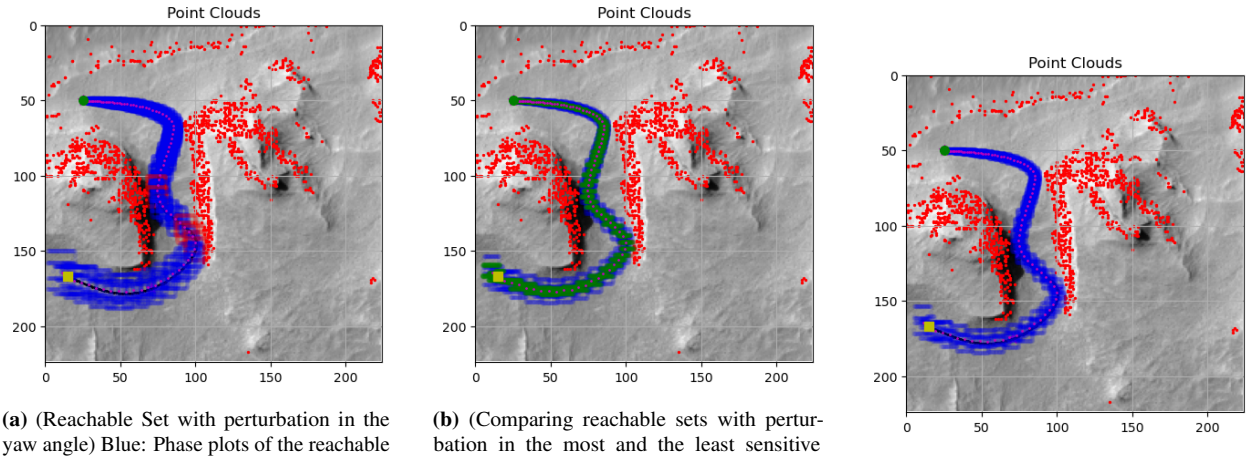
The answers to the above questions will help an engineer/practitioner to carefully tune the error tolerances of the various sensors available. This is particularly useful in scenarios, such as this, where there is a limited resource available and therefore resources should be allocated very judiciously to various sensors and controllers.

Results. In the following experiments, to maintain consistency for comparing the results, we have used the same waypoints and therefore yielding the same MPC path. In the following figures, the red dots represent high raised obstacles (or unsuitable temperature) in the Martian Terrain.

- *Answer to Question (1).* We compute the reachable sets, with $\pm 15\%$ perturbation in yaw angle (ϕ), using the method proposed in Section 2.3.1.2. In Fig. 2.10a we show the computed reachable sets: Blue represents the computed reachable sets with $\pm 15\%$ perturbation in yaw angle (ϕ); the same reachable set is highlighted in red when it intersects with an unsafe point in the Martian Terrain. This analysis just took 2.30 seconds.

- *Answer to Question (2).* Since we are interested in identifying simpler terms that affect the evolution of just the x and y dimensions, in relation to perturbation, we projected our model to the x and y dimensions for this analysis. After the projection, we applied Algorithm 3 to find the simpler trigonometric terms (*i.e.* elements of individual cells) that are most and least sensitive to perturbation with respect to $x-y$ dimension. The most sensitive terms for x dimension are: $v \cdot sin(\phi) \cdot \phi$, and $cos(\phi)$ relating x to the state variable v. The most sensitive terms for y dimension are: $v \cdot cos(\phi)$ relating y to state variable ϕ, and $sin(\phi)$ relating y to the state variable v. Similarly, the least sensitive terms for x dimension are: $-v \cdot sin(\phi)$ relating x to ϕ, and $cos(\phi)$ relating x to v. The least sensitive term for y dimension is: $-v \cdot cos(\phi) \cdot \phi$. By introducing a perturbation of $\pm 10\%$ to the most and the least sensitive terms, we compute the reachable sets separately, illustrating the impact of the top versus the bottom cells (as in this case terms), as returned by Algorithm 3, on the reachable set. This is illustrated in Fig. 2.10b: Blue represents reachable set with perturbation in the most sensitive cells, and green represents reachable set with perturbation in the least sensitive cells. Identifying the most and the least sensitive terms, using Algorithm 3, took 0.04 seconds.

- *Answer to Question (3).* We applied Algorithm 4 to find out the amount of perturbation that can be introduced to the yaw angle without violating the safety. In 2675.39 seconds, we found that an error of $\pm 9\%$ can be introduced to the yaw angle without violating safety. The reachable sets with $\pm 9\%$ perturbation in the yaw angle have been shown in Fig. Fig. 2.10c: Blue represents the computed reachable sets with $\pm 9\%$ perturbation in ϕ.



(a) (Reachable Set with perturbation in the yaw angle) Blue: Phase plots of the reachable sets with perturbation in the yaw angle. Red: The same reachable sets are highlighted in red when it intersects with an unsafe point in the Martian Terrain.

(b) (Comparing reachable sets with perturbation in the most and the least sensitive terms) Blue: Phase plots of the reachable sets with perturbation in the most sensitive terms. Green: Reachable sets with the same amount of perturbation in the least sensitive terms.

(c) (Reachable Set with tolerable perturbation in the yaw angle) Blue: Phase plots of the reachable sets with tolerable perturbation in the yaw angle.

Figure 2.10: (Mars Rover Reachable Sets) Red Dots: Unsafe points in the Martian Terrain representing high raised obstacle or unsuitable temperature.

2.3.2 Verifying Complex Autonomous Systems: Statistical Approaches

As discussed earlier, model uncertainty has a complex nonlinear impact on trajectories. The existing methods for rigorous safety analysis face limitations in scalability, requiring either conservative overapproximations or time-consuming computations to determine reachable sets accurately. To address these challenges, this section proposes computing reachable set artifacts that offer statistical guarantees. Computing statistically correct reachable sets presents several advantages over statistical verification. Firstly, the reachable set artifacts provide a comprehensive understanding of the system's possible states in the presence of model uncertainties, offering better intuition for system designers. Secondly, these artifacts can be reused for evaluating safety across different specifications, unlike statistical verification that necessitates repeated safety analyses when modifying specifications. Thirdly, the artifacts enable a trade-off between computational cost and desired statistical guarantees, empowering users to compute reachable sets with varying levels of confidence based on specific application scenarios. Lastly, the proposed approach eliminates the *"wrapping effect"* ensuring that the overapproximation at each time step is independent of previous representations.

It is further worth noting that these reachable set computation techniques have also been used in monitoring autonomous systems [27], and in robotics to perform safe navigation with least computation cost [47]. In the context of monitoring, these techniques are utilized when provided with a noisy system log that may have missing samples at certain time steps, along with a bounding model of the system (such as a

uncertain linear system). By leveraging reachable set computation techniques, the behavior of the system for the missing samples can be overapproximated. Subsequently, monitoring algorithms examine the intersection of the reachable set with unsafe specifications to ensure safety. During monitoring tasks, the ability to compute tight reachable sets rapidly becomes crucial, and statistical techniques can be particularly advantageous in such scenarios. Secondly, these techniques have also been used in safe navigation of simulated Mars rover to maneuver through a real-Martian terrain. In this context, the rover follows a predetermined path, and considering the criticality of each maneuver, it computes reachable sets with varying degrees of confidence throughout its navigation. This approach allows the rover to verify the safety of its path while minimizing the cumulative computation time. As a result, the navigation process remains safe while ensuring efficient resource utilization. Therefore, the technique proposed in this paper for computing reachable sets holds great potential in verifying and designing complex autonomous systems, offering valuable insights and practical solutions to enhance their safety and performance.

In this section, we introduce two techniques for computing reachable sets. The first technique applies sequential hypothesis testing, commonly used in statistical verification, to compute reachable sets using a counterexample guided refinement framework. The second technique utilizes model learning with probably approximately correct guarantees. By solving an optimization problem, the technique learns a model that approximates the reachable set of uncertain linear systems, ensuring statistical guarantees. A python prototype tool was implemented to showcase the applicability of the approaches on standard benchmarks. Comparisons with an existing techniques, `Flow*`, was also conducted to demonstrate the trade-off between computational effort and statistical guarantees. To the best of authors' knowledge, the standard statistical verification techniques and model learning techniques have not been applied to compute statistically approximate reachable set for uncertain linear systems.

2.3.2.1 Reachable Sets With Probabilistic Guarantees using Hypothesis Testing

This section introduces a technique to compute reachable sets, using a counterexample guided refinement framework, with a statistical verifier using hypothesis testing. This approach consists of three sub-routines: Generator, Verifier, and Refiner. The Generator sub-routine computes a candidate reachable set by performing operations on the reachable sets of sampled dynamics from Λ. The Verifier sub-routine employs statistical verification techniques, specifically the *Jeffries Bayes Factor* test, to confirm that the candidate set generated

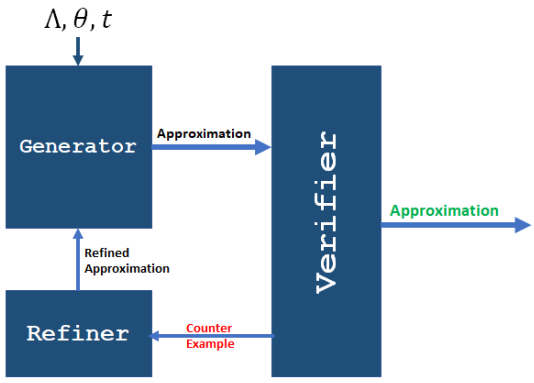


Figure 2.11: Workflow for computing probably approximately correct reachable set for uncertain linear systems.

by the Generator highly likely contains reachable sets of dynamics from Λ. The Refiner sub-routine is invoked only if the Verifier determines that the candidate set does not pass the test. In such cases, the Refiner reuses the artifacts generated during the verification phase and calls Generator with additional parameters to generate a different candidate set. The workflow is illustrated in Figure 2.11.

Generator: Computing Candidate Reachable Set. Here, we present several heuristics for computing the candidate for provably approximate reachable set. Given that the validation of a candidate reachable set relies on a statistical hypothesis testing framework, any heuristic can be employed to generate a candidate reachable set. A common approach involves computing reachable sets of dynamics sampled from Λ using various methods. We present four well-informed heuristics for computing a candidate reachable set. Among these heuristics, one in particular, that generates the candidate based on *structure of uncertainties* is a **conservative overapproximation** of the reachable set of uncertain systems. For each proposed heuristic, we offer a fundamental rationale for its inclusion and provide theoretical guarantees if applicable.

Bloating the Reachable Set of Mean Dynamics. In this particular heuristic, our initial step involves computing the reachable set of the mean dynamical system (obtained by substituting all the intervals with the corresponding mid-points). Subsequently, we determine the Hausdorff distance between the mean reachable set and the reachable set obtained from a random sample of N dynamics originating from $\dot{x} = \Lambda x$. Next, we bloat the reachable set of the mean dynamics by incorporating the computed Hausdorff distance along with an additional predetermined padding factor ε. The detailed steps for this heuristic can be found in Appendix 2.4.3. The resulting reachable set produced by this process is labeled as $\mathsf{bloatMean}(\Lambda, \theta, t)$.

Convex Overapproximation Using Oriented Rectangular Hulls. Instead of simply bloating the reachable set of the mean dynamics of the system from random samples, a more natural approach is to generate a set that

tightly encloses the randomly generated sets. After computing the reachable sets from the generated samples, we employ the concept of *Oriented Rectangular Hulls* to aggregate these reachable sets. Oriented rectangular hulls, as proposed in [48], are parallelotopes where the *directions* are selected through principal component analysis (PCA) of various sample trajectories. For a linear uncertain system $\dot{x} = \Lambda x$, an initial set θ, and a given time t, the candidate reachable set produced using this heuristic is denoted as $\texttt{encloseORH}(\Lambda, \theta, t)$. One specific instance of computing the candidate reachable set would be to use the orthonormal directions for the template directions, instead of performing PCA on sample trajectories. For such template directions, there is no need to invoke an optimization problem as required for computing the *Oriented Rectangular Hull*. Instead, an axis-aligned overapproximation can be computed using interval arithmetic. Furthermore, performing a convex hull operation on such axis-aligned sets is computationally efficient (as it can be done by computing a box hull of the sets). We represent such axis aligned reachable sets as $\texttt{boxBloat}(\Lambda, \theta, t)$.

Structure Guided Reachable Set Computation. In [17], a set of sufficient conditions (referred to as linearity conditions) is proposed to exploit the structure of the dynamics. When these conditions are satisfied, the uncertainties in the system do not exhibit non-linear growth over time but instead preserve linearity at all time steps. Consequently, if these conditions are met, verification can be performed efficiently and precisely, ensuring both soundness and completeness. However, it is important to note that our paper focuses on uncertainties that go beyond the scope of the aforementioned work [17]. In this particular heuristic, we separate the uncertainties in the dynamics into two parts: one that satisfies the linearity conditions and another that does not. For the part of the dynamics that satisfies the linearity conditions, we propose a deterministic technique to compute a provable overapproximation of the reachable set associated with that portion of the dynamics. The details and proof of this method can be found in Appendix 2.4.4. For the remaining part of the dynamics that does not satisfy the linearity conditions, we compute the reachable set using heuristics based on random sampling of the dynamics. The specific mathematical treatments and further information about this heuristic can be found in Appendix 2.4.4.1. Given a linear uncertain dynamics $\dot{x} = \Lambda x$, an initial set θ and time t, we denote the reachable set computed using this heuristic as $\texttt{uniformORH}(\Lambda, \theta, t)$.

Sampling Based on Singular Values. Expanding on the heuristic involving the use of *Mean Dynamics*, it is not immediately clear why the reachable set of the mean dynamics is chosen and subsequently bloated. Therefore, we question whether there exists a more suitable sample reachable set that can be appropriately bloated to compute the candidate reachable set. Since computing the reachable set involves evaluating the matrix exponential, a more appropriate choice would be a sample with higher magnitude of e^{At}. This leads

us to consider matrices with higher singular values, as we know that the singular values directly influence the magnitude of the matrix exponential. For our analysis, we focus on uncertain systems represented as interval matrices. In [20], given an interval matrix Λ, the *norm-2* of Λ, denoted as $||\Lambda||_2$, is defined as follows: $||\Lambda||_2 = \texttt{sup}\{\ ||A||_2 \mid A\ \in \Lambda\}$. Let $A_{\max} \in \Lambda$ be an element in Λ with the maximum singular value, *i.e.*, $\forall_{A\in\Lambda}\sigma_{\max}(A_{\max}) \geq \sigma_{\max}(A)$, where $\sigma_{\max}(A_{\max}) = ||\Lambda||_2$. In [20], Theorem 7 provides an algorithm to compute $||\Lambda||_2$. We utilize this result to determine the matrix with the maximum singular value, denoted as *A*max. The steps for computing the candidate reachable set using the matrix with the maximum singular value are similar to those of $\texttt{bloatMean}(\Lambda,\theta,t)$, with the distinction that A_c is replaced by $A_{\max}$. Further details can be found in the Appendix, specifically in Section 2.4.5. For a linear uncertain dynamics $\dot{x} = \Lambda x$, an initial set θ, and a given time t, we refer to the candidate reachable set computed using this approach as $\texttt{maxSV}(\Lambda,\theta,t)$.

2.3.2.2 Statistical Verification of Candidate Reachable Sets

In this subsection, we introduce the statistical hypothesis testing framework utilized to demonstrate that the candidate reachable set is a conservative estimate of the reachable set of the linear uncertain system with a high level of confidence. Let $Prob(\Lambda,\theta,t,\mathcal{S})$ represent the probability that the reachable set of a randomly selected sample dynamics (denoted as $A \in \Lambda$, $\texttt{RS}(A,\theta,t)$) is contained within the set $\mathcal{S}$. To be more specific, given a candidate reachable set $\mathcal{S}$ and a confidence threshold c (say, 0.99), we employ the *Jeffries Bayes Factor* test [49] to determine if $Prob(\Lambda,\theta,t,\texttt{RS}) \geq c$. We choose the Bayes Factor test for two main reasons. Firstly, this verification process involves computing the reachable set for sample dynamics generated from Λ, which can be performed very efficiently. Secondly, the number of samples that need to be verified using this technique is independent of the number of uncertainties in Λ or the size of the domain of uncertainties D_Λ. This is in contrast to other methods of computing reachable sets, where the computation time depends on the number of uncertainties and the size of the domain D_Λ. The algorithm we propose in this method will be the `Verifier` subroutine in the overall workflow.

In *Statistical Hypothesis Testing*, one is required to form two hypotheses — the null hypothesis H_0 and the alternate hypothesis H_1. After the hypotheses are formalized, evidence is gathered by randomly sampling (finite number of samples) the sample space — once sufficient evidence has been gathered to support either of the hypotheses *i.e.* H_0 or H_1, the algorithm terminates by *accepting* the hypothesis, which has been

supported by the observed random samples and *rejecting* the other one. The criterion for *accepting/rejecting* a hypothesis is based on the method one chooses. In this work, we choose a *Bayesian approach — Jeffries Bayes Factor* test as in [49], as it is suitable and yet simple for our purpose. The probability of wrongly *accepting* H_1, when H_0 is the actual truth, is known as the *type I error*. Informally, if H_1 states that the given *approximation property* is true (H_0 states otherwise) — for our answer to be credible, the value of *type I error* has to be sufficiently low. Given a confidence value $c \in [0,1]$, our *null hypothesis* H_0 and *alternate hypothesis* H_1 are as follows: $H_0 : Prob(\Lambda, \theta, t, \mathtt{RS}) < c$ and $H_1 : Prob(\Lambda, \theta, t, \mathtt{RS}) \geq c$. Next, we proceed to describe the methodology we employ to determine whether to accept H_0 or H_1. Let, $A_1, A_2, \cdots, A_K$ be a set of K *sample dynamics* of Λ, *s.t.* $\underset{1 \leq j \leq K}{\forall} A_j \in \Lambda \wedge \mathtt{RS}_\theta(A_i, t) \subseteq \mathtt{RS}$. The probability of this event occurring under the assumption that the null hypothesis H_0 is true can be expressed as:

$$\mathbb{P}\big[\bigwedge_{j=1}^{K} \mathtt{RS}_\theta(A_i, t) \subseteq \mathtt{RS} \mid H_0\big] = \int_0^c q^K \, dq$$

Similarly, the probability of $A_1, A_2, \cdots, A_K$ satisfies $\mathtt{RS}_\theta(A_i, t) \subseteq \mathtt{RS}$, given H_1 is true is:

$$\mathbb{P}\big[\bigwedge_{j=1}^{K} \mathtt{RS}_\theta(A_i, t) \subseteq \mathtt{RS} \mid H_1\big] = \int_c^1 q^K \, dq$$

Given the above two probabilities, the *Bayes Factor* is given by the ratio of the above two probabilities:

$$\frac{\mathbb{P}\big[\bigwedge_{j=1}^{K} \mathtt{RS}_\theta(A_i, t) \subseteq \mathtt{RS} \mid H_1\big]}{\mathbb{P}\big[\bigwedge_{j=1}^{K} \mathtt{RS}_\theta(A_i, t) \subseteq \mathtt{RS} \mid H_0\big]} = \frac{1 - c^{K+1}}{c^{K+1}} > B \tag{2.29}$$

Intuitively, the *Bayes Factor* is the strength of evidence favoring one hypothesis over another. A sufficiently high value of B indicates that the evidence favors H_1 over H_0. Given B, we can compute K from Equation 4.12 (as given in [49])

$$K \quad > \quad -\frac{log(B+1)}{log(c)} \tag{2.30}$$

Given B and c, [49] gives the formula for calculating *type I error* (falsely accepting H_1, when H_0 is the actual truth), denoted as $err(B,c)$ as follows:

$$err(B,c) = \frac{c}{c+(1-c)B} \tag{2.31}$$

Note that, $err(B,c)$ is therefore the probability that our answer is wrong — we infer $Prob(\Lambda,\theta,t,\texttt{RS}) \geq c$, when in reality $Prob(\Lambda,\theta,t,\texttt{RS}) < c$. The algorithm, as used by the `Verifier` module to *accept/reject* H_0 or H_1 is therefore given as follows:

1. Given (sufficiently high) values of B and c, compute K using Section 2.3.2.2.
2. Let, $S = \{A_1, A_2, \cdots, A_K\}$ be a set of K random *sample dynamics* of Λ generated according to the probability density function μ.
3. If all the K samples in S satisfies: $\texttt{RS}_\theta(A_i,t) \subseteq \texttt{RS}$; we accept H_1.
4. Otherwise, we *reject* H_1 and accept H_0 with a *counterexample* $\mathscr{C}$. Where $\mathscr{C} = \texttt{RS}_\theta(A',t)$, such that $\texttt{RS}_\theta(A',t) \nsubseteq \texttt{RS}$.

In the above mentioned algorithm (steps 1-4), the probability of falsely accepting H_1, when H_0 is the real truth (*type I error*) is given by $err(B,c)$.

2.3.2.3 Main Algorithm

In this subsection, we integrate the sub-routines `Generator`, `Verifier`, and `Refiner`. The algorithm takes three inputs: (i) an uncertain dynamics Λ, (ii) an initial set θ, and (iii) a time value t. Following are the details of our algorithm:

1. `Generator`: When provided with the uncertain dynamics Λ, initial set θ, and time t as input, the generator produces a candidate reachable set using one of the four heuristics outlined in Section 2.3.2.1.
2. `Verifier`: When given the input the property of interest (whether the reachable set is contained in the candidate reachable set or is from bounded distance from it), the confidence value (generally $c = 0.99$), and the threshold of type I error, the `Verifier` either accepts the candidate reachable set or rejects it by producing an instance of the dynamics A' that does not satisfy the property of interest (that is, the

reachable set is not contained in the candidate set or not within the bounded distance from candidate set).

3. `Refiner`: Given the candidate reachable set RS rejected by the `Verifier` and the instance of reachable set that violates the property, the `Refiner` increases the value of ε used in the bloating of each set such that $\varepsilon > \texttt{dis}(\texttt{RS}, \texttt{RS}_\theta(A', t))$. This new value of ε is given as input to the generator in recomputing the candidate reachable set.

Given the three modules `Generator`, `Verifier` and `Refiner`, the main algorithm, an ensemble of the three modules is given in Algorithm 5

Algorithm 5: Reachable set computation

input : Uncertain dynamics Λ, an initial set θ, time t, $\varepsilon > 0$.

output : Reachable set $\mathscr{A}$.

```
RS := Generator(Λ, θ, t, ε) ;
(res, 𝒞) := Verifier(RS, c, Property) ;
if res = accept then
    return RS ;
while True do
    ε := Refine(RS, 𝒞) ;
    RS := Generator(Λ, θ, t, ε) ;
    (res, 𝒞) := Verifier(RS, c, Property) ;
    if (res = accept) then
        return RS ;
```

2.3.2.4 Reachable Sets Using Model Learning

In this section, we present a Model Learning based approach for computing the reachable set of a linear uncertain system. Here, we aim to learn a probably approximately correct model of $e^{\Lambda t}$. To achieve efficient model learning and reachable set computation, we approximate $e^{\Lambda t}$ using a linear function represented by

$w(x,\gamma,t)$, which follows the following template:

$$w(x,\gamma,t) = e^{A_c t}x + \underbrace{\begin{bmatrix} c_{1,1} & \cdots & c_{1,p} \\ \vdots & \ddots & \vdots \\ c_{n,1} & \cdots & c_{n,p} \end{bmatrix}}_{C} \gamma; \tag{2.32}$$

where $x \in \theta$, $\gamma \in D_\Lambda$, $t \in \mathbb{R}_{\geq 0}$, θ is the initial set, D_Λ is the set values of the uncertain variables and $A_c \in \Lambda$ is the *mean dynamics*. Once such a model is learnt —given θ represented as a hyper-rectangle, D_Λ represented as an interval set and a time step t—the reachable set can be computed very efficiently using interval arithmetic by plugging in the values in Eq. (2.32). Once the model is learned (*i.e.*, the parameters $c_{i,j}$ are learnt) the reachable set can be computed efficiently using interval arithmetic. This can be achieved by substituting the values into Eq. (2.32), considering the given θ represented as a hyper-rectangle, D_Λ represented as an interval set, and the chosen time step t. In Section 2.4.6.1, we present a concise overview of *Scenario Optimization* [50] (as detailed in [51]). Next, we outline the necessary steps for model learning, specifically the process of learning the parameters $C[i,j]$. In the last step, we propose our algorithm for computing reachable sets based on model learning.

1. **Sampling.** Given a set $\theta \subset \mathbb{R}^n$, the set of values of uncertain variables in Λ represented as an interval set $D_\Lambda \subseteq \mathbb{R}^p$ (where p is the number of uncertainties in Λ), time interval $t_\delta \subseteq [0,T]$, we define the following randomly chosen samples according to their corresponding probability distributions. (i) Let, $\{x_1, x_2, \cdots, x_M\}$ be M random samples of θ, *i.e.*, $x_i \in \theta$. We denote the set $\{x_1, x_2, \cdots, x_M\}$ as $\{x_i\}^M$. (ii) Let, $\{\gamma_1, \gamma_2, \cdots, \gamma_N\}$ be N random samples of D_Λ, *i.e.* $\gamma_j \in D_\Lambda$. We denote the set $\{\gamma_1, \gamma_2, \cdots, \gamma_N\}$ as $\{\gamma_j\}^N$. (iii) Let, $\{t_1, t_2, \cdots, t_O\}$ be O random samples of t_δ, *i.e.* $t_k \in t_\delta$. We denote the set $\{t_1, t_2, \cdots, t_O\}$ as $\{t_k\}^O$. Given a $\gamma_j \in D_\Lambda$, let $A_{\gamma_j} \in \Lambda$ be the matrix obtained by assigning values γ_j to the variables of Λ. We denote the reachable point for a given sample point (x_i, γ_j, t_k) as $y_{i,j,k}$, *i.e.*, $y_{i,j,k} = \texttt{RS}(x_i, A_{\gamma_j}, t_k)$. Let, $\{y_{i,j,k}\}^{M \cdot N \cdot O} = \left\{ y_{i,j,k} = \texttt{RS}(x_i, A_{\gamma_j}, t_k) \mid x_i \in \{x_i\}^M \wedge \gamma_j \in \{\gamma_j\}^N \wedge t_k \in \{t_k\}^O \right\}$. Let, the shorthand of the above sampling method be denoted as $\texttt{sample}(\theta, D_\Lambda, t_\delta) = \left(\{x_i\}^M, \{\gamma_j\}^N, \{t_k\}^O, \{y_{i,j,k}\}^{M \cdot N \cdot O}\right)$.

2. **Learning the Model.** We formulate the problem of learning the parameters C in Equation 2.32 as a *Scenario Optimization* problem. Consequently, the guarantees obtained will be statistical in nature. Consequently, the guarantees we obtain will have a statistical nature. We start with an initial set θ,

a collection of uncertainties D_Λ, and a time interval t_δ. We represent the samples derived from these inputs as $\texttt{sample}(\theta, D_\Lambda, t_\delta)$. Using these samples, we proceed to learn the parameters C (as shown in Equation 2.32) by formulating it as an optimization problem. Additional details can be found in Appendix 2.4.6.2. We denote the solution to this optimization problem as (C^*, κ^*), where $C^*[i,j] = c^*_{i,j}$ and κ^* denotes the *deviation* of the learnt model from the true model. Let, the shorthand of this learning module be denoted as $\texttt{learnModel}(\theta, D_\Lambda, t_\delta) = (C^*, \kappa^*)$.

3. **Computing Reachable Sets from the Learnt Model.** Once a probably approximately correct model of $e^{\Lambda t}$ is learned, we compute the reachable set by *bloating* the reachable set obtained from the model by κ^*. Let the reachable set from the learned model be θ_t. From the optimization problem in Eq. (2.37), we get a statistical guarantee that the set θ_t is κ^* close to $\texttt{RS}(\theta, \Lambda, t)$. So to compute an overapproximation of $\texttt{RS}(\theta, \Lambda, t)$, we bloat θ_t by κ^*. Consequently, $\texttt{RS}(\theta, \Lambda, t) \subseteq \mathit{Bloat}(\theta_t, \kappa^*)$, with statistical guarantee as in Theorem 16. We formalize this approach in Algorithm 6, and refer to this approach as modelLearn.

2.3.2.5 Evaluation

Algorithm 5 and 6 was implemented in a `Python` based tool. We have used `Gurobi`[2] [45] for visualizing the reachable sets. For computing distances and to perform subset checking, we have used the tool `pypolycontain`[3]. Additionally, we use `numpy`[4] [42], `scipy`[5] [43] and `mpmath`[6] for several other functionalities. If the paper gets accepted, we will open source our tool. In the following subsections, we will evaluate our Algorithm 5 and 6 on several benchmarks, and show that our technique is indeed scalable, even for high dimensional benchmarks. Most of the benchmarks are taken from ARCH workshop[7]. All the experiments were performed on a Lenovo ThinkPad Mobile Workstation with i7-8750H CPU with 2.20 GHz and 32GiB memory on Ubuntu 18.04 operating system (64 bit). We assume that all the model uncertainties of the system are uniformly distributed and use uniform distribution for statistical validation and model learning. We report

[2] `https://www.gurobi.com/`

[3] `https://github.com/sadraddini/pypolycontain`

[4] `https://numpy.org/`

[5] `https://www.scipy.org/`

[6] `http://mpmath.org/`

[7] `https://cps-vo.org/group/ARCH/benchmarks`

		Hypothesis Testing						
Benchmark	**Dim**	**Mean**	**SV**	**ORH**	**Uniform**	**Box**	**Learn**	**Flow***
Adaptive Cruise Control	4	9.28	14.15	11.16	11.58	**1.57**	3.97	41.35
Flight Collision	4	12.35	7.28	10.36	7.59	**2.045**	4.077	922.48
Anaesthesia	5	7.39	12.44	31.53	17.89	**1.78**	8.13	24.73
Motor-Transmission	5	17.95	12.73	13.02	12.78	**1.72**	9.18	19.51
Spacecraft Rendezvous	6	25.18	19.1	19.477	18.98	**3.05**	24.57	2176.54
2-wheeled Robot	10	112.68	-	119.52	109.88	**7.604**	90.51	50.48
5 Vehicle Platoon	15	250.1	-	231.41	214.38	**11.455**	147.9	1745.47
Quadrotor	16	330.1	-	392.37	350.84	**5.74**	152.73	39.62

Table 2.1: Evaluation on Benchmarks. **Dim**: Number of dimensions of the benchmark; **Hypothesis Testing**: Time taken (in seconds) by various hypothesis testing based methods (The sub-columns are as follows. **Mean**: bloatMean, **SV**: maxSV, **ORH**: encloseORH, **Uniform**: uniformORH, **Box**: boxBloat.); **Learn**: Time taken (in seconds) by modelLearn method; **Flow***: Time taken (in seconds) by `Flow*`.

the following: (i) performance of Algorithm 5 using all the heuristics, bloatMean, maxSV, encloseORH, uniformORH and boxBloat; (ii) performance of Algorithm 6; (iii) finally, we compare the performance of Algorithm 5 and 6 mutually and with Flow*. A summary of the evaluations on all the benchmarks is given in Table 2.1.

Flight Collision. [52] provides an aircraft collision avoidance maneuver model. In air traffic control, collision avoidance maneuvers are used to resolve conflicting paths that arise during free flight. Given the safety critical nature of the system, and also the environmental uncertainties associated with the system dynamics, this benchmark is ideally suited for our analysis. The model of the system is given in Section 1 of [11]. The differential equations governing the maneuver is dependent on ω, angular velocity of the flight. Given the model, and an uncertain parameter ω, we assume $\omega \in [1,3]$ to evaluate our algorithm. And the initial set θ = [-1,1] $\times$ [-1,1] $\times$ [20,30] $\times$ [20,30]. We tune the `StatVer` module with $B = 9000$ and $c = 0.99$, yielding $K = 906$ (number of samples required) from Equation 2.30, and *type I error* of $err(B,c) = 0.01$ from Equation 2.31. And we tune `learnModel` module with $N = O = 100$ (We don't consider M as we are doing it for a particular time step). In Fig. 2.12, we show the reachable set for time step 2000, compared with `Flow*`, and the timing details are provided in Table 2.1.

Spacecraft Rendezvous. [53] provides a linear model of autonomous maneuver for rendezvous between two spacecraft. The dynamics of the two spacecraft in orbit, called as *target* and *chaser*, respectively, are derived from Kepler's laws. The two spacecrafts are assumed to be in same the orbital plane. The target is assumed to move on a circular orbit. The linearized model is given in Section 3.2 of [54]. The equations depend on parameters like $n = \sqrt{\frac{\mu}{r}}$, where $\mu = 3.986 \times 10^{14} m^3/s^2$ and $r = 42164km$ and $m_c = 500kg$, the mass of the spacecraft. For our experiments, we assume $m_c \in [475,525]$, as a result of perturbation due to measurement

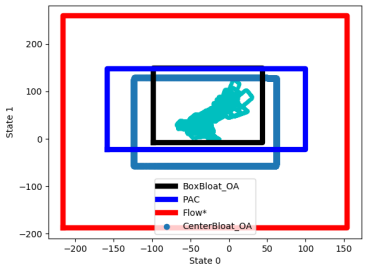


Figure 2.12: Over-approximate reachable sets of Flight Collision model. Dirty Blue: Computed using `bloatMean`; Black: Computed using `boxBloat`; Cyan: Random reachable set; Navy Blue: Computed using `modelLearn`; Red: Computed using `Flow*`.

and $n \in [97200, 97250]$. And let the initial θ = [-1,1] $\times$ [-1,1] $\times$ [0,0] $\times$ [0,0] $\times$ [1,1] $\times$ [1,1]. We tune the `StatVer` module with $B = 9000$ and $c = 0.99$, yielding $K = 906$ from Equation 2.30, and $err(B,c) = 0.01$ from Equation 2.31. We tune `learnModel` module with $N = O = 200$. In Fig. 2.13 (Right), we show the reachable set for time step 1901, compared with `Flow*`. *Note that Flow* stopped after 1901 time steps, as it could not compute the flowpipes anymore. This tool showed no such limitations*. The details of the time taken is given in Table 2.1.

Figure 2.13: Left: Over-approximate reachable sets of 5 Vehicle Platoon model. Dirty Blue: Computed using `bloatMean`; Black: Computed using `boxBloat`; Cyan: Random reachable set; Navy Blue: Computed using `modelLearn`; Red: Computed using `Flow*`. **Right**: Over-approximate reachable sets of Spacecraft Rendezvous model. Dirty Blue: Computed using `maxSV`; Black: Computed using `boxBloat`; Cyan: Random reachable set; Navy Blue: Computed using `modelLearn`; Red: Computed using `Flow*`.

More Benchmarks. We evaluated the algorithms on several other benchmarks — (i) An *Anaesthesia Delivery Model* [10], a 5 dimensional model; (ii) a 16 dimensional *Autonomous Quadrotors Model* [55]; (iii) a 5 dimensional *Motor Transmission Drive System* [56]; (iv) A 10 dimensional model of a *Self-balancing Two-wheeled Robot* [57]; (v) a 4 dimensional model of an *Adaptive Cruise Control (ACC)* [9]. The summary of all the evaluation is provided in Table 2.1.

2.4 Detailed Analysis

2.4.1 Computational Benefits of Using Frobenius Norm Over 2-Norm

The 2-norm of an interval matrix Λ can be computed using Theorem 7 of [20] as follows:

$$|| \Lambda ||_2 = sup_{|y|=e_n, |z|=e_n} || C_\Lambda + (yz^T) \cdot \Delta_\Lambda || . \tag{2.33}$$

where $e_n = (1, 1, \cdots, 1)^T$. Additionally, the elements of y and z can only be ± 1. Therefore, computing $|| \Lambda ||_2$, in the worst case, requires computing 2-norm of 2^{2n} matrices. That is, $O(2^{2n} f_2(n))$, where $f_2(n)$ is the time complexity for computing 2-norm of a matrix. On the other hand, Frobenius norm, which upper bounds 2-norm, is easy to compute. From Theorem 10 of [20], we have: $|| \Lambda ||_F = ||\ \ | C_\Lambda | + \Delta_\Lambda ||_F$. Hence, Frobenius norm of interval matrices can be computed by computing Frobenius norm of a single matrix. Therefore, using this, we provide conservative upper bounds of `Kagstrom1`, `Kagstrom2` and `Loan` (Theorem 6, Theorem 7 and Theorem 8), using Frobenius norm, in Theorem 9, Theorem 10 and Theorem 11.

2.4.2 Zonotope Based Reduction

Our second method for reducing the number of generators uses a tool for performing operations on polytopes, called `pypolycontain` [41][8]. In [41], the authors propose a representation for polytopes that makes checking of polytope containment very efficient. As a side effect, it also provides an efficient Zonotope reduction method that uses linear programming. When provided with an initial set of generators and a Zonotope as input, the tool performs refinement and returns a Zonotope that has same number of generators as the provided initial set of generators and overapproximates the input Zonotope. Given a star $\Theta_m = \langle a, G, P \rangle$ representing a reachable set of an n dimensional system, where G is a set of m basis vectors ($m \gg n$); using this open source tool, we obtain a Zonotope (therefore a star) Θ_n, with n basis vectors, $s.t.$ $\Theta_m \subseteq \Theta_n$. We modified this tool to take our star as input and return a star as output. Similar to the first method, this method also gives a quadratic improvement in performance and increases the order of overapproximation of the reachable set. Additionally, the overapproximation is also dependent on the initial set of generators provided to the tool.

[8] `https://github.com/sadraddini/pypolycontain`

Apart from the two approaches mentioned in this section, [58, 59] provides several other approaches that can be used to perform zonotope order reduction. However, in this paper, our intent is to show that one can use any existing zonotope order reduction based on their application. Here we explore two such methods, interested readers are referred to [58, 59] for more such reduction techniques.

2.4.3 Bloating the Reachable Set of Mean Dynamics

1. We compute the *mean dynamics* $A_c \in \Lambda$. If the uncertainties in the dynamics are intervals, then, this mean dynamics is often the matrix obtained by substituting all the intervals with the corresponding mid-points.

2. Let, $S = \{A_1, A_2, \cdots, A_N\}$ be a set of N random *sample dynamics* of Λ. Let, $d = \max\limits_{A \in S} \left\{ \texttt{dis}\big(\texttt{RS}_\theta(A_c, t), \texttt{RS}_\theta(A, t)\big) \right\}$. Where, $\texttt{dis}(S_1, S_2)$ is defined as the *Hausdorff distance* between S_1 and S_2.

3. Compute $\texttt{Bloat}\big(\texttt{RS}_\theta(A_c, t), d + \varepsilon\big)$, where $\varepsilon > 0$ is specified by the user.

2.4.4 Structure Guided Reachable Set Computation

In [17], authors considered uncertain matrices as linear matrix expressions and present sufficient conditions where the linear matrix expressions are closed under multiplication. We now present a sampling based reachable set computation technique that computes a provable overapproximation of the uncertain systems considered in [17]. However, if the conditions provided in [17] are not satisfied, then our reachable set need not be a conservative overapproximation. Hence, for such instances, one has to perform statistical verification, similar to the verification of the candidate reachable sets for other heuristics. We follow [17] and separate the variables in linear uncertain system as *LME preserving* and *non LME preserving*. For the LME preserving variables, we sample all the vertices of the domain D_Λ and grid based uniform sampling for all non LME preserving variables.

Definition 24. *Given an linear uncertain system $\dot{x} = \Lambda x$, a variable $y_i \in Vars$ is called* LME preserving *for M_Λ if and only if the following conditions are satisfied.*

1. *The variable y_i only has terms of degree one in M_Λ.*

2. *$supp(M_0) \times supp(M_0) \leq supp(M_0)$, where M_0 is the constant term in the matrix polynomial expression M_Λ.*

3. $supp(M_0) \times supp(M_i) + supp(M_i) \times supp(M_0) \leq supp(M_i)$ *where* M_i *is the coefficient matrix of the variable* y_i *in* M_Λ.

All the variables that do not satisfy these conditions are considered as non LME preserving.

We compute the oriented rectangular hull overapproximation of the reachable sets of dynamics sampled from the vertices of D_Λ and for non LME preserving variables, we generate additional sample dynamics, generated at regular intervals in D_Λ. The steps for computing the candidate reachable set using this heuristic are given below.

1. Compute all LME preserving variables in *Vars*.
2. Let $Vertices_\Lambda = \{A_1, A_2, \ldots, A_M\}$ be the sample dynamics obtained by evaluating the linear uncertain system on the vertices of D_Λ. Additionally, for all non LME preserving variables, generate K samples at regular intervals for each variable from D_Λ, called $Uniform_\Lambda$.
3. Compute the reachable set of all the dynamics in $Vertices_\Lambda$ and $Uniform_\Lambda$ at time t.
4. Compute the oriented rectangular hull overapproximation of all the reachable sets computed in $Vertices_\Lambda$ and $Uniform_\Lambda$.

Given a linear uncertain dynamics $\dot{x} = \Lambda x$, an initial set θ and time t, we denote the reachable set computed using this heuristic as $\texttt{uniformORH}(\Lambda, \theta, t)$. If all the variables in Λ are LME preserving, then $\texttt{uniformORH}(\Lambda, \theta, t)$ is an overapproximation of the reachable set of all the dynamics in Λ for initial set θ. We provide a proof of this is in Appendix, Section 2.4.4.1.

2.4.4.1 Structure Guided Reachable Set Computation

In this section, we prove that if all the variables in Λ are LME preserving, then $\texttt{uniformORH}(\Lambda, \theta, t)$ is an overapproximation of the reachable set of all the dynamics in Λ for initial set θ. This proof has two parts. First, we show that if all variables are LME preserving, then the matrix exponential is also an LME. Second, if the matrix exponential is an LME, then the union of matrix exponential of all samples in Λ can be obtained by computing convex hull of the matrix exponential of vertices of Λ. As a consequence, the convex hull of the reachable set of the vertices of Λ contains the union of reachable set of all samples in Λ.

Lemma 12. *If the uncertain linear system satisfies the conditions in Definition 16, then e^{M_Λ} is also an LME. Here, the matrix exponential $e^M = I + \frac{M}{1!} + \frac{M^2}{2!} + \frac{M^3}{3!} + \ldots$.*

Proof. Follows from the fact that M^n_Λ is also an LME for all $n \geq 1$ and LMEs are closed under addition operation. □

Lemma 13. *Given an uncertain linear system Λ with closed LME, and $\overline{\gamma_1}, \overline{\gamma_2}$ be two valuations of variables in D_Λ, we have $e^{\Lambda_{\beta\overline{\gamma_1}+(1-\beta)\overline{\gamma_2}}} = \beta e^{\Lambda_{\overline{\gamma_1}}} + (1-\beta)e^{\Lambda_{\overline{\gamma_2}}}$.*

Proof. From Lemma 12, we know that e^Λ is an LME, say $Q_0 + Q_1 y_1 + \ldots Q_k y_k$. Evaluating this LME over valuation $\overline{\gamma_1}$ would yield

$$e^{\Lambda_{\overline{\gamma_1}}} = Q_0 + Q_1\overline{\gamma_1}[1] + \ldots Q_k\overline{\gamma_1}[k].$$

and evaluating the LME over $\overline{\gamma_2}$ yields

$$e^{\Lambda_{\overline{\gamma_2}}} = Q_0 + Q_1\overline{\gamma_2}[1] + \ldots Q_k\overline{\gamma_2}[k].$$

Observing these expressions, it is trivial to observe that the lemma holds. □

Theorem 14. *Given an uncertain linear system Λ that is an LME closed under multiplication, and let $A_1, A_2, \ldots, A_M$ be the sample dynamics that are a result of assigning variables to the vertices of D_Λ.*

Given an initial set θ, let RS_i denote $RS_\theta(A_i, t)$ for $1 \leq i \leq M$. Then we have

$$\mathsf{ConvexHull}(RS_1, \ldots, RS_M) \supseteq \bigcup_{A \in \Lambda} RS_\theta(A, t).$$

Proof. consider an state $x_0 \in \theta$ and $A \in \Lambda$. The state reached by x_0 after time t with the dynamics $\dot{x} = Ax$ is $e^{At}x_0$.

Now, since $A \in \Lambda$ and $A_1, \ldots, A_M$ are the vertices of Λ, from Lemma 13, we have $\exists \lambda_1, \ldots, \lambda_M$ such that $0 \leq \lambda_i \leq 1$ and $\Sigma_{i=1}^M \lambda_i = 1$ such that

$$e^{At} = \lambda_1 e^{A_1 t} + \ldots + \lambda_M e^{A_M t}.$$

Therefore,

$$e^{At}x_0 = \lambda_1 e^{A_1 t}x_0 + \ldots + \lambda_M e^{A_M t}x_0.$$

Since $\lambda_1 e^{A_1 t} x_0 + \ldots + \lambda_M e^{A_M t} x_0 \in \mathsf{ConvexHull}(\mathtt{RS}_1, \ldots, \mathtt{RS}_M)$, the proof is complete. □

2.4.5 Singular Values based Reachable Set Computation

The steps to compute reachable set using singular values are:

1. Compute the *maximum singular value matrix* $A_{max} \in \Lambda$ using the technique described in [20] and compute the reachable set $\mathtt{RS}_\theta(A_{max}, t)$.
2. Let, $S = \{A_1, A_2, \cdots, A_N\}$ be a set of N random *sample dynamics* of Λ. Let, $d = \max\limits_{A \in S} \left\{ \mathtt{dis}\big(\mathtt{RS}_\theta(A_{max}, t), \mathtt{RS}_\theta(A, t)\big) \right\}$. Where, $\mathtt{dis}(S_1, S_2)$ is the *Hausdorff distance* between the two sets.
3. Compute $\mathtt{Bloat}\big(\mathtt{RS}_\theta(A_{max}, t), d + \varepsilon\big)$, where $\varepsilon > 0$ is provided by the user.

2.4.6 Model Learning Based Reachable Set Compution

Algorithm 6: Reachable set computation using Model Learning

input : Λ, θ, t_δ

output : Θ_t

$D_\Lambda \leftarrow$ Hyper-box representing the values of the uncertain variables in Λ ;

(C^*, κ^*)=`learnModel` $\big(Bloat(\theta, \delta_1), Bloat(D_\Lambda, \delta_2), Bloat(t_\delta, \delta_3) \big)$;

$\Theta_t = \{\}$;

for *all* $t \in t_\delta$ **do**

  $tmp = e^{A_c t} \cdot \mathtt{Box}(\theta) + C^* \cdot D_\Lambda$;

  $\Theta_t = \Theta_t \cup \{Bloat(tmp, \kappa^*)\}$

return Θ_t ;

2.4.6.1 Scenario Optimization Details

In this subsection, we provide a brief introduction to *Scenario Optimization* [50] (as explained in [51]). Given an optimization problem as follows [eq. (2) of [51]]:

$$\begin{aligned} \min_{\gamma \in \Gamma \subseteq \mathbb{R}^m} \quad & c^\intercal \gamma \\ \text{s.t.} \quad & f_\delta \leq 0, \ \ \forall \delta \in \Delta \end{aligned} \tag{2.34}$$

where $f_\delta(\gamma)$ are continuous continuous and convex functions over the m dimensional optimization variable γ for every $\delta \in \Delta$. Also, the sets Γ and Δ are convex and closed. The optimization problem in Equation 2.34 can be reformulated and solved, with statistical guarantees, as shown in [50], as follows [eq. (3) of [51]]:

$$\begin{aligned} \min_{\gamma \in \Gamma \subseteq \mathbb{R}^m} \quad & c^\intercal \gamma \\ \text{s.t.} \quad & \wedge_{i=1}^{K} f_{\delta_i} \leq 0, \ \ \forall \delta \in \Delta \end{aligned} \tag{2.35}$$

Note that the reformulation of the problem (in Equation 2.34), to the problem in equation 2.35 only considers finite subset out of the infinite number of constraints in Equation 2.34. Given, γ^*, the solution to Equation 2.35, the statistical guarantees can be given as follows [Theorem 1 in [51]]:

Theorem 15. *(Theorem 1 in [51]) If Equation 2.35 is feasible and attains unique optimal solution γ^*, and*

$$\varepsilon \geq \frac{2}{K}(ln\frac{1}{\beta} + m) \tag{2.36}$$

where $\varepsilon \in (0,1)$ and $\beta \in (0,1)$ are user-chosen error and confidence levels, then with at least $1-\beta$ confidence, γ^* satisfies all constraints in Δ but at most a fraction of probability measure ε, i.e., $P\big(\{\delta \in \Delta \mid f_\delta(\gamma^*) \leq 0\}\big) \leq \varepsilon$, where the confidence β is the K-fold probability p^K in $\Delta^k = \Delta \times \Delta \times \cdots \Delta$, which is the set to which the extracted sample $(\delta_1, \delta_2, \cdots, \delta_K)$ belongs.

The above conclusion still holds if the uniqueness of optimal solutions to Equation 2.35 is removed [51], since a unique optimal solution can always be obtained according to Tie-break rule if multiple optimal solutions occur.

2.4.6.2 Learning the Model

$$
\begin{aligned}
\min_{C,\kappa} \quad & \kappa \\
\text{s.t.} \quad & \forall (x_i \in \{x_i\}^M, \gamma_j \in \{\gamma_j\}^N, t_k \in \{t_k\}^O) \\
& w(C, x_i, \gamma_j, t_k) - y_{i,j,k} \leq \underbrace{[\kappa \ \kappa \ \cdots \ \kappa]^\intercal}_{n} \\
& y_{i,j,k} - w(C, x_i, \gamma_j, t_k) \leq \underbrace{[\kappa \ \kappa \ \cdots \ \kappa]^\intercal}_{n} \\
& \forall_{r \in [1,n], s \in [1,p]} \ -U_C \leq C[r,s] \leq U_C \\
& 0 \leq \kappa \leq U_\kappa
\end{aligned}
\tag{2.37}
$$

Let, the solution to the above optimization problem be (C^*, κ^*), where $C^*[i,j] = c^*_{i,j}$. Let, the shorthand of this learning module be denoted as

$\texttt{learnModel}(\theta, D_\Lambda, t_\delta) = (C^*, \kappa^*)$.

Statistical Guarantees We now formalize the statistical guarantees on the solution to the *Scenario Optimization* problem in Eq. (2.37). The statistical guarantees on the solution (C^*, κ^*) is given as follows.

Theorem 16. *Let*

$$
\begin{aligned}
\tau(x,\gamma) &= \{t \in t_\delta \mid w(C^*, x, \gamma, t) - RS(x, A_\gamma, t) \leq [\kappa^* \ \kappa^* \ \cdots \ \kappa^*]^\intercal\} \\
\chi(u) &= \{x \in \theta \mid P_t(\tau(x,u)) \geq 1 - \varepsilon_1 \textit{ with confidence } 1 - \beta_1\} \\
\mathcal{U} &= \{u \in D_\Lambda \mid P_x(\chi(u)) \geq 1 - \varepsilon_2 \textit{ with confidence } 1 - \beta_2\}
\end{aligned}
$$

then we have: $P_u(\{u \mid u \in \mathcal{U}\} \geq 1 - \varepsilon_3$ *with confidence* $1 - \beta_3$

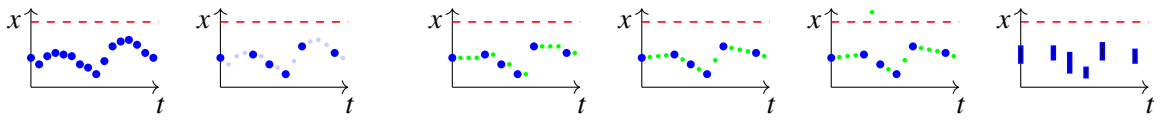
where P_t, P_x, P_u *are the probability measures on the sets* t_δ, θ, U *respectively. And* $\varepsilon_i \geq \frac{2}{M}(ln\frac{1}{\beta_i} + np + 1)$*, for* $i \in \{1,2,3\}$

Proof. Follows directly from Theorem 3 of [51]. □

CHAPTER 3: MONITORING AUTONOMOUS SYSTEMS—ANALYZING LOGS TO DETECT A CRASH

The prevalence of distributed autonomous systems is experiencing significant growth, along with the emergence of safety concerns associated with them. Traditional formal verification methods typically rely on a white-box model, which is often unavailable due to certain components being black-box or the absence of a formal model for the entire system. Moreover, formal verification techniques for autonomous systems often encounter challenges related to state space explosion, impeding their scalability and effectiveness. Consequently, *monitoring*, as a lightweight yet feasible verification technique, can bring practical results of high importance for larger models.

Monitoring aims at analyzing the log of a concrete system, so as to deduce whether a specification (*e.g.* a safety property) is violated. Monitoring can be done *offline* (*i.e.* after the system execution, assuming the knowledge of the entire log), or *online* (at runtime, assuming a partial log). When the log is an aperiodic timed sequence of valuations of continuous variables, with a logging *not* occurring at every discrete time step, and when the system under monitoring is a black box, a major issue is: how to be certain that, in between two discrete valuations, the specification was not violated at another discrete time step at which no logging was performed? For example, consider a system for which a logging at every discrete time step would yield the log depicted in Fig. 3.1a. Assume the logging was done at only *some* time steps, given in Fig. 3.1b, due to some sensor faults, or to save energy with only a sparse, scattered logging. How to be certain that, in between two discrete samples, another discrete sample (not recorded) did not violate the specification? For example, by just looking at the discrete samples in Fig. 3.1b, there is no way to formally guarantee that the unsafe zone (*i.e.* above the red, dashed line) was never reached by another discrete sample which was



(a) Full **(b)** Monitored **(c)** Extrapo. **(d)** Extrapo. **(e)** Violation **(f)** Uncertain

Figure 3.1: Monitoring at discrete time steps

not recorded. In many practical cases, a piecewise-constant or linear approximation (see, *e.g.* Figs. 3.1c and 3.1d, where the large blue dots denote actual samples, while the small green dots denote reconstructed samples using some extrapolation) is arbitrary and not appropriate; even worse, it can yield a "safe" answer, while the actual system could actually have been unsafe at some of the missing time steps. On the contrary, assuming a completely arbitrary dynamics will always yield "potentially unsafe"—thus removing the interest of monitoring. For example, from the samples in Fig. 3.1b, without any knowledge of the model, one can always envision the situation in Fig. 3.1e, which shows the variable x crossing the unsafe region (dashed) at some unlogged discrete time step—even though this is unlikely if the dynamics is known to vary "not very fast".

Contribution. In this chapter, the problem of performing monitoring over a set of scattered and *uncertain* samples is addressed. First, uncertainties arising from the sensors are addressed by accommodating for uncertain samples, represented as zonotopes encompassing the continuous variables. In other words, instead of providing a constant value for the continuous variables at each logged timestamp, the log presents a zonotope. An illustration of a straightforward instance of an uncertain log involving a single variable, denoted as x, is presented in Fig. 3.1f in the form of basic intervals. In addition, the timestamp at each discrete sample of the log can itself be uncertain, in the form of an *interval* (not shown in Fig. 3.1f, where the timestamp is punctual). Second, rather than defining the bounding model with an LHA to over-approximate the system behavior, and following the approach of "model-bounded monitoring" proposed in [12], an extension of linear dynamical systems is employed. This extension incorporates uncertainty by allowing for uncertainties in the dynamics matrix [11]. Having some over-approximated knowledge of the system is a natural assumption in practice: when monitoring a car, one generally knows an upper-bound on its maximum speed, or on its maximum acceleration (perhaps depending on its current speed). In order to handle the flexible dynamics of the extended linear dynamical systems, a recent technique [18] (see Chapter 2) is employed. This technique enables an efficient reachability analysis for uncertain linear dynamical systems. By utilizing an over-approximation of the actual system, this approach focuses on identifying and disregarding unlikely behaviors, including the safety violation illustrated in Fig. 3.1e.

The first major contribution of the this chapter is the introduction of a novel and rigorous analysis technique for offline monitoring of safety properties concerning scattered *uncertain* samples. This technique involves employing uncertain linear systems as an over-approximation of the system. By utilizing this

over-approximation, it becomes possible to extrapolate the system's behavior from the most recent known sample and eliminate the possibility of safety violations at certain missing discrete samples. Note that this approach uses some discrete analysis as underlying reachability computation technique, and will not however guarantee the absence of safety violations at arbitrary (continuous) timestamps; its main advantage is to offer formal guarantees in the context of missing discrete samples for a given logging granularity. The second main contribution focuses on *energy-efficient online monitoring*. For each recorded sample, reachability analysis is run to derive the smallest next discrete time step t in the future at which the safety property may be violated depending on the latest known sample and the over-approximated model dynamics. In a context in which monitoring simply observes the behavior and does not lead to corrective actions, any sample before t is useless because we *know* from the over-approximated model dynamics that no safety violation can happen before t. Therefore, one can schedule the next sample at time t, which reduces the number of discrete samples, and therefore the energy consumption and bandwidth use. The third main contribution of this chapter is to introduce a tool `MoULDyS` to perform monitoring of autonomous systems as proposed in this paper. `MoULyS` implements efficient offline and online monitoring algorithms of black-box autonomous systems w.r.t. safety properties. `MoULDyS` takes as input an uncertain log (with noisy and missing samples), as well as a bounding model in the form of an uncertain linear system; this latter model plays the role of an over-approximation so as to reduce the number of false alarms. `MoULDyS` is Python-based and available under the *GNU General Public License v3.0* (`gpl-3.0`). The practical applicability of these approaches on three benchmarks—an anesthesia model, an adaptive cruise controller, and an aircraft orbiting system—is shown in [27, 28].

About the chapter. The content of this chapter is taken from [27, 28, 29].

3.1 Preliminaries

This section lays out the notations and definitions used in the rest of the chapter. Given an initial set $\theta_0 \subset \mathbb{R}^n$ and given a time step t, $\theta_t \subset \mathbb{R}^n$ denotes the reachable set of the system (represented as uncertain linear dynamical system) at time step t. Next, we define a log of the system with uncertainties in both in system states and timestamps.

Definition 25 (Uncertain log)**.** *Given an uncertain linear dynamical system, a finite length* uncertain log *is defined as follows:*

$$\ell = \Big\{ \big(\hat{\theta}_t, [t^{lb}, t^{ub}]\big) \mid \theta_t \subseteq \hat{\theta}_t, \textit{for some } t \in [t^{lb}, t^{ub}], t^{lb} \leq t^{ub} \leq H \Big\},$$

where H is a given time bound.

Each tuple $\big(\hat{\theta}_t, [t^{lb}, t^{ub}]\big)$ is called a *sample*. Observe that, both the system state $\hat{\theta}_t$ and timestamp $[t^{lb}, t^{ub}]$, in a sample are not necessarily reduced to a *point*. The length of log ℓ—number of samples in ℓ—is given by $|\ell|$. Given an uncertain log ℓ, the *k-th* sample of ℓ is given as $\ell_k = \big(\hat{\theta}_{t_k}, [t_k^{lb}, t_k^{ub}]\big)$, where $\hat{\theta}_{t_k}$ is an over-approximation of the system state at some time step t_k, where $t_k \in [t_k^{lb}, t_k^{ub}]$. Such a sample denotes that over-approximate state of the system was observed to be $\hat{\theta}_{t_k}$, for some time step t_k, where $t_k \in [t_k^{lb}, t_k^{ub}]$ (but the exact t_k is not known). This formalism facilitates modeling of situations when the samples are collected over an uncertain channel (such as a shared network with delays) and the precise timestamp is unknown. Note that the log can be *scattered*—it does not necessarily contain a sample for each $t \in \{1, \ldots, H\}$, *i.e.*

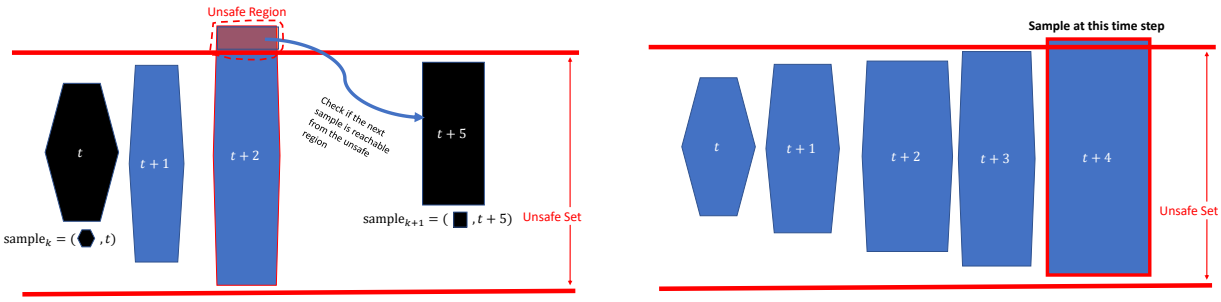
$$\{1, \cdots, H\} \not\subseteq \bigcup_k [t_k^{lb}, t_k^{ub}].$$

Further note that the uncertainties in the logs, arising from the sensor uncertainties of the logging system, are independent of the uncertainties in the system modeling. Given two consecutive samples $\ell_k = \big(\hat{\theta}_{t_k}, [t_k^{lb}, t_k^{ub}]\big)$ and $\ell_{k+1} = \big(\hat{\theta}_{t_{k+1}}, [t_{k+1}^{lb}, t_{k+1}^{ub}]\big)$, we assume that their timestamps do not intersect, *i.e.* $[t_k^{lb}, t_k^{ub}] \cap [t_{k+1}^{lb}, t_{k+1}^{ub}] = \emptyset$; or, put differently, $t_k^{ub} < t_{k+1}^{lb}$.

When the samples are being collected locally, timestamps can be precisely known (for all practical purposes). In such cases—while we still can have uncertainties in the system state—the timestamp can be known precisely. This leads to a simpler case of logs as defined as follows.

Definition 26 (Fixed timestamp uncertain log)**.** *A finite length* uncertain log with fixed timestamps *is defined as follows:* $\ell = \{(\hat{\theta}_t, t) \mid \theta_t \subseteq \hat{\theta}_t, t \leq H\}$*, where H is a given time bound.*

We call a fixed timestamp uncertain log ℓ *accurate* if it satisfies the following condition: $\forall 1 \leq k \leq |\ell| : \hat{\theta}_{t_k} = \theta_{t_k}$. Given an uncertain linear dynamical system, $x^+ = \Lambda x$ with an initial set $\theta_0 \subset \mathbb{R}^n$, an *over-approximate reachable set of* x^+ *at time step t* is $\texttt{overReach}(\Lambda, \theta_0, t)$, such that $\theta_t \subseteq \texttt{overReach}(\Lambda, \theta_0, t)$. The technique proposed in [18] is used to compute $\texttt{overReach}(\Lambda, \theta_0, t)$ in this work (see Section 2.3.1.2).



(a) Offline **(b) Online**

Figure 3.2: (3.2a): **Offline Monitoring**. Black: Two consecutive samples, k and $k+1$, at time steps t and $t+5$ respectively. Blue: The over-approximate reachable set computed from sample k using `overReach`(.). (3.2b): **Online Monitoring**. Blue: Over-approximate reachable set computed, at each step, using `overReach`(.).

3.2 Offline Monitoring: Analyzing Noisy Logs with Missing Samples

Given an uncertain log—arising, *e.g.* due to faulty sensors and collected over a shared network—this section proposes an algorithm to infer the safety of a system represented as uncertain linear dynamical system. In this setting, we assume full knowledge of the (possibly scattered) uncertain log, usually after an execution is completely over. In a first step, logs are assumed to be known with full certainty regarding the time step; that is, the input is a *fixed timestamp uncertain log*. The case of a fully uncertain log will be addressed in Section 3.2.1.

Before the offline algorithm is proposed, it is illustrated in Fig. 3.2a. Consider two consecutive samples k and $k+1$, marked in black, at time steps t and $t+5$ respectively. The reachable sets, in blue, represent the over-approximate behaviors possible by the system between time steps t and $t+5$. Consider the case where at time step $t+2$ the over-approximate reachable set intersects with the unsafe region. Once the algorithm detects a possible unsafe behavior, it computes the intersection between the over-approximate reachable set (here, the reachable set at time-step $t+2$) and the unsafe set. Then it checks whether the reachable set, given in the next sample $(k+1)$, is reachable from the unsafe region—if yes, it infers *unsafe*; if not, it infers *safe*. Next, the offline monitoring method in proposed in Algorithm 7 for a given fixed timestamp uncertain log ℓ.

Description. As we first consider an input log given in the form of a fixed timestamp uncertain log (Definition 26), we consider a simpler version of Algorithm 7 without the highlighted lines: at line 2, we have $t_k^{lb} = t_k^{ub} = t_k$, and at line 3 we have $t_{k+1}^{lb} = t_{k+1}^{ub} = t_{k+1}$; in addition, the **for** loops at lines 4 and 5 can be discarded. Let us now describe this simpler version of the algorithm. The **for** loop, starting in line 1, traverses through each sample, and checks if the system can reach a possibly unsafe behavior between two consecutive

Algorithm 7: Offline monitoring

input : An uncertain log ℓ of a system $x^+ = \Lambda x$, and an unsafe set $\mathcal{U}$.
output : Return *safe* (resp. *unsafe*) if the actual system behavior is safe (resp. potentially unsafe).

1 **for** $k \in \{1, \ldots, |\ell| - 1\}$ **do**
2 $\quad \left(\hat{\theta}_{t_k}, \left[t_k^{lb}, t_k^{ub}\right]\right) \leftarrow \ell_k$; // current sample, with interval time step
3 $\quad \left(\hat{\theta}_{t_{k+1}}, \left[t_{k+1}^{lb}, t_{k+1}^{ub}\right]\right) \leftarrow \ell_{k+1}$; // next sample, with interval time step
4 $\quad$ **for** $t_k \in \left\{t_k^{lb}, \ldots, t_k^{ub}\right\}$ **do**
5 $\quad\quad$ **for** $t_{k+1} \in \left\{t_{k+1}^{lb}, \ldots, t_{k+1}^{ub}\right\}$ **do**
6 $\quad\quad\quad t_\Delta = t_{k+1} - t_k - 1$; // time gap between two possible time steps of the two samples
/* Compute reachable set for all possible time steps (from the two intervals of time steps) between two samples. Check if any of the sets intersect with the unsafe set */
7 $\quad\quad\quad$ **for** $p \in \{1, \ldots, t_\Delta - 1\}$ **do**
8 $\quad\quad\quad\quad$ **if** $\hat{\theta}_{t_k+p} \cap \mathcal{U} \neq \emptyset$ **then**
/* Refinement process starts */
9 $\quad\quad\quad\quad\quad \psi \leftarrow \hat{\theta}_{t_k+p} \cap \mathcal{U}$; // compute the unsafe region of the system
10 $\quad\quad\quad\quad\quad t_d = t_{k+1} - (t_k + p)$;
11 $\quad\quad\quad\quad\quad \vartheta \leftarrow \texttt{overReach}(\Lambda, \psi, t_d)$;
/* Check if the next sample is reachable from the unsafe region */
12 $\quad\quad\quad\quad\quad$ **if** $\vartheta \cap \hat{\theta}_{t_{k+1}} \neq \emptyset$ **then**
13 $\quad\quad\quad\quad\quad\quad$ **return** *unsafe* ; // the next sample is reachable from the unsafe region
14 $\quad\quad\quad\quad \hat{\theta}_{t_k+p+1} \leftarrow \texttt{overReach}(\Lambda, \hat{\theta}_{t_k+p}, 1)$;
15 **return** *safe* ;

samples (computed in lines 2-3), using over-approximate reachable set computation. If the over-approximate reachable set between two consecutive samples intersect with the unsafe set (line 8), a refinement is performed as follows (lines 9–13): The unsafe region is computed (intersection between unsafe set and over-approximate reachable set) in line 9, then checked if the next sample can be reached from the unsafe region (lines 11–13). If the next sample is reachable from the unsafe behavior, we conclude the system is unsafe (lines 12–13).

Soundness and incompleteness. Our proposed offline monitoring approach is *sound* at discrete time steps, but not *complete*—there might be cases where our algorithm returns *unsafe* even though the actual system is *safe*. The primary reason for its incompleteness is due to the fact that `overReach`(.) computes an over-approximate reachable set. Formally:

Theorem 17 (soundness at discrete time steps for a fixed timestamp uncertain log)**.** *If the actual system is unsafe at some discrete time step, then Algorithm 7 returns unsafe. Equivalently, if Algorithm 7 returns safe, then the actual system is safe at every discrete time step.*

Proof. We consider a fixed timestamp uncertain log. Let the actual trajectory τ, between two samples k and $k+1$, become unsafe at time step t_{un}. Therefore, the over-approximate reachable set, computed by $\texttt{overReach}(\cdot)$ at time step t_{un}, will also intersect with the unsafe set (due to soundness of $\texttt{overReach}(\cdot)$). Note that the actual trajectory τ, originating from the sample k, intersects the unsafe region at time step t_{un}, and reaches the sample $k+1$. The refinement module (lines 7, 9–13), using over-approximate reachable sets will therefore infer the same, concluding the system behavior to be unsafe. □

3.2.1 What If Timestamps Are Noisy, Too?

We now extend the offline monitoring to logs with uncertainty not only in the state dimension, but also in the timestamp dimension (as in Algorithm 25). The extended version of the algorithm is the full Algorithm 7, including the highlighted parts. We namely add a pair of **for** loops at lines 4 and 5, iterating over each (concrete) timestamp in the current uncertain sample ($[t_k^{lb}, t_k^{ub}]$) and over the next uncertain sample ($[t_{k+1}^{lb}, t_{k+1}^{ub}]$). That is, we handle uncertainty over the logging times by iterating over each possible concrete log time in the logged interval. This is a crux to ensure soundness of this approach, and guarantee that a safe answer indeed guarantees safety of the actual system (at all discrete time steps).

Conversely, and as in Section 3.2, our algorithm is not necessarily *complete* (our algorithm might return *unsafe* even though the actual system is *safe*) due to the over-approximation of the reachable set computation. We prove formally the soundness of Algorithm 7 below:

Theorem 18 (soundness at discrete time steps for an uncertain log)**.** *If the actual system is unsafe at some discrete uncertain time step, then Algorithm 7 returns unsafe. Equivalently, if Algorithm 7 returns safe, then the actual system is safe at every discrete uncertain time step.*

Proof. Let the actual trajectory τ, between two samples k and $k+1$, with uncertain time steps $[t_k^{lb}, t_k^{ub}]$ and $[t_{k+1}^{lb}, t_{k+1}^{ub}]$, become unsafe at time step t_{un}. Algorithm 7 computes the reachable set of all possible time steps between $[t_k^{lb}, t_k^{ub}]$ and $[t_{k+1}^{lb}, t_{k+1}^{ub}]$. Therefore, the over-approximate reachable set, computed by $\texttt{overReach}(\cdot)$ at time step t_{un}, will also intersect with the unsafe set (due to soundness of $\texttt{overReach}(\cdot)$). Note that the actual trajectory τ, originating from the sample k, intersects the unsafe region at time step t_{un}, and reaches the sample $k+1$. The refinement module (lines 7, 9–13), using over-approximate reachable sets will therefore infer the same, concluding the system behavior to be unsafe. □

3.3 Online Monitoring: Gather Samples Only When Necessary

We now move to *online* monitoring. In contrast to Section 3.2, in the online setting, timestamps are necessarily exact, as we suppose we can trigger (instantaneously) a sample. However, the logged states are still uncertain (as in Definition 26). We propose the online monitoring method in Algorithm 8.

Algorithm 8: Online monitoring

input : An uncertain system $x^+ = \Lambda x$, an unsafe set $\mathcal{U}$, time bound H.
output : Return *safe iff* the actual system behavior is safe.

1 $\hat{\theta}_0 \leftarrow$ Sampling at time step 0 ; // initial behavior of the system.
/* Check whether the initial behavior is safe */
2 **if** $\hat{\theta}_0 \cap \mathcal{U} \neq \emptyset$ **then return** *unsafe* ;
3 **for** $t \in \{1, 2, \ldots, H-1\}$ **do**
4 $\quad \hat{\theta}_{t+1} \leftarrow \texttt{overReach}(\Lambda, \hat{\theta}_t, 1)$; // over-approximate reachable set at next step
$\quad$ /* Check whether the over-approximate reachable set is unsafe */
5 $\quad$ **if** $\hat{\theta}_{t+1} \cap \mathcal{U} \neq \emptyset$ **then**
6 $\quad\quad \ell_{t+1} \leftarrow$ Sample at time step $t+1$;
$\quad\quad$ /* Check whether the actual reachable set is unsafe */
7 $\quad\quad$ **if** $\ell_{t+1} \cap \mathcal{U} \neq \emptyset$ **then**
8 $\quad\quad\quad$ **return** *unsafe* ;
9 $\quad\quad \hat{\theta}_{t+1} = \ell_{t+1}$; // reset to actual behavior
10 **return** *safe*;

Description. The online monitoring algorithm begins by sampling the system at the initial time step, say 0, in Algorithm 1. As a sanity check, the initial behavior of the system is confirmed to be safe in line 2. The **for** loop starting in line 2—where each iteration corresponds to the set of actions for a time step t—performs the following: At a given time step t, the over-approximate reachable set is computed at the next time step $t+1$ (line 6). If the computed over-approximate reachable set intersects with the unsafe set, the system is sampled at time step $t+1$ to check if the actual behavior is also unsafe (lines 5–9). If safe, the behavior is rest (line 9); if unsafe, it is returned to be *unsafe* (line 8). Intuitively, this method samples the actual system only when the over-approximate reachable set, computed by `overReach`(.), intersects the unsafe set. This process is illustrated in Fig. 3.2b.

Soundness and completeness. The online monitoring algorithm is correct (safe and complete) at discrete time steps, *provided* the samples are accurate—it returns safe if and only if the actual behavior of the system

is safe at all discrete time steps, when accurate samples are obtained. Intuitively, we get the completeness from the fact that it returns unsafe if and only if the (accurate) sample is unsafe. Formally:

Theorem 19. *Algorithm 8 returns safe iff the actual behavior at all discrete time steps is safe.*

Proof. The soundness proof—if the actual behavior is unsafe, Algorithm 8 infers unsafe—is straightforward. Hence, we now argue the completeness—if the actual behavior is safe, Algorithm 8 infers *safe*. Note that, Algorithm 8 infers the system behavior as *unsafe* only when a sampled log (actual behavior) becomes unsafe. Therefore, if the samples are free from uncertainties (*i.e.* exact), Algorithm 8 is complete. □

Remark. While our aim is to consider continuous systems, note that, for *discrete-time systems*, this approach is entirely correct (sound and complete), without the need for a restriction to "discrete time steps", since one can find a granularity small enough for the discrete-time evolution. This is notably the case for systems where the behavior does not change faster than a given frequency (*e.g.* the processor clock).

3.4 `MoULDyS`: A Monitoring Tool for Autonomous Systems

Here, we present a monitoring tool, `MoULDyS`[1], which aims to identify potential safety breaches by analyzing logs. The distinctive characteristics of `MoULDyS` can be summarized as follows.

1. The possibility to monitor aperiodic logs, or periodic logs with missing samples, and with possible noise over the recorded data; and.

2. The presence of a bounding model following the formalism of uncertain linear systems.

3.4.1 Software Architecture

The architecture of `MoULDyS` is given in Fig. 3.3. Each block in Fig. 3.3 represents a core part (either functional or input) of the tool, and the arrows indicate the flow of data. The dataflow of `MoULDyS`, for both Figs. 3.4 and 3.5 (offline and online monitoring respectively), starts from the blocks in the left (*e.g.* bounding model, unsafe set, etc.) and ends at the extreme right of the figures (outputting the safety status, visualization, and/or synthesized log). Since both the offline (`Block 1` of Figs. 3.3 and 3.4) and online monitoring algorithm (`Block 2` in Figs. 3.3 and 3.5) uses reachability of uncertain linear systems (`Block 3` of Fig. 3.3),

[1] `https://sites.google.com/view/mouldys`

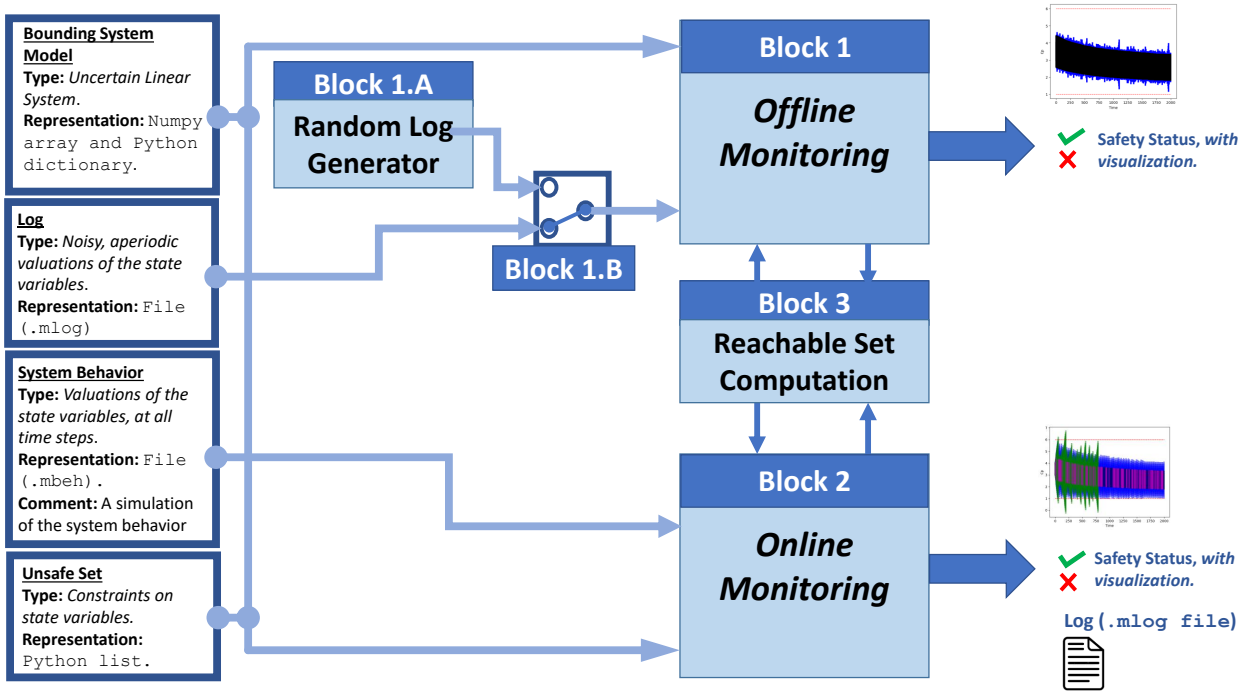


Figure 3.3: MoULDyS ***Architecture.*** Each block identifies a core part of the tool, and the arrows indicate the flow of data.

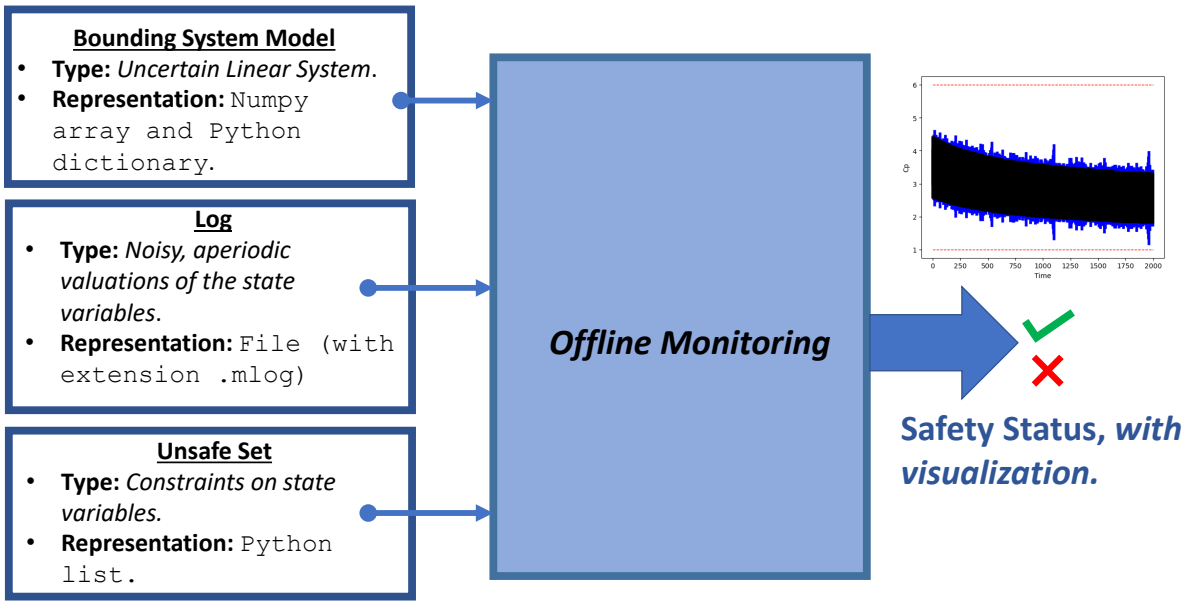


Figure 3.4: Dataflow diagram of the offline monitoring functionality of MoULDyS.

a data exchange occurs between Block 1 and Block 3, as well as Block 2 and Block 3. MoULDyS implements a built-in reachability algorithm. The native reachable set computation support facilitates faster

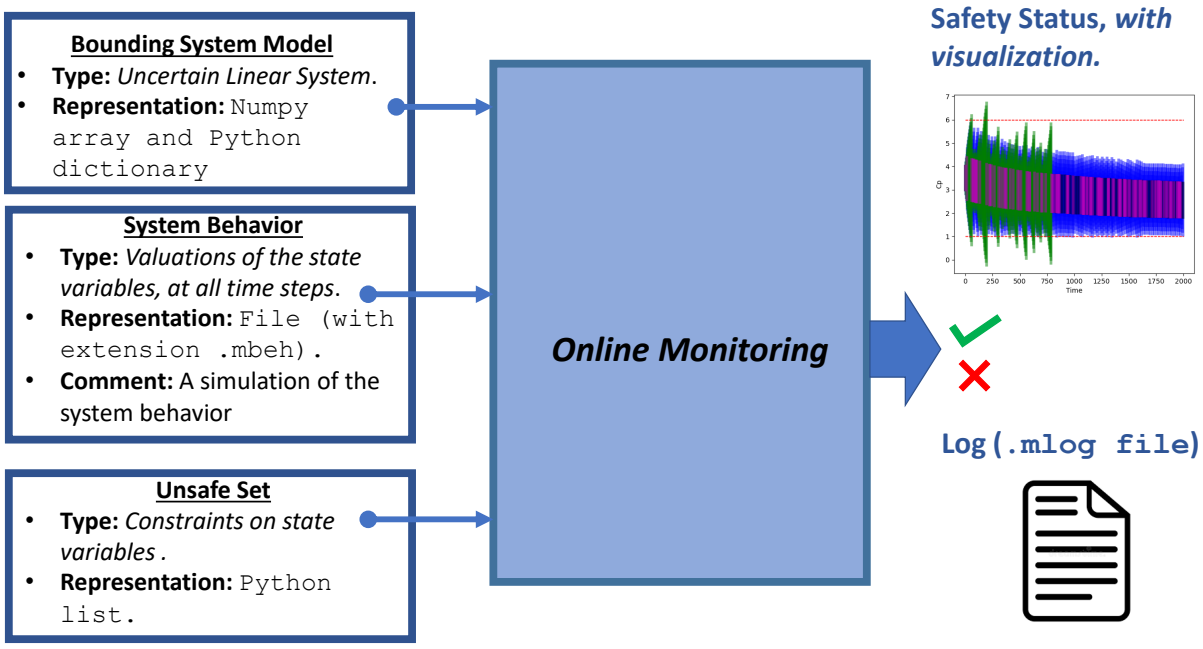


Figure 3.5: Dataflow diagram of the online monitoring functionality of MoULDyS.

computing (*i.e.* no data exchange with third party tool is required, which would potentially require additional data reformatting) with no additional tool installation required. MoULDyS implements efficient technique for computing such over-approximations presented in [18]. After obtaining the reachable sets, MoULDyS verifies the safety of these sets by comparing them against provided safety specifications. These safety specifications are constraints on state variables, which can be complex and involve multiple state variables (*e.g.* linear inequalities involving several state variables). To represent such safety specifications, MoULDyS uses zonotopes. MoULDyS can check multiple safety specifications, each represented as a zonotope and involving several state variables.

3.4.2 Implementation

MoULDyS is implemented using Python 3.7.x, and runs in a Linux environment. The tool can be used in the following two ways. On the one hand, users can use it through the provided virtual machine[2], which already contains all the necessary dependencies and has the path variable set. Nevertheless, users are still required to obtain and install the Gurobi license themselves, since Gurobi only grants free academic licenses to individuals. On the other hand, the tool can also be downloaded and setup from its public GitHub

[2] 10.5281/zenodo.7888502

repository[3]. If users aim to recreate the results in the paper or simply employ it for basic monitoring purposes, using the provided virtual machine is recommended. However, if the tool will be used for research and development purposes, it is recommended to download and set it up on a local machine. The detailed installation instructions are provided in [60]. The system model (represented as an uncertain linear system), in `MoULDyS`, is represented with a `numpy` array and a dictionary. The unsafe set is represented with a Python list. An example code of how to encode the system model and its unsafe specification can be found in the public repository[4]. The log and the system behavior, on the other hand, is given as a file to `MoULDyS` (`MoULDyS` can also generate random logs; discussed later). Logs can either be represented as zonotopes or intervals (an example of the required file can be found in `MoULDyS` public repository[5]).

Both the online and offline monitoring algorithm (`Block 1` and `Block 2` of Fig. 3.3) have been implemented in Python using standard libraries, namely `numpy`, `scipy` and `mpmath`. The reachable set computation (`Block 3` of Fig. 3.3) module implements the algorithm proposed in [18] in Python. Both the online and the offline module require performing intersection checking of zonotopes, which has been implemented as an optimization formulation using `Gurobi`. `Gurobi` has been further used to visualize the reachable sets.

Installation. While the virtual machine comes preinstalled with the required dependencies, setting up `MoULDyS` on a local machine requires installing the following dependencies: `numpy`, `scipy`, `mpmath`, `pandas`, `Gurobi` along with `gurobipy` (`Gurobi` requires an *ad-hoc* install but is free to use for academic purposes). The detailed steps for installing `MoULDyS` are given in the installation guide [60].

User Guide. A tutorial on how to use several functionalities of `MoULDyS`, along with sample codes (encoding a toy dynamics), is given in the user guide [61].

3.4.3 Software Functionalities

In this section, we discuss the core functionalities of `MoULDyS`:

Offline Monitoring. The offline monitoring requires the bounding model of the system (represented as an uncertain linear system) to be given as input. Further, a log of the system behavior is required, which can be achieved by the following ways:

[3] `https://github.com/bineet-coderep/MoULDyS/releases/tag/v1.1`

[4] `https://www.github.com/bineet-coderep/MoULDyS/blob/main/src/tutorial/TutorialOfflineMonitoring.py`

[5] `https://www.github.com/bineet-coderep/MoULDyS/blob/main/data/toyEg_5_interval.mlog`

1. If the user already has a log to be monitored, it can be simply passed as an input to `Block 1` of Fig. 3.3.
2. Alternatively, `MoULDyS` can also generate a random (noisy and aperiodic) log of the system, from a given initial set, using `Block 1.A`. The selection between the two possible choices is facilitated by `Block 1.B`. Analyzing the log, the final output of the offline monitoring is either `safe` (indicating the system behavior is certainly safe at all time steps), or `possibly-unsafe` (indicating the system might have shown unsafe behavior). An example code snippet to perform offline monitoring, on a toy example, can be found in its public repository.[6]

Online Monitoring. The online monitoring requires the bounding model of the system (represented as an uncertain linear system) to be given as input, as well as the actual behavior of the system. The actual behavior of the system is given as a file (with extension `.mbeh`) representing the values of the state variables at every time step—an example of the expected file (with extension `.mbeh`), containing valuation of the state variables at every time step, can be found in its public repository.[7] The final output of this feature is the safety status of the system (`safe`/`possibly-unsafe`), and a synthesized log. An example code snippet to perform online monitoring, on a toy example, can be found in its public repository[8].

As a side functionality, `MoULDyS` also allows to generate random logs (using `Block 1.A`). While this is not strictly speaking part of monitoring, it greatly helps to perform experiments using `MoULDyS`. Basically, given a bounding model, `MoULDyS` can generate a random log following the bounding model.

3.5 Case Studies

Environment. All the experiments were performed on a Lenovo ThinkPad Mobile Workstation with i7-8750H CPU with 2.20 GHz and 32 GiB memory on Ubuntu 20.04 LTS operating system (64 bit).

Implementation details vis-à-vis Algorithm 7 and Algorithm 8. The intersection checking between two sets in Algorithm 7 and Algorithm 8 has been implemented as an optimization formulation in `Gurobi`. That is, given two sets, our implementation of intersection check returns true iff the two sets intersect. In other

[6] `https://www.github.com/bineet-coderep/MoULDyS/blob/main/src/tutorial/TutorialOfflineMonitoring.py`

[7] `https://www.github.com/bineet-coderep/MoULDyS/blob/main/data/toyEg_5_interval.mbeh`

[8] `https://www.github.com/bineet-coderep/MoULDyS/blob/main/src/tutorial/TutorialOnlineMonitoring.py`

words, our intersection check is *exact*. In contrast, *computing* the result of the intersection between two sets adds an over-approximation in our implementation—given two sets, we compute a box hull of the two sets and then compute intersection of the two box hulls. Therefore, the only over-approximate operation we perform in Algorithm 7 and Algorithm 8—apart from `overReach`$(\cdot)$—is Algorithm 7, line 9.

Generating scattered uncertain logs for offline monitoring. At each time step, the logging system may take a snapshot of the system evolution at that time step; the logging occurs with a probability p (given). In other words, at each time step, it records the evolution of the system with probability p. Clearly, due to the probabilistic logging, this logger is not guaranteed to generate periodic samples. We also do not assume that the samples logged by the logging system, at each time step, are accurate—the logging system, due to sensor uncertainties, logs an over-approximate sample of the system at that time step. In our experiments, each log was generated statically from our bounding model (the uncertain linear dynamical system) by simulating its evolution from an uncertain initial set (*i.e.* not reduced to a point). In the end, we get an uncertain log.

Logging system for online monitoring When the logging system is triggered, at a time step, to generate a sample, the logging system records the evolution of the system and sends it to the online monitoring algorithm. Similar to the offline logging system, we do not assume that the samples logged by the logging system are perfectly accurate—the logging system, due to sensor uncertainties, logs an over-approximate sample of the system at that time step. That is, we use the same method—as the offline logging system—to generate logs (statically), but unlike the offline algorithm, the online algorithm uses the samples only when required. For both our case studies, all the generated logs are *safe*.

Research questions We consider the following research questions in our case studies:

1. Effect of logging probability (number of log samples) on the rate of false alarms raised by the offline monitoring—inferring a behavior as "potentially unsafe" when the actual behavior is "safe".

2. For offline monitoring, does the size of the samples (in other words, volume of the set obtained as sample), gathered at each step, have an impact on the rate of false alarms? Put it differently, what is the effect, *vis-à-vis* false alarms, of the amount of the uncertainty in the log?

3. For online monitoring, how frequent is the logging system triggered to generate a sample?

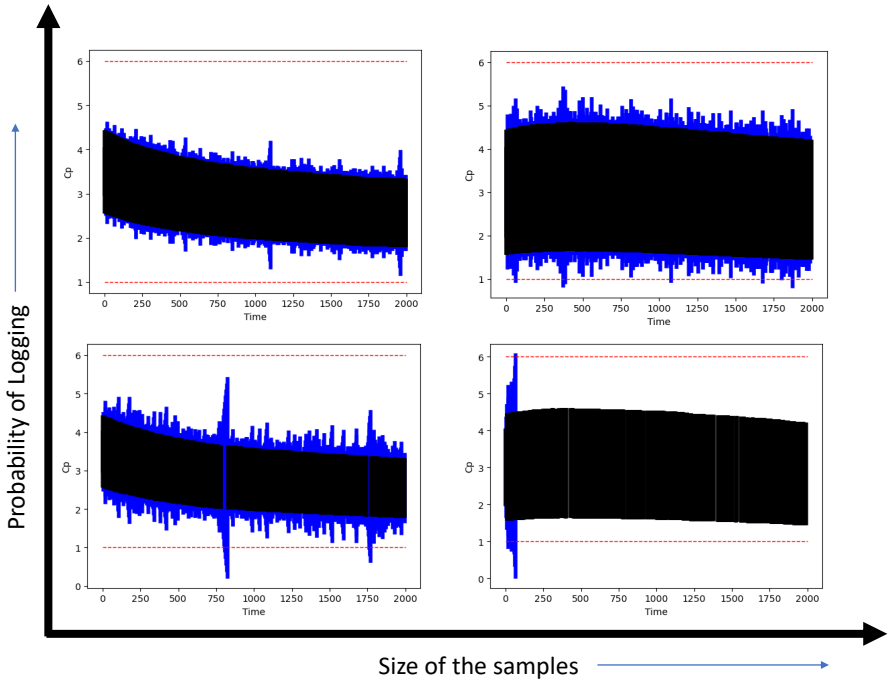


Figure 3.6: *Offline monitoring*: We plot the change in concentration level of c_p with time. The volume of the samples increase from left to right, and the probability of logging increases from bottom to top. The blue regions are the reachable sets showing the over-approximate reachable sets as computed by the offline monitoring, the black regions are the samples from the log given to the offline monitoring algorithm, and the red dotted line represents safe distance level. Note that although the top-row plots and the bottom left plots' reachable sets seem to intersect with the red line (unsafe set), the refinement module infers them to be *unreachable*, therefore concluding the system behavior as *safe*—unlike the bottom-right plot.

.

4. For the same execution, how do the outcome (in terms of verdict on safety by the monitoring algorithms) and the efficiency (in terms of number of samples needed) of the offline and online monitoring algorithms compare?

3.5.0.1 First benchmark: Anesthesia

We first demonstrate our approach on an automated anesthesia delivery model [10]. The anesthetic drug considered in this model is propofol. Such safety critical systems are extremely important to be verified formally before they are deployed, as under or overdose of the anesthetic drug can be fatal to the patient.

Model: The model as in [10] has two components:

1. Pharmacokinetics (PK): models the change in concentration of the drug as the body metabolizes it.

2. Pharmacodynamics (PD): models the effect of drug on the body.

The PK component is further divided into three compartments:

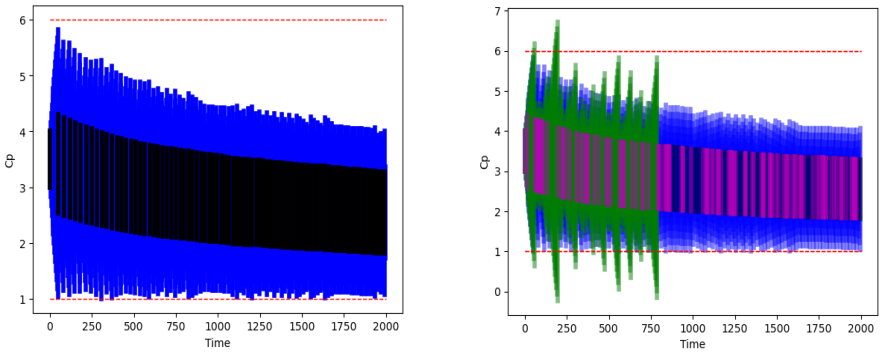


Figure 3.7: *Online monitoring*: We plot the change in concentration level of c_p with time. The blue regions are the reachable sets showing the over-approximate reachable sets as computed by the online monitoring, the black regions are the samples generated when the logging system was triggered by the online monitoring algorithm, and the red dotted line represents safe concentration levels. *Left*: We apply our online monitoring to the anesthesia model. *Right*: We compare our online and offline algorithms. The green regions are the reachable sets showing the over-approximate reachable sets between two consecutive samples from the offline logs, the magenta regions are the offline logs, given as an input to the offline monitoring algorithm, generated by the logging system, and the red dotted line represents safe concentration levels. The blue regions are the reachable sets showing the over-approximate reachable sets as computed by the online monitoring, the black regions are the samples generated when the logging system was triggered by the online monitoring algorithm, and the red dotted line represents safe concentration levels.

.

1. first peripheral compartment c_1,
2. second peripheral compartment c_2,
3. plasma compartment c_p.

The PD component has one compartment, called c_e. The set of state variables of this system is $\begin{pmatrix} c_p & c_1 & c_2 & c_e \end{pmatrix}^\top$. The input to the system is the infusion rate of the drug (propofol) u. The complete state-space model of this system in given in [10, Equation 5].

Model parameters: The evolution of states—c_p, c_1, c_2—is dependent on several parameters, such as: the weight of the patient (*weight*), the first order rate constants between the compartments k_{10}, k_{12}, k_{13}, k_{21} and k_{31}. The evolution of the state c_e is dependent on the parameter k_d, the rate constant between plasma and effect site.

Safety: The system is considered safe if the following concentration levels are maintained at all time steps: $c_p \in [1,6]$, $c_1 \in [1,10]$, $c_2 \in [1,10]$, $c_e \in [1,8]$.

In this case study, we focus our attention on the effect of perturbation, in the weight of the patient (*weight*), on the concentration level of plasma compartment c_p. We assume that the weight of the patient has an additive perturbation of ± 0.8 kg in this case study—at each time step, the weight of the patient is

weight $+ \delta_w$, $\delta_w \in [0, 0.8]$. With perturbation in the weight, we want to infer safety of this system using monitoring.

Clearly, monitoring of this system *vis-à-vis* safety is crucial. It is not practical for a busy human doctor or a practitioner to monitor each patient continuously at all time steps—monitoring, either offline or online, provides them an efficient way to save their time and treat their patients without compromising their safety.

We now answer questions (1)-(4), using Figs. 3.6 and 3.7. In Fig. 3.6:

1. the plots in the bottom row have logging probability of 20%, and the plots in top row have a logging probability of 40%;

2. the plots in left column and the right column have been simulated with an initial set of $\big[\,[3{,}4]\ [3{,}4]\ [4{,}5]\ [3{,}4]\,\big]^\top$, $u \in [2,5]$ and $\big[\,[2{,}4]\ [3{,}6]\ [3{,}6]\ [2{,}4]\,\big]^\top$, $u \in [2,10]$ respectively.

That is, the volume of the samples increases from left to right. In Fig. 3.7, we simulated the trajectory with an initial set $\big[\,[3{,}4]\ [3{,}4]\ [4{,}5]\ [3{,}4]\,\big]^\top$, $u \in [2,5]$.

Following are the answers to questions (1)-(4) from Figs. 3.6 and 3.7:

Answer to Question 1. We answer this question by comparing two sets figures in the left column and the right column of Fig. 3.6. *For the left column, i.e. with smaller sample size*: the bottom-left plot took 51.40 s and concluded the system to be safe. The analysis in this plot invoked the refinement module of the offline algorithm. But increasing the probability of logging, *i.e.* more number of samples, as in the top-left plot, resulted in not invoking the refinement module at all, thus taking 32.92 s. *For the right column, i.e. with larger sample size*: this analysis, as shown in the bottom-right column, took 1.73 s to complete, and concluded the system behavior to be unsafe. The behavior of the system, shown in top-right plot with 40% probability of logging, results in inferring the behavior of the system as safe, by invoking the refinement module several times. Overall, this analysis, as shown in the top-right plot, took 35.93 s to complete, and concluded the system behavior to be safe.

Answer to Question 2. We answer this question by comparing two sets figures in the top row and the bottom row of Fig. 3.6. *For the bottom row, i.e. with smaller logging probability*: Increasing the volume of the samples results in inferring the behavior from safe (bottom-left plot) to unsafe (bottom-right plot), as per the offline monitoring algorithm. *For the top row, i.e. with higher logging probability*: Increasing the volume of the samples results in not invoking the refinement module (top-left plot) to invoking the refinement module several times (top-right plot), as per the offline monitoring algorithm.

Answer to Question 4 The result is given in Fig. 3.7 (left). Using our online algorithm, we were able to prove safety of the system in 109.04 s. The online algorithm triggered the logging system to generate samples for 83 time steps—this is less than 5% of total time steps. We observe, as shown in Fig. 3.7 (left), that the logging system is triggered more when the trajectory is closer to the unsafe region.

3.5.0.2 Second benchmark: Adaptive Cruise Control

We now apply the offline and online monitoring algorithms to an adaptive cruise control (ACC) model [9].

An adaptive cruise control (ACC) behaves like an ordinary cruise control when there is no car in the sight of its sensor, and when there is a car in its sight, it maintains a safe distance.

An adaptive cruise control behaves like an ordinary cruise control when there is no car in the sight of its sensor, and when there is a car in its sight, it maintains a safe distance.

Model The model as in [9] has the following state variables:

1. velocity of the vehicle v,
2. distance between the two vehicles h, and
3. velocity of the lead vehicle v_L.

The state space of the system is given in [9, Equation 3]. The set of state variables of this system is $\begin{pmatrix} v & h & v_L \end{pmatrix}^\top$.

Model parameters. The model is dependent on two parameters:

1. acceleration of the lead vehicle a_L, and
2. breaking force and torque applied to the wheels as a lumped net force F.

Note that the model is dependent on acceleration of the vehicle a_L, which is very hard to accurately measure due to sensor uncertainties. Similarly the torque F applied to the wheels is also dependent of the coefficient of friction of the ground. To reflect such uncertainties, we consider $a_L \in [-0.9, 0.6]$ and $F \in [-0.6, 2.46]$.

Safety. The system is considered safe if the distance between vehicles $h > 0.5$.

Consider an event of a car crash, where the log stored by the car before the crash, is the only data available to analyze the crash; such an analysis might benefit police, insurance companies, vehicle manufacturers,

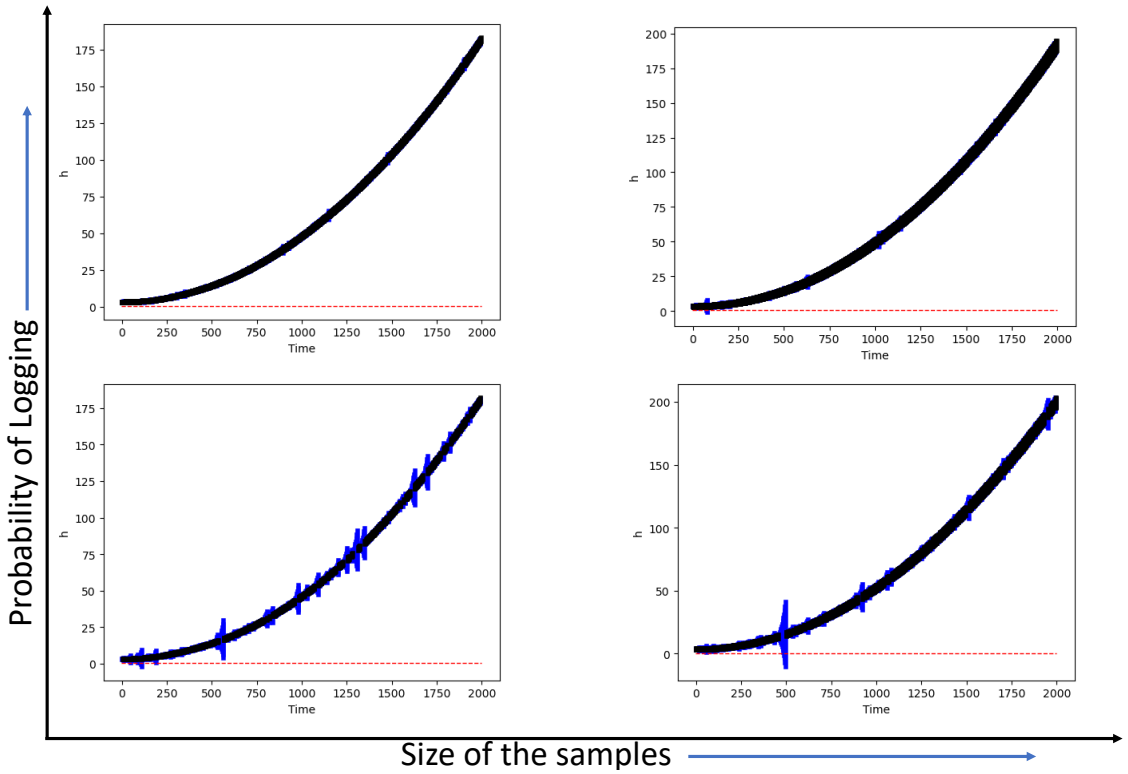


Figure 3.8: *Offline monitoring (ACC)*: We plot the change in distance h between the vehicles with time. The volume of the samples increase from left to right, and the probability of logging increases from bottom to top. The color coding is same as Fig. 3.6.

etc. Using our offline algorithm one can figure out if the car might have shown unsafe behavior or not. Similarly, consider a vehicle on a highway with a lead vehicle in its sight. The ACC in such a case needs to continuously read sensor values to track several parameters, such as acceleration of the lead vehicle, braking force, etc.—this results in wastage of energy. Using our online monitoring algorithm, the car reads sensor values only when there is a potential unsafe behavior. This intermittent behavior will result in saving energy without compromising safety of the system.

With the aforementioned reasons for applying our offline and online monitoring, we apply our algorithms on the ACC model and answer questions (1)-(4). We think that the answers to these questions will help the car designers to design efficient ACC models without compromising safety. Again, the logs for this second case study are uncertain "only" in the "valuation" dimension: that is, they are fixed timestamps uncertain logs.

Next, we answer questions (1)-(4), using Figs. 3.8 and 3.9. In Fig. 3.8:

1. the plots in the bottom row have logging probability of 20%, and the plots in top row have a logging probability of 40%;

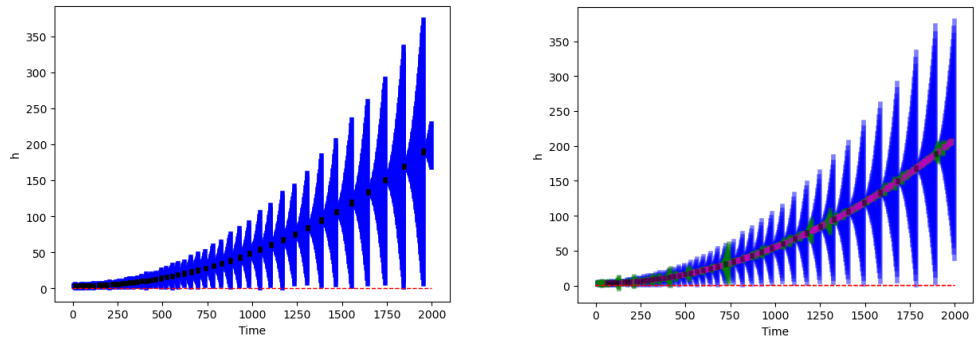


Figure 3.9: *Online monitoring (ACC)*: We plot the change in distance between two vehicle h with time. The color coding is same as Fig. 3.7. *Left*: We apply our online monitoring to the ACC model. *Right*: We compare our online and offline algorithms.

.

2. the plots in left column and the right column have been simulated with an initial set of $\big[[15{,}15.01]\ [3{,}3.03]\ [14.9{,}15]\big]^\top$ and $\big[[15{,}15.1]\ [3{,}3.5]\ [14.9{,}15.1]\big]^\top$ respectively.

In Fig. 3.7, we simulated the trajectory with an initial set $\big[[15{,}15.01]\ [3{,}3.03]\ [14.9{,}15]\big]^\top$, $u \in [2,5]$.

Answer to Question 1. We answer this question by comparing two sets figures in the left column and the right column of Fig. 3.8. *For the left column, i.e. with smaller sample size*: the bottom-left plot took 19.08 s and concluded the system to be safe. This analysis in this plot invoked the refinement module of the offline algorithm. But increasing the probability of logging, *i.e.* more number of samples, as in the top-left plot, resulted in not invoking the refinement module at all, thus taking 16.5 s. *For the right column, i.e. with larger sample size*: The analysis is similar to that of the left column. The bottom-right plot invoked the refinement module several times, thus taking 20.84 s, while the top-right plot took 17.5 s, as it invoked the refinement module a smaller number of times.

Answer to Question 2. We answer this question by comparing two sets figures in the top row and the bottom row of Fig. 3.8. *For the bottom row, i.e. with smaller logging probability*: Comparing the bottom-left and bottom-right shows that increasing sample volume results in invoking the refinement module more frequently. A very similar behavior is seen by comparing the top row (*i.e.* with higher logging probability).

Answer to Question 3. Using our online algorithm, we were able to prove safety of the system in 104.58 s. The online algorithm triggered the logging system to generate samples for 53 time steps—this is less than 3% of total time steps. This is shown in Fig. 3.9 (left).

Answer to Question 4. We compare our offline and online algorithm, for 2000 time steps, on the same trajectory. The result is given in Fig. 3.9 (right). Note that, using our online algorithm, we were able to prove

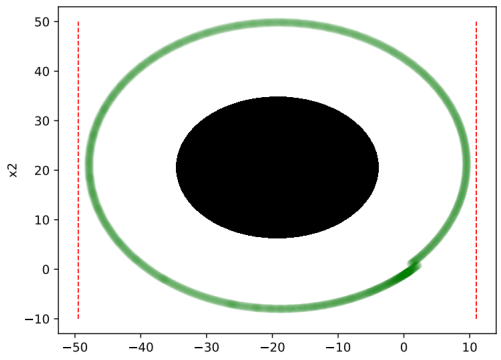


Figure 3.10: *Planned Behavior of the Aircraft*: The axis of the plot denotes the position of the aircraft in (x, y) plane (with a fixed altitude). The ideal (planned) path of the aircraft, orbiting the object (black), is shown in green. The red dashed lines indicate the safety constraint of the aircraft—these lines must not be crossed at any time step.

safety of the system in 124.46 s. The online algorithm triggered the logging system to generate samples only 50 times. In contrast, the offline algorithm, with a log size of 281 (14% logging probability) took 28.54 s to infer that the system is safe.

3.5.0.3 Third benchmark: Aircraft Orbiting

Next, we apply our offline algorithm to an aircraft orbiting case study. Unlike the former two case studies, we consider in this third benchmark not only uncertain valuations, but also uncertain timestamps. That is, logs for this benchmark are uncertain logs. In this case study, an aircraft is orbiting an object to gather information about the object—depicted in Fig. 3.10.

The aircraft orbits the object by controlling its angular velocity. Since the aircraft operates in an uncertain environment, the angular velocity of the aircraft can have added noise. That is, the model of the aircraft used in this case study can have uncertainties in its angular velocity. The axis of Fig. 3.10 denotes the position of the aircraft in (x, y) plane (we assume a fixed altitude). The ideal (planned) path of the aircraft, orbiting the object (black), is shown in green. The aircraft must always stay sufficiently close to the object for the data collected about it to be considered reliable. One such bound is shown in the plot with red dashed lines—the aircraft must not cross these lines at any time step. The aircraft transmits its positional data (the x, y values) to the local station at aperiodic intervals. Since the data is transmitted over a shared, long-distance network, a delay is experienced when the data reaches the local station. In other words, the timestamps of the log (behavioral data) can have added noise. Additionally, the behavioral data can also have added noise due to

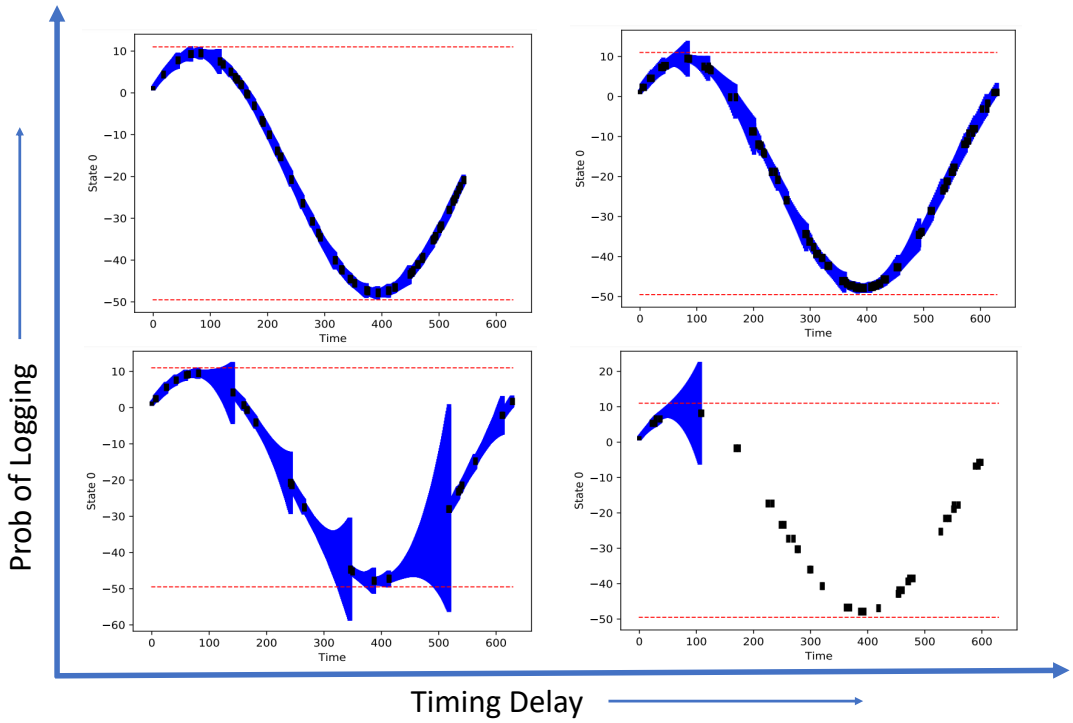


Figure 3.11: *Offline monitoring (Aircraft Orbiting)*: We plot the position of the aircraft, along x axis, with time. The timing delay of the samples increases from left to right, and the probability of logging increases from bottom to top. The color coding is same as Fig. 3.6.

sensor uncertainties. Therefore, this case study becomes an ideal candidate to demonstrate the applicability of our offline monitoring approach, with added noise in timestamps

Model The model as in [11] has the following state variables:

1. position of the aircraft in two dimensional plane (x_1, x_2),
2. and velocity (d_1, d_2).

The state space of the system is given in [11]. The set of state variables of this system is $\begin{bmatrix} x_1 & x_2 & d_1 & d_2 \end{bmatrix}^\top$.

Model parameters. The model is dependent on the angular velocity ω of the aircraft. We assume $\omega \in 1.5 \pm 1\%$.

Safety. The behavior of the aircraft is considered safe, under the presence of model uncertainties, if $x \in [-49.5, 11]$.

All the plots in Fig. 3.11 have been generated with an initial set of $\begin{bmatrix} [1.1,1.11] & [1.1,1.11] & [20,20.1] & [20,20.1] \end{bmatrix}^\top$. In Fig. 3.11, the plots in the bottom and top rows have a logging probability of 5% and 10% respectively, and the plots in the left column and the right column have a timing delay of 2 units and 10 units respectively.

We now answer Questions (1) and (5) using Fig. 3.11.

Answer to Question 1. The observations *vis-à-vis* this question are very similar to previous two case studies. We answer this question by comparing two sets figures in the left column and the right column of Fig. 3.11. *For the left column, i.e. with smaller timing delay*: the bottom-left plot took 111.6 s and concluded the system to be safe. The analysis in this plot invoked the refinement module of the offline algorithm. But increasing the probability of logging, *i.e.* more number of samples, as in the top-left plot, resulted in not invoking the refinement module at all, thus taking 31.61 s. *For the right column, i.e. with larger timing delay*: this analysis, as shown in the bottom-right column, took 2.33 s to complete, and concluded the system behavior to be unsafe. The behavior of the system, shown in top-right plot with 10% probability of logging, results in inferring the behavior of the system as safe, by invoking the refinement module. Overall, this analysis, as shown in the top-right plot, took 48.39 s to complete, and concluded the system behavior to be safe.

Answer to Question 5. We answer this question by comparing two sets figures in the top row and the bottom row of Fig. 3.11. *For the bottom row, i.e. with smaller logging probability*: Increasing the timing delay of the samples results in inferring the behavior from safe (bottom-left plot) to unsafe (bottom-right plot), as per the offline monitoring algorithm. *For the top row, i.e. with higher logging probability*: Increasing the timing delay the samples results in not invoking the refinement module (top-left plot) to invoking the refinement module several times (top-right plot), as per the offline monitoring algorithm.

CHAPTER 4: SCHEDULING IN AUTONOMOUS SYSTEMS—HANDLING WORKLOADS SAFELY

Design and verification of autonomous system controllers is becoming increasingly challenging due to the heterogeneous nature of the implementation platforms. Further, autonomous systems are required to run several processes together on such platforms. Owing to all such difficulties, it is often hard or impossible to schedule all the processes in a timely manner—as a result, several processes might miss its deadlines—even though the controller has been designed with the requirement that all the deadlines are always met. However, it is often the case that the safety properties of the controllers are not violated even if the deadlines are not always met. This observation lets us develop a flexible design paradigm that is not overly conservative (*i.e.*, the deadline is always met). In this chapter, we propose a framework to identify deadline miss patterns that do not violate safety. Using such artifacts, one can propose correct-by-construction synthesis algorithm that can generate such schedules.

About the chapter. The content of this chapter is taken from [1, 62, 63, 64, 65, 66].

4.1 Effect of Timing Uncertainties on Autonomous System Controllers

Providing verifiable assurances for autonomous systems is a challenge that has attracted considerable scientific interest [67, 68, 69]. Traditionally, this involved designing suitable *feedback control loops* and formally verifying their correctness. An emerging challenge in providing such assurances for current generation autonomous systems is providing timing guarantees of the increasingly complex on-board hardware platforms utilizing machine learning (ML) components. Presently, these consist of multiple multicore processors and hardware accelerators like GPUs and FPGAs. As a result, the timing behavior of control software [70] running on them can be highly variable because of complex interference patterns between heterogeneous components. Analyzing the timing behavior of such software consists of two main steps:

1. Determining the worst-case execution time (WCET) of the code [71]
2. Using schedulability analysis to validate the timing constraints (or deadlines) assuming which the feedback controllers have been designed

Unfortunately, there is now widespread consensus that on modern hardware platforms, safe WCET estimates cannot be guaranteed without excessive pessimism [72, 73].

This situation is further aggravated by ML components for sensor (camera, radar, lidar) processing that incur content-dependent processing times. Hence, unless very pessimistic WCET bounds are acceptable—which makes designs highly over-provisioned and impractical—a certain non-determinism in the timing behavior of software is unavoidable. Further, even with pessimistic WCET bounds, modern-day automobiles are required to run several complex processes. Conventional approaches, which require strict adherence to deadlines, have become increasingly impractical or require the adoption of costly high-performance processors which can meet such hefty requirements. The method proposed in this paper, however, proposes a strategy that can meet the safety and the performance requirements while efficiently using the limited bandwidth of a processor by strategically missing some deadlines. These techniques can be further be used to design schedulers of modern-day automobiles that can efficiently utilize the limited processing capacity of a processor by strategically allowing certain processes to occasionally miss their deadlines. Further, usage of statistical approaches help us in handling model complexity of the processes running on modern-day automobiles.

This raises the question: *"What performance guarantees can be provided for feedback control loops subjected to certain non-deterministic timing behavior?"* There are various incarnations of this question and the one most widely studied, particularly within *networked control systems*, asks how to ensure *stability* in the presence of uncertainties in network behavior such as delays and dropped packets [74, 75, 76, 77, 78]. Instead of a *qualitative* property like stability, *i*n this chapter we ask whether *quantitative* properties, such as safety specification over a bounded time horizon, hold in the presence of timing uncertainties.

F1Tenth Example. As an example, consider Fig. 4.1. It shows the trajectory of a lane-following controller for an F1Tenth [79] model car. The car's steering angle and velocity are computed by a feedback controller, designed for the car to follow a predetermined path. Running the controller as designed, with *no* timing uncertainty, results in the *nominal* trajectory shown in black. Around this trajectory, there is a *safety envelope* shown in light blue, representing a safe space for the car to occupy without hitting any obstacles. Due to timing uncertainties in the implementation platform, the software task that computes the control inputs can miss the deadline imposed by the scheduler, resulting in deviation from the nominal trajectory. 100 such trajectories where the control task missed its deadline are shown in the figure—the green trajectories are

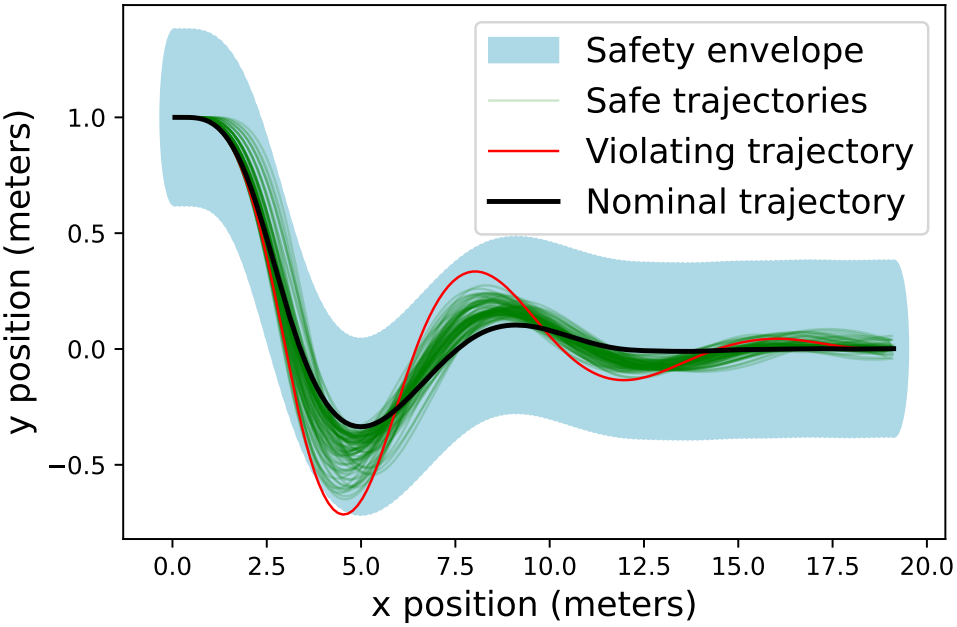


Figure 4.1: Deviation in the path of an F1Tenth car due to timing uncertainty.

safe, remaining within the safety envelope for the entire time horizon. However, the trajectory shown in red deviates too far from the nominal trajectory, briefly leaving the safety envelope near $x = 4$, potentially resulting in a collision with an obstacle. This illustrates the importance of bounding deviations of control systems if timing variations in the control software may occur.

Contributions. Given an ideal system behavior (when the control task always meets its deadline), and a pattern of deadline hits and misses, **we want to estimate the maximum possible deviation from the ideal behavior in the presence of the allowed patterns of deadline misses** over a time horizon of length H. This

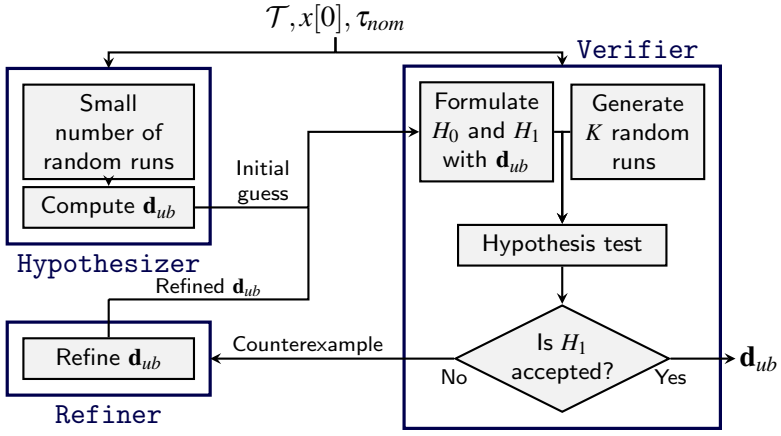


Figure 4.2: Proposed statistical hypothesis testing approach.

involves *reachability analysis* of the states visited by the trajectories of the closed-loop system in the time interval $[0, H]$. This is however not a scalable problem and **the main contribution** is a *statistical hypothesis testing* (SHT) framework, as shown in Fig. 4.2. In particular, we use the *Jeffreys's Bayes factor test* [80, 49] to solve our problem.

In this setup, timing behaviors of interest are specified as patterns of *hits* (the deadline is met) and *misses* (deadline is not met) and are assumed to constitute a *regular language* over the alphabet $\{\text{hit}, \text{miss}\}$. We further assume a uniform distribution over these strings of length H (the time horizon of interest). This enables us to implement an efficient sampling method based on the *Recursive RGA* algorithm [81]. However, this method is applicable to other types of languages and distributions as well, provided the runs of the system can be efficiently sampled. In the current setup, we are given a set of initial states of the system, a mathematical model of its (discrete time) dynamics, and a regular language L of strings of length H over the alphabet $\{\text{hit}, \text{miss}\}$. Our goal is to estimate an upper bound $\mathbf{d}_{ub}$ on the deviation of the trajectory induced by any string in L from the nominal trajectory induced by the string consisting of only hits (the ideal timing behavior).

Lastly, in this chapter, we investigate an approach to synthesize a scheduler for control tasks that miss some deadlines without compromising any of the safety requirements. Given that the number of possible schedules would increase combinatorially with the number of tasks involved, our scheduler synthesis uses an efficient automata representation to search for the appropriate schedule. We incorporate statistical verification techniques to construct this automaton and accelerate the search process. Statistical verification is advantageous compared to deterministic verification in the synthesis process in two ways: first, it enables us to synthesize schedules that would not be possible otherwise, and second, it drastically reduces the time taken to synthesize such a schedule.

4.1.1 System Modeling

We study the state feedback control of discrete time-invariant linear dynamical systems of the form:

$$x[t+1] = Ax[t] + Bu[t], \tag{4.1}$$

where $A \in \mathbb{R}^{n \times n}$, and $B \in \mathbb{R}^{n \times p}$. The control input u is computed by a periodic real-time task running on a processor, and is assumed to be of the form:

$$u[t] = Kx[t-1], \tag{4.2}$$

where $K \in \mathbb{R}^{p \times n}$. Here, Eq. (4.1) represents the *plant*—the system whose evolution needs to be controlled. The *controller*, given in Eq. (4.2), issues control inputs to the plant to control the evolution of the system as desired by the user. Internally, the plant receives the control inputs at some discrete time steps, and controls the evolution of the system based on that input—this part of the system is often termed as the *actuator*. The plant (Eq. (4.1)) and the controller (Eq. (4.2)) can alternately be represented using an augmented state space [82] as $z[t] = \begin{bmatrix} x[t]^T & u_a[t-1]^T \end{bmatrix}^T$, giving the following model:

$$z[t+1] = \begin{bmatrix} A & B \\ K_x & K_u \end{bmatrix} z[t] \tag{4.3}$$

Here, we denote the two blocks of the feedback gain matrix K providing feedback from each of the vectors x and u as $K_x \in \mathbb{R}^{p \times n}$ and $K_u \in \mathbb{R}^{p \times p}$, respectively. This augmented form permits standard controller design techniques such as *linear-quadratic regulator* (LQR). Further, it allows the plant and controller to be represented as a single *dynamics matrix*.

In order to model the system behavior under a sequence of deadline hits and misses, we use standard techniques, similar to those in [8]. In this model, the logical execution time (LET) paradigm is followed, *i.e.*, a sample of the system state at step $t-1$ is used to compute the control input at time t. A software job is released when $x[t-1]$ is read, and has its deadline as when $x[t]$ is to be read. If the job completes on time, the control input is computed. If the job misses its deadline, several different actions can be taken—described below—both for generating the missing control input and for handling the task that has missed its deadline [8].

We specify the behavior of the scheduler as an automaton that maps the allowed patterns of hits and misses to the accompanying plant dynamics and control inputs.

Definition 27. *A* transducer automaton $\mathcal{T}$ *is a tuple* $\langle L, \mathcal{A}, T, \mu, \ell_{int} \rangle$*, defined as follows:*

- $L = \{\ell_1, \ell_2, \ldots, \ell_m\}$*: Set of automaton states.*
- $\mathcal{A}$*: Set of scheduler actions.* $\mathcal{A} = \{hit, miss\}$

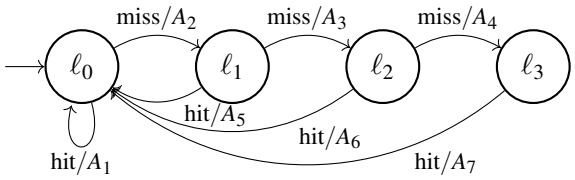


Figure 4.3: Transducer automaton capturing 3 maximum consecutive misses.

- *T: The transition function, where $T : L \times \mathcal{A} \to L$. Let $\mathbb{T} = \left\{ T(\ell_i, a) = \ell_j \middle| \ell_i, \ell_j \in L, a \in \mathcal{A} \right\}$ denote the set of all transitions of the automaton.*

- *μ: Associates a dynamics matrix with a transition. Formally, $\mu : \mathbb{T} \to \mathbb{R}^{n \times n}$, where n is the dimension of the system under consideration.*

- *$\ell_{int} \in L$: Initial state of the automaton.*

We sometimes refer to transducer automata as Deterministic Finite Automata (DFA). An example of a DFA, capturing all possible behaviors with at most 3 consecutive deadline misses, is shown in Fig. 4.3. The labels are of the form a_i/A_i, where $a_i \in \mathcal{A}$ and A_i is the dynamics matrix associated to the label using the function μ.

Several policies for handling deadline misses, both in terms of the control input to apply and how to treat the job that has missed its deadline have been proposed in [8]. Many control input strategies can be devised, but any such strategy must be simple enough to be implemented on an actuator. The two we consider here are *Zero*, where a control input of 0 is applied, and *Hold*, where the current control input is used again until a new one can be computed. As for the handling of jobs that have missed their deadlines, there are again multiple ways to handle them. Here, we consider the *Kill* strategy, where the job is killed as soon as its deadline is passed, and the *Skip-Next* strategy, where a job is allowed to run to completion past its deadline, but no new job instance may be released until this happens. By combining a strategy for control input and real-time job scheduling, we arrive at a single *deadline miss strategy*. The resulting combination of policies, named *Zero&Kill*, *Hold&Kill*, *Zero&Skip-Next*, and *Hold&Skip-Next* in [8], will result in different closed loop dynamics under deadline misses. However the nominal behaviors will be identical.

Let the plant states $x[t]$ at time t be subsets of the metric space $(\mathbb{R}^n, dis)$, where *dis* is a metric on $\mathbb{R}^n$. We do not impose any assumption on *dis* (but use the Euclidean distance in this paper). We note that this metric applies only to the *plant state*, not the augmented state vector used by a transducer automaton. Given $\mathcal{T}$, a

possible behavior of the system is defined as a *run* consisting of an alternating sequence of locations and actions:

$$\tau = \{\ell_1, a_1, \ell_2, \ldots, a_{H-1}, \ell_H\} \tag{4.4}$$

where $\ell_1 = \ell_{\text{int}}$, $a_i \in \mathcal{A}$, and H is the time bound. Let the set of all possible runs be $\bar{\tau}$.

Next, the evolution of a run $\tau = \{\ell_1, a_1, \ell_2, \ldots, a_{H-1}, \ell_H\}$, with an initial set $x[0] \subset \mathbb{R}^n$ is denoted as $evol(\tau)$, given by

$$evol(\tau) = \big\{x[0], x[1] = A_1 x[0], x[2] = A_2 x[1], \ldots, x[H] = A_H x[H-1]\big\}. \tag{4.5}$$

Note that since the evolution starts from an initial set of states ($x[0] \subset \mathbb{R}^n$), the states reached at every time step t ($x[t]$) is also a set.

Here, $A_t = \mu(a_t)$ and $x[t]$ are the plant states reached at time step t. Given evolution of a run $evol(\tau)$, let

$$evol(\tau)[t] = x[t], \qquad \text{for } 1 \leq t \leq H.$$

Note that we must consider distance between sets, not points, because we are now working with the sequence of a set of states (trajectories)—such a concept of distance is Hausdorff. It is important to note that the representation of the sets has a significant impact on the computational complexity of computing the Hausdorff distance—from polytime to NP.

We now define the distance between two sets $S, R \subset \mathbb{R}^n$ using the standard Hausdorff distance, which we denote as

$$\Delta(S,R) = \max\Big\{\sup_{s\in S}\inf_{r\in R} dis(s,r),\ \sup_{r\in R}\inf_{s\in S} dis(s,r)\Big\}.$$

Given two runs $\tau_1, \tau_2 \in \bar{\tau}$, we define deviation between the two runs as the maximum Hausdorff distance between the evolution of the two runs. Formally:

Definition 28 (Deviation)**.** *The deviation between two runs τ_1 and τ_2 is given by:*

$$dev(\tau_1, \tau_2) = \max_{1\leq t\leq H} \Big\{\Delta\big(evol(\tau_1)[t], evol(\tau_2)[t]\big)\Big\}. \tag{4.6}$$

Finally, as mentioned in the introduction, we assume a probability distribution $\mathcal{D}$ over the set of runs $\bar{\tau}$. Accordingly, by a random run we shall mean a run drawn from $\bar{\tau}$ according to $\mathcal{D}$. In the present setting, $\mathcal{D}$ is the uniform distribution. However, our analysis method is applicable to any distribution $\mathcal{D}$, provided one can effectively draw samples from $\mathcal{D}$.

4.2 The Problem Statement and a Deterministic Approach

The analysis problem we wish to solve is as follows.

Problem 1. *Given a transducer automaton $\mathcal{T}$, an initial set of plant states $x[0] \subset \mathbb{R}^n$, and a nominal run $\tau_{nom} \in \bar{\tau}$, compute the maximum deviation $\mathbf{d}_{max}$, where:*

$$\mathbf{d}_{max} = \max\{dev(\tau, \tau_{nom}) | \tau \in \bar{\tau}\}. \tag{4.7}$$

Assuming a time bound of H, where at each step a deadline can either be met or missed, computing the exact maximum deviation $\mathbf{d}_{max}$ will require computing the deviation of 2^H trajectories from the given nominal trajectory. This becomes intractable for realistic values of H, so more efficient methods must be used to instead *approximate* the value of $\mathbf{d}_{max}$.

To this end, there are many reachability algorithms for linear dynamical systems that can be used to **safely overapproximate** the maximum deviation. We propose one such approach here, which we call `RS` (*i.e.*, **r**eachable **s**et), as a baseline against which we will compare our statistical hypothesis testing approach in our experiments.

The `RS` method begins by fixing a small number of time steps m. Given an axis-aligned n-dimensional interval $x[0]$ as an initial set, the algorithm proceeds iteratively, computing the reachable sets for each successive span of m sampling periods. For the first iteration, all trajectories of length m starting from the corners of the initial set $x[0]$ are computed. We store the minimum bounding box of all such trajectories at each time step, yielding our first m over-approximated reachable sets. At the end of each iteration, we group the runs by their final locations in the automaton, and compute a bounding box for each location.

Using these boxes and their corresponding locations as initial conditions, we compute the over-approximated reachable sets for the following m time steps. This procedure is iterated as many times as required to span the time horizon H (*i.e.*, $\lceil H/m \rceil$ iterations). While the runtime of the `RS` algorithm is exponential in the parameter m, it is linear in the number of iterations. Thus by running only a small number of time steps in

each iteration, we can compute a *sound* over-approximation of the reachable sets for large time horizons efficiently. From these reachable sets, it is straightforward to compute a safe upper bound $\mathbf{d}_{ub} \geq \mathbf{d}_{max}$.

Unfortunately, this simple reachability-based approach often produces bounds on the maximum deviation that are either quite pessimistic, or require a large amount of execution time (due to a large number of steps per iteration m). Thus, in the next section, we propose the main contribution of this chapter, a method to *estimate* the value of $\mathbf{d}_{max}$ based on statistical hypothesis testing.

Proposed SHT framework. The statistical hypothesis testing framework, proposed in this chapter, formulates two hypotheses, namely null hypothesis H_0 and alternative hypothesis H_1, to test if a given $\mathbf{d}_{ub}$ is an upper bound for the maximum deviation. The null hypothesis H_0 will assert that with *at most* probability c, a randomly chosen trajectory will have a deviation bounded by $\mathbf{d}_{ub}$. The alternative hypothesis H_1 will assert that with *at least* probability c, a randomly chosen run will have a deviation that is bounded by $\mathbf{d}_{ub}$. This is illustrated in Fig. 4.2. We then use the Jeffreys's Bayes factor test to decide between these two hypotheses [80]. An important consequence of our test is, when the samples we have drawn do not support the alternative hypothesis, they will contain a *counterexample* with a deviation that exceeds the current value of $\mathbf{d}_{ub}$. We leverage this counterexample to generate new hypotheses, for the next iteration, based on a new and larger $\mathbf{d}_{ub}$. In this sense, our method is driven by a *counterexample guided refinement strategy* to eventually accept the alternative hypothesis (see Fig. 4.2). In a statistical hypothesis frameworks, in addition to the probabilistic guarantee c, we can also bound the so called type I and type II error rates. The type I error rate is defined as the probability of inferring the alternative hypothesis when in fact the null hypothesis holds. Similarly, the type II error rate is defined as the probability of inferring the null hypothesis when in fact the alternate hypothesis holds. We choose the relevant parameters such that the type I error is kept significantly low (as desired), while type II error [80] is not relevant in our setting.

Handling Initial Set of States and Environmental Disturbances. When a system starts its evolution form a single initial state, the resulting trajectory is a sequence of states (or *points*) in time, as was the case in the F1Tenth example. In [1], the estimation of deviation is computed when the system starts from a given initial state. However, one might want to conduct the analysis for a set of states either because the initial state is unknown or because the user would like to do a similar analysis for a large collection of initial states. Moreover, for all practical purposes, it is impossible to locate the state from where evolution of the system (such as a robot) starts its execution from. In such cases, one must consider an initial set of states to perform

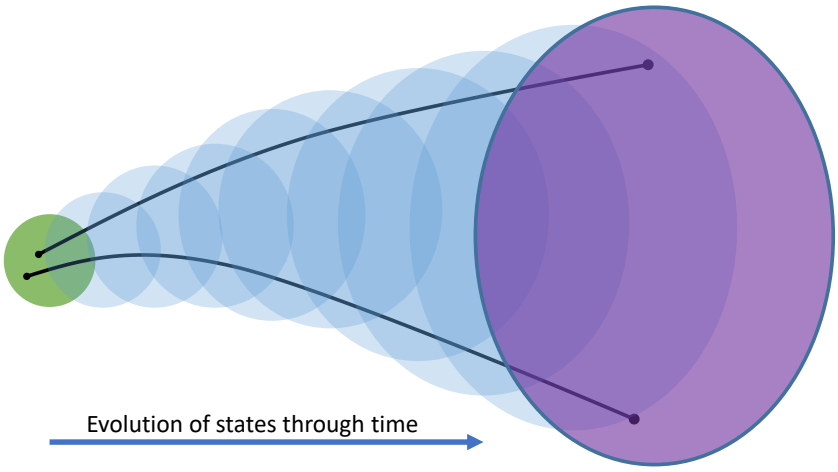


Figure 4.4: Evolution of states through time from an initial set of states. The figure depicts all the possible set of states that are reachable when the system starts from a given initial set of states. The set of initial states is given in green, blue represents the set of intermediate states, and purple represents the set of states reached at time t.

an analysis, as a single initial state is often useless in practice. We call the collection of all trajectories starting from the (possibly infinite) initial set as *pipes*. We consider the example given in Fig. 4.4 that demonstrates the evolution of the system from initial set of states (marked in green). The set of states reached at time t is given in purple, and intermediate set of states is given in blue. The black lines in the figure demonstrates two random trajectories of the system starting from two random states from the initial set of states. In this chapter, we extend our previous approach [1] for computing upper bound from a specific initial state to an initial set. This generalized analysis requires two main improvements. First, one has to choose an appropriate representation for the reachable set. Second, one has to compute, not just distance between trajectories, but rather compute distances between two (infinite) sets of trajectories. Certain representations can be useful to compute the distance between sets, while computing evolution of the system being hard (and vice versa). Thus, a representation must be chosen trading-off various factors [83, 84]. For the clarity of presentation, we use trajectory to refer to pipes in this chapter hereon.

A system might also encounter environmental disturbances in its behavior at each time step in addition to uncertainty in its initial state. That is, the system may encounter uncertainty due to environmental disturbances at every time step resulting in a new set of possible system states. Although feedback controllers are frequently used to manage these uncertainties, for computing tight estimates of deviation from nominal trajectory, one has to account for such uncertainties. In this chapter, we further discuss how one can adapt the proposed method to compute deviations under the presence of such uncertainties in the behavior of the system.

Salient features of the proposed method. To conclude this section, we note that the intrinsic nature of hypothesis testing ensures that the number of samples to be drawn depends only on the required strength of the test and not on factors like the length of time horizon H, or the scheduling policy that results in deadline hits and misses. Furthermore, while we characterize *timing uncertainty* using a language of deadline hit/miss patterns, our scheme can be extended to other fine grained types of timing uncertainty as well such as task completion times. It is worth pointing out that verifying quantitative safety properties for control systems is harder than stability, a qualitative safety property. This is because stability analysis can lean on techniques such as the existence of a *Lyapunov function* and on results from stability analysis of switched systems [77].

4.3 Quantitative Safety Analysis of Autonomous System Controllers in the Presence of Timing Uncertainties: A Statistical Approach

In this section, we present a statistical hypothesis testing based approach to solve Problem 1. Specifically, we propose a counterexample guided refinement method to compute an upper bound $\mathbf{d}_{ub}$ for $\mathbf{d}_{max}$ using statistical hypothesis testing. Consequently, our estimate $\mathbf{d}_{ub}$ will be accompanied by a statistical guarantee. We refer to $dev(\tau, \tau_{nom})$ as *deviation of the run* τ. The inputs to our algorithm that estimates $\mathbf{d}_{ub}$ are a DFA $\mathcal{T}$ that models the behavior of scheduler, the initial set of plant states $x[0]$, a time horizon H, and the nominal behavior τ_{nom}. A network of three modules as illustrated in Fig. 4.2 will constitute our algorithm. Before we describe each module in detail, we provide a brief overview of the modules constituting the main approach, as follows.

`Hypothesizer`: Using the heuristics described in Section 4.3.0.1, we guess an initial $\mathbf{d}_{ub}$. This is then sent to the `Verifier` module.

`Verifier`: This module statistically verifies—using K randomly drawn sample runs—whether the given $\mathbf{d}_{ub}$ is indeed an upper-bound with the required probability (say, ≥ 0.99). If $\mathbf{d}_{ub}$ is accepted, it is returned as our final answer. If not, $\mathbf{d}_{ub}$ is sent to the `Refiner` module. The details of the `Verifier` module are presented in Section 4.3.0.2.

`Refiner`: This module pads the deviation bound obtained from the counterexample with slack ε, it is set to be the new $\mathbf{d}_{ub}$ and sent to the `Verifier`. This will initiate the next round of hypothesis testing.

4.3.0.1 `Hypothesizer`: Guessing an upper-bound on deviation

To guess an initial upper bound, we observed that a small set of randomly chosen samples can sometimes provide a reasonably good representation of the actual distribution. Thus our initial guess consists of the following steps, with R and ε being parameters supplied by the user:

1. Let $\mathcal{S} = \{\tau_1, \tau_2, \ldots, \tau_R\}$ be a set of randomly generated runs sampled according to the given distribution over the set of strings of length H specified by the DFA.
2. Let $\mathbf{d}'_{ub} = \max_{\tau \in \mathcal{S}} \{dev(\tau, \tau_{nom})\}$.
3. Return $\mathbf{d}_{ub} = \mathbf{d}'_{ub} + \varepsilon$.

We will refer to this heuristic as $\texttt{SmallSample}(\cdot)_{[R,\varepsilon]}$.

4.3.0.2 `Verifier` and `Refiner`: Validating and refining the bound on deviation

Here we describe how the hypothesis test is carried for a given value of $\mathbf{d}_{ub}$ and a parameter $c \in (0,1)$ representing the strength (probability) with which the user wishes to assert that $\mathbf{d}_{ub}$ is an upper bound.

We use Bayesian hypothesis testing based on Jeffreys's Bayes factor [49]. Accordingly we first formulate the null and alternative hypotheses:

$$H_0 : \mathcal{P}rob\,[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}] < c \tag{4.8}$$

$$H_1 : \mathcal{P}rob\,[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}] \geq c \tag{4.9}$$

Here $\mathcal{P}rob\,[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}]$ denotes the probability that a randomly drawn sample has a deviation that does not violate the upper bound $\mathbf{d}_{ub}$. Our goal is to determine an upper bound for which the alternative hypothesis is accepted.

First we fix a sufficiently high value (say 10^5) for the *Bayes factor* B which we will soon define. Using the Bayes factor B and probability c we shall compute K, the number of samples needed to choose between the null and alternative hypotheses, whose derivation we will detail below. We next draw K samples $X = \{\tau_1, \tau_2, \ldots, \tau_K\}$ according to the distribution assumed over the set of executions. We then check if *every* member of X satisfies the upper bound $\mathbf{d}_{ub}$. If yes, we accept the alternative hypothesis and return $\mathbf{d}_{ub}$ as the estimated upper bound. If not, we send the first counterexample encountered to the `Refiner` module.

Using this counterexample, the `Refiner` module will extract a new bound estimate $\widehat{d}$ (guaranteed to exceed the current value of $\mathbf{d}_{ub}$), set it to be $\mathbf{d}_{ub}$ (after adding a padding factor ε) and send it to the `Verifier` for the next round of hypothesis testing.

We will illustrate the relationship between the Bayes factor B, confidence c, and the number of samples to be generated K. Let $X = \{\tau_1, \tau_2, \ldots, \tau_K\}$ be a set of random runs, such that the maximum deviation exhibited by each $\tau_i \in X$ is bounded by $\mathbf{d}_{ub}$. The probability that $\mathbf{d}_{ub}$ is valid, given the null hypothesis, is

$$\mathbf{Pr}\left[\forall \tau \in X : dev(\tau, \tau_{nom}) \leq \mathbf{d}_{ub} \,|\, H_0\right] = \int_0^c q^K \, dq. \tag{4.10}$$

Similarly, the probability that $\mathbf{d}_{ub}$ is valid, given the alternative hypothesis, is Similarly, the probability that $\mathbf{d}_{ub}$ is valid, given the alternative hypothesis, is

$$\mathbf{Pr}\left[\forall \tau \in X : dev(\tau, \tau_{nom}) \leq \mathbf{d}_{ub} \,|\, H_1\right] = \int_c^1 q^K \, dq. \tag{4.11}$$

The *Bayes Factor* is the ratio of the above two probabilities:

$$\frac{\mathbf{Pr}\left[\forall \tau \in X : dev(\tau, \tau_{nom}) \leq \mathbf{d}_{ub} \,|\, H_1\right]}{\mathbf{Pr}\left[\forall \tau \in X : dev(\tau, \tau_{nom}) \leq \mathbf{d}_{ub} \,|\, H_0\right]} = \frac{1 - c^{K+1}}{c^{K+1}} \tag{4.12}$$

Bayes Factor is the strength of evidence favoring alternative hypothesis over null hypothesis. In Jeffreys's Bayes Factor test, we enforce that the ratio computed in Equation 4.12 is greater than the Bayes factor B provided by the user. Thus a sufficiently high value of the Bayes Factor indicates that the evidence favors the alternative hypothesis over the null hypothesis. Given B, we can now compute the required number of samples K as:

$$\frac{1 - c^{K+1}}{c^{K+1}} > B \iff K > -\log_c(B+1) \tag{4.13}$$

We call this procedure $\texttt{Verifier}_B(H_0, H_1)$, where H_0 and H_1 are the null and alternative hypothesis, respectively. The hypothesis testing procedure coupled to a counterexample guided refinement method is listed in Algorithm 9.

Algorithm 9: Computing upper bound on the deviation as defined in Eq. (4.7)

input : A transducer automaton $\mathcal{T}$, initial set $x[0]$, nominal run τ_{nom}, time bound H ; // $x[0]$ represents initial set of states

output : Compute an upper bound $\mathbf{d}_{ub}$ for $\mathbf{d}_{max}$

/* we assume parameters R, ε, B and c are provided by the user. */

$\mathbf{d}_{\sqcap\lfloor} \leftarrow \texttt{SmallSample}(\mathcal{T}, x[0], \tau_{nom})_{[R,\varepsilon]}$; // initial guess

$H_0 \leftarrow \mathcal{P}rob\,[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}] < c$; // form H_0 using Eq. (4.8)

$H_1 \leftarrow \mathcal{P}rob\,[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}] \geq c$; // form H_1 using Eq. (4.9)

$res \leftarrow \texttt{Verifier}_B(H_0, H_1)$; // perform statistical verification

if res = *True* **then**

  return $\mathbf{d}_{ub}$;

while *True* **do**

  $\mathbf{d}_{ub} \leftarrow \texttt{Refiner}(res)_{\varepsilon}$; // refine using the counter example

  $H_0 \leftarrow \mathcal{P}rob\,[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}] < c$; // refine H_0 with new $\mathbf{d}_{ub}$

  $H_1 \leftarrow \mathcal{P}rob\,[\mathcal{T}, x[0], \tau_{nom}, \mathbf{d}_{ub}] \geq c$; // refine H_1 with new $\mathbf{d}_{ub}$

  $res \leftarrow \texttt{Verifier}_B(H_0, H_1)$; // re-perform statistical verification

  if res = *True* **then**

    return $\mathbf{d}_{ub}$;

We conclude this subsection by bounding type I errors, *i.e.*, where the alternative hypothesis accepted but the null hypothesis actually holds. According to [49], the type I error rate is bounded by:

$$err(B,c) = \frac{c}{c+(1-c)B}.$$

We also note that our iterative procedure terminates only when the alternative hypothesis is accepted. Hence, type II errors (where the null hypothesis is accepted when the the alternative hypothesis actually holds) are not relevant.

4.4 Statistical Hypothesis Testing with Sets of Initial States

Computing maximum deviation under timing uncertainties (Problem 1), using our proposed Statistical Hypothesis Testing Approach, requires computing several random trajectories, and computing deviation between those trajectories with the given nominal trajectory. Here, it is worth recalling that trajectories are a sequence of set of states (see Fig. 4.4), as the evolution starts from an initial set of states. To compute the deviation between two such trajectories, we use Hausdorff distance at every time step (as defined in Eq. (4.6)). Thus, the representation of sets must be chosen based on its impact on the computational complexity while computing the trajectories and Hausdorff distance. Some representations, such as zonotopes [85] and

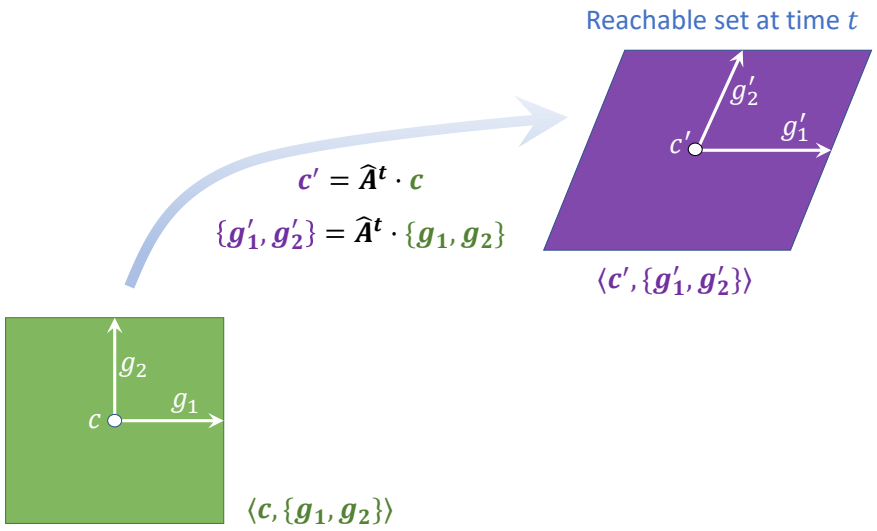


Figure 4.5: Reachability with zonotopes.

stars [39], can very efficiently compute trajectories, but computing Hausdorff distance is hard. While some representations, such as V-Polytope [86, 84], can compute Hausdorff distance very efficiently, but computing trajectories is hard. Unfortunately, there is no one representation that can compute both the trajectories and the distance efficiently. Therefore, in this section, we discuss the following choices of representation and its computational complexity *vis-à-vis* computing trajectories and distances. Further note that the techniques mentioned in this section are also incorporated in Algorithm 9. In other words, Algorithm 9 can handle initial set of states, not just a single state.

4.4.1 Representing Sets using Zonotopes

A zonotope is an affine transformation of an unit (hyper-)box, represented by its center (c) and a set of generator vectors (G). Fig. 4.5 shows two zonotopes (in $\mathbb{R}^2$) in green and purple. A zonotope $\mathcal{Z}$ in $\mathbb{R}^n$ is define formally as follows.

Definition 29 (Zonotopes [85])**.** *A zonotope $\mathcal{Z}$ is tuple $\langle c, G\rangle$; where $c \in \mathbb{R}^n$ is the center, and $G = \{g_1, g_2, \cdots, g_m\}$ is a set of m vectors in $\mathbb{R}^n$ called the generator vectors. The set of states represented by the zonotope $\mathcal{Z}$ is given as*

$$[\![\mathcal{Z}]\!] = \{x \mid x = c + \sum_{i=1}^{m} \alpha_i g_i;\ \textit{such that, for all } i,\ -1 \leq \alpha_i \leq 1\}.$$

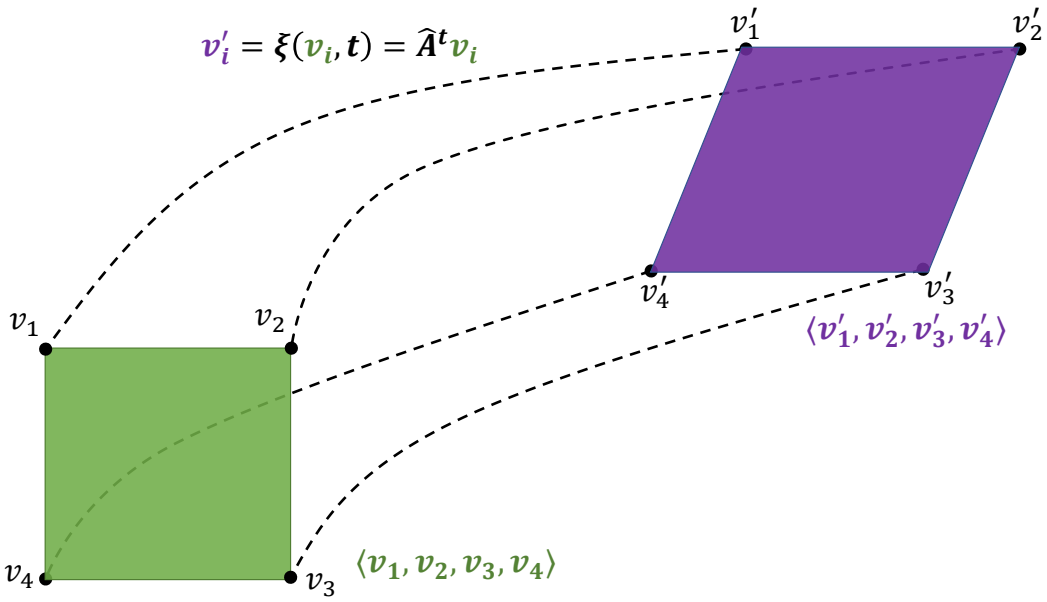


Figure 4.6: Reachability with V-Polytopes.

Given a linear system, with dynamics matrix $\hat{A}$, the reachable set at time step t using V-Polytopes can be computed as follows.

Definition 30 (Reachability using V-Polytopes)**.** *The reachable set at time step t, from an initial set represented using a V-Polytope $\mathcal{V}_0$, is given by V-Polytope $\mathcal{V}_t$; where*

$$\mathcal{V}_t = \hat{A}^t \cdot \mathcal{V}_0 = \langle \hat{A}^t \cdot v_1, \hat{A}^t \cdot v_2, \cdots, \hat{A}^t \cdot v_m \rangle$$

This is also illustrated in Fig. 4.6, where $\mathcal{V}_0$ and $\mathcal{V}_t$ are illustrated in green and purple respectively.

Hausdorff distance between two V-Polytopes, $\mathcal{V}_1$ and $\mathcal{V}_2$, can be computed as follows.

Definition 31 (Hausdorff distance with V-Polytopes)**.** *Given two V-Polytopes, $\mathcal{V}_1 = \langle v_1^1, v_1^1, \cdots, v_p^1 \rangle$ and $\mathcal{V}_2 = \langle v_1^2, v_1^2, \cdots, v_q^2 \rangle$, the Hausdorff distance between $\mathcal{V}_1$ and $\mathcal{V}_2$ can be computed as:*

$$\Delta(\mathcal{V}_1, \mathcal{V}_2) = \max \left\{ \max_i \left\{ \min_j \{ dis(v_i^1, v_j^2) \} \right\}, \max_j \left\{ \min_i \{ dis(v_i^1, v_j^2) \} \right\} \right\}$$

Advantage. Computing Hausdorff distance between two V-Polytopes, as given in Definition 31, is straightforward and computationally very efficient.

Disadvantage. Enumerating all potential vertices is required to represent a polytope with its vertices, and this process takes $O(2^n)$ time, where n is the system's dimension. While it introduces computational

bottlenecks in high dimensional systems, *this is not a significant problem in low dimensional control applications.* Therefore, using Definition 30, one can compute trajectories using $evol(\cdot)$, where each element of the sequence is a V-Polytope, if $x[0]$ is also represented using a V-Polytope.

V-Polytopes: Our choice of representation After considering the benefits and drawbacks of zonotopes and V-Polytopes, we chose to *use V-Polytopes in our experiments. However, we wish to note that one can adapt their implementation to use any representation (such as zonotopes, stars, V-Polytopes) according to their application—as our method poses no restriction on the choice of representation of sets.*

1. *Efficient implementation of Hausdorff distance.* Since our method heavily relies on performing random sampling of trajectories to compute its deviation from the nominal trajectory, an efficient computation of Hausdorff distance between trajectories is critical. For a typical case study, a single iteration of our algorithm requires computing deviation of over 1K trajectories, each up to a time bound of 150. Such an iteration involves 150K calculations of the Hausdorff distance. Thus, V-Polytope is an obvious choice in this regard.

2. *Computing Random Trajectories.* While zonotopes are more efficient at computing trajectories than V-Polytopes, this is not a significant problem for low-dimensional control applications. Moreover, given V-Polytopes' advantage of computing Hausdorff distance, we chose to use V-Polytopes in our experiments.

4.5 Handling Environmental Disturbances

So far we have only considered uncertainties in the initial set, resulting in initial set of states. We now take a closer look at how our proposed approach can be used for systems whose environmental disturbances produce additive noise in the set of states reached at each time step. Such a behavior can be modeled using the following dynamics.

$$x[t+1] = Ax[t] + Bu[t] + Cs[t],$$

where $u[t] = Kx[t-1]$, and $s[t]$ encodes the environmental disturbances experienced by the system at every time step.

Given $s[t]$, for all t, the evolution of the system, therefore, changes as

$$\begin{aligned}\widehat{evol}(\tau) = \big\{x[0], x[1] = A_1 x[0] \oplus Cs[0], x[2] = A_2 x[1] \oplus Cs[1],\\ \ldots, x[H] = A_H x[H-1] \oplus Cs[H-1]\big\},\end{aligned} \tag{4.14}$$

where τ is a random run, and $\oplus$ denotes Minkowski sum between two sets.

We would like point out that the only change that occurs, as a result of environmental uncertainties, is in the computation of trajectories (*i.e.* $\widehat{evol}$), while the rest of the proposed method remains the same. Further note that the only change in, computation of $\widehat{evol}$, is the Minkowski sum of sets. Therefore, next we discuss how to compute Minkowski sum when the sets are represented as zonotopes and V-Polytopes.

Definition 32 (Minkowski sum of zonotopes)**.** *Given zonotopes, $\mathcal{Z}_1 = \langle c_1, G_1 \rangle$ and $\mathcal{Z}_2 = \langle c_2, G_2 \rangle$, we can compute the Minkowski sum, $\mathcal{Z} = \mathcal{Z}_1 \oplus \mathcal{Z}_2$ as follows. $\mathcal{Z} = \langle c, G \rangle$, where $c = c_1 + c_2$, and G is a list of generator vectors obtained by joining the list of generator vectors G_1 and G_2.*

Clearly, when the sets are represented as zonotopes, computing trajectories can be performed very easily using Definition 32. However, note that every Minkowski sum results in adding more generator vectors in the representation. In other words, every Minkowski sum increases the representational complexity of zonotopes—which will have impact on the computation of Hausdorff distance too.

After discussing Minkowski sum of zonotopes, we now briefly discuss Minkowski sum of V-Polytopes. Using the algorithm proposed in [86], one can perform Minkowski sum of V-Polytopes in polynomial time—this further justifies our usage of V-Polytopes. In other words, zonotopes may experience computational inefficiencies while computing Hausdorff distance, however employing V-Polytopes does not lead to any significant computational bottleneck. **One can easily incorporate these modifications in Algorithm 9**.

4.6 Illustration of the Proposed Approach And Advantages of Jeffreys's Bayes Factor

In this section, we demonstrate how the three modules described in Section 4.3, namely `Hypothesizer`, `Verifier`, and `Refiner`, are used by Algorithm 9 to compute an upper bound on the deviation ($\mathbf{d}_{ub}$) with a probabilistic guarantee. This is to intuitively demonstrate our main approach on the following illustrative

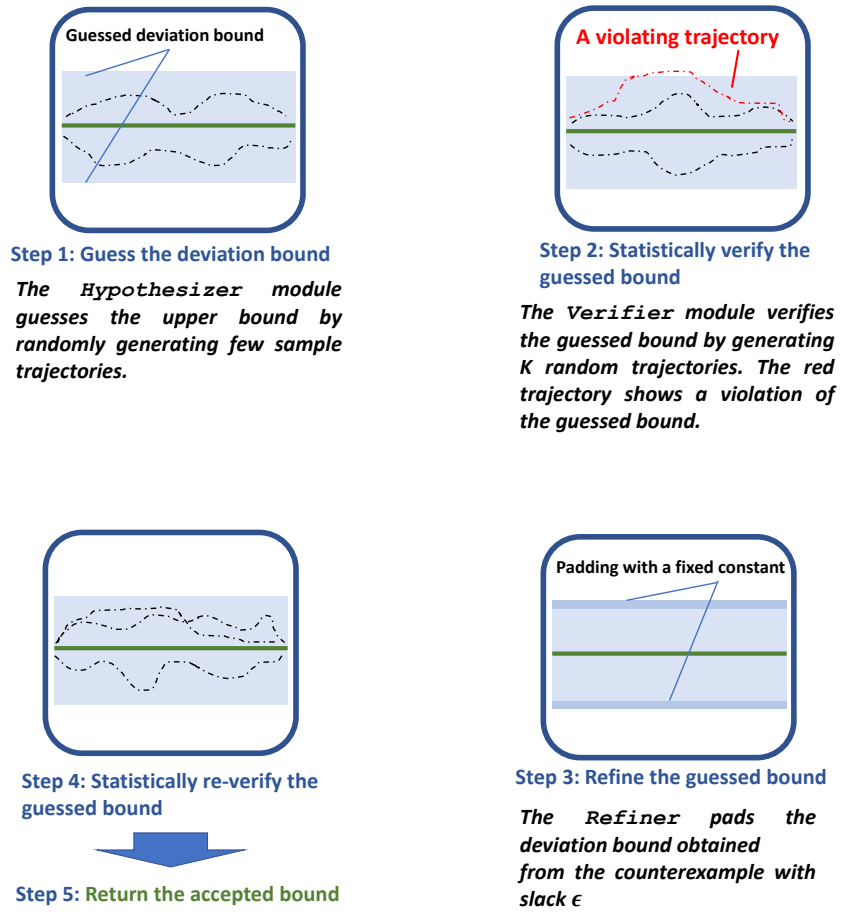


Figure 4.7: The steps performed by Algorithm 9 to compute a deviation bound with a desired confidence. The nominal trajectory is shown in green, randomly generated trajectories are shown in black, and $\mathbf{d}_{ub}$ is shown in light blue.

example:

$$x[t+1] = \begin{bmatrix} 0.1 & 0.2 \\ 0.2 & 0.1 \end{bmatrix} x[t] + \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} u[t]$$
$$u[t] = \begin{bmatrix} 0.05149186 & 0.4189839 \end{bmatrix} x[t]$$

We will compute the maximum deviation from a nominal trajectory with no deadline misses, starting from the initial state $x[0] = \langle [10 \;\; 10]^T \rangle$ (represented as a V-Polytope). We assume that no more than two consecutive deadline misses occur up to our time bound $H = 5$. Using the *Hold&Skip-Next* policy, we can accordingly construct a transducer automaton representing this behavior of the scheduler and dynamical system. Given these inputs, Algorithm 9 performs the following steps to compute the maximum deviation, illustrated in Fig. 4.7.

1. The first module invoked by Algorithm 9 is the `Hypothesizer`, which *guesses* an upper bound $\mathbf{d}_{ub}$ to be the maximum deviation. To do so, the module considers the following two random sequences of deadline hit/miss: 01001, 00111 (0 indicates miss, 1 indicates hit). It computes the maximum deviation from the two random samples, $\mathbf{d}_{ub} = 0.1462$.

2. Next, the guessed upper bound $\mathbf{d}_{ub} = 0.1462$ is verified by invoking the module `Verifier`. The `Verifier` module, pre-tuned with $B = 4.15 \times 10^5$ and $c = 0.99$, returns *False*, *i.e.*, the upper bound $\mathbf{d}_{ub} = 0.1462$ is not verified to be correct with the desired probabilistic guarantees.

3. Since the guessed bound $\mathbf{d}_{ub}$ was not verified, Algorithm 9 next invokes the `Refiner` module with the counterexample. `Refiner` updates the previous $\mathbf{d}_{ub} = 0.1462$ to $\mathbf{d}_{ub} = 0.2262$ (by padding a fixed constant of 0.001 on top top of the deviation bound obtained from the counterexample).

4. The refined upper bound $\mathbf{d}_{ub} = 0.2262$ is again sent to the `Verifier` module for re-checking. This time, the `Verifier` module accepts the $\mathbf{d}_{ub} = 0.2262$ as a valid upper bound up-to the desired probabilistic guarantees.

5. The final $\mathbf{d}_{ub} = 0.2262$ is returned as the maximum deviation (with the desired probabilistic guarantees).

Having illustrated the steps performed by our algorithm on a simple example, we argue in the following subsection why we chose Jeffreys's Bayes factor over other methods.

4.6.1 Reason for Choosing Jeffreys's Bayes Factor

In theory, it is possible to use other hypothesis testing methods, such as sequential hypothesis testing. However, for this work, Jeffreys's Bayes factor enjoys several advantages.

1. Our method imposes no restriction on the distribution. It merely requires that samples can be drawn from the distribution at random.

2. In sequential hypothesis testing, unlike our method, the number of required samples cannot be fixed *a priori*.

3. Multiple counterexamples can be encountered during random sampling in a round of sequential hypothesis testing, allowing known counterexamples to be explored. In our setting a counterexample represent a violation of a safety property estimate and accordingly Jeffreys's Bayes factor does not allow such known counter examples.

4. Similar to sequential hypothesis testing, our method is independent of the sample space size once the desired confidence (on the answer) is provided by the user.

4.7 Experimental Evaluation

The tool, `StatDev`[1], is available both in Python[2] and Juila[3].

For $dis(\cdot)$ we use the 2-norm. As mentioned in the introduction, to generate uniform random samples for a given transducer automaton, we implemented the *Recursive RGA* algorithm [81, 87]. We demonstrate the applicability of the method given in Algorithm 9, extended to handle initial set of states as per Section 4.4, on four standard examples: an RC network [88], an electric steering application [8], an unstable second-order system [8] and a F1Tenth car model [79]. For these examples we investigate the following questions.

Q1: What impact do the different scheduling policies have on on the computed deviations?

Q2: What effect does the probabilistic guarantee c have on the computed deviation?

[1] `sites.google.com/view/statdev`

[2] `https://github.com/bineet-coderep/StatJitteryScheduler`

[3] `https://github.com/Ratfink/ControlTimingSafety.jl`

Table 4.1: Results Comparing Our Proposed Statistical Method to the RS Method on Four Examples

Example	Statistic	*Hold&Kill*	*Zero&Kill*	*Hold&Skip-Next*	*Zero&Skip-Next*
RC network	$\mathbf{d}_{ub}$ (Algorithm 9)	2.277 (0)	2.277 (0)	2.277 (0)	2.277 (0)
	$\mathbf{d}_{ub}$ (RS)	2.277	2.277	2.277	2.277
	Time Taken (Algorithm 9)	1.745 s	1.774 s	1.878 s	1.831 s
	Time Taken (RS)	0.918 s	1.040 s	0.943 s	1.012 s
Electric Steering	$\mathbf{d}_{ub}$ (Algorithm 9)	4.568 (0)	9.297 (0.28)	4.573 (0.027)	9.168 (0.28)
	$\mathbf{d}_{ub}$ (RS)	4.795	10.226	8.882	10.625
	Time Taken (Algorithm 9)	1.74 s	2.90 s	1.79 s	2.90 s
	Time Taken (RS)	1.569 s	47.43 s	178.0 s	182.8 s
Unstable second-order	$\mathbf{d}_{ub}$ (Algorithm 9)	3.959 (0)	14.969 (1.17)	4.632 (0)	12.767 (1.06)
	$\mathbf{d}_{ub}$ (RS)	4.269	—	5.162	—
	Time Taken (Algorithm 9)	1.50 s	2.46 s	1.99 s	2.97 s
	Time Taken (RS)	1.068 s	timed out ($>$ 1 h)	3.837 s	timed out ($>$ 1 h)
F1Tenth	$\mathbf{d}_{ub}$ (Algorithm 9)	10.42 (0)	19.08 (0.83)	18.53 (1.34)	18.90 (0.74)
	$\mathbf{d}_{ub}$ (RS)	10.425	—	—	—
	Time Taken (Algorithm 9)	1.81 s	2.87 s	3.07 s	2.76 s
	Time Taken (RS)	171.1 s	timed out ($>$ 1 h)	timed out ($>$ 1 h)	timed out ($>$ 1 h)

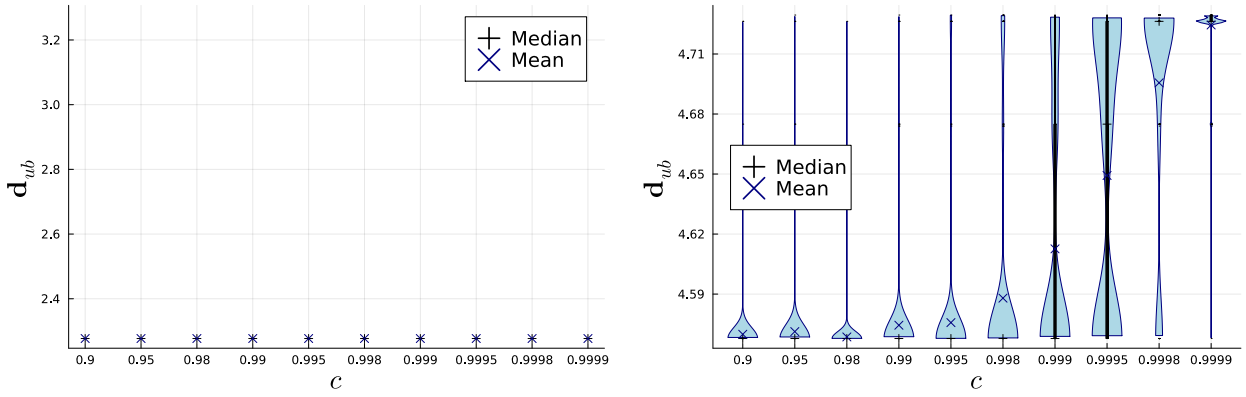
Q3: How do our statistically computed deviations compare to the results obtained using the RS method? Recall that RS refers to the deterministic reachable set based method described in Section 4.2.

The following parameters were used in the experiments: $R = 50$, $\varepsilon = 10^{-3}$, $B = 4.15 \times 10^5$, an initial state of $\langle [10\ \ 10]^T, [12\ \ 10]^T, [12\ \ 12]^T, [10\ \ 12]^T \rangle$ (represented as a V-Polytope). and time bound $H = 150$.

For Fig. 4.9, we use a single initial state $[10\ \ 10]^T$, with rest of the parameters same. Since Algorithm 9 is stochastic, we execute it over several trials (50 in this case) and report the mean and the standard deviation (SD) of the obtained $\mathbf{d}_{ub}$ values. For instance, 5 (0.3) means that the mean value of $\mathbf{d}_{ub} = 5$ with SD 0.3, and the reported computation time is the average computation time taken.

The DFAs we consider enforce constraints of the form "at most k consecutive misses." But any other form of constraints, as long as they are regular, could also be used. We denote these DFAs as $\{\mathcal{T}_k\}$ with k ranging over $\{1,2,3\}$. For most of the experiments, the DFA $\mathcal{T}_3$ was used.

The main results are summarized in Table 4.1; *the statistical method always computes tighter bounds than RS*. For unstable systems and and F1tenth, RS seems to have an exponential increase in computation time, whereas our statistical method scales much better (note that RS fails to compute a bound within an hour).



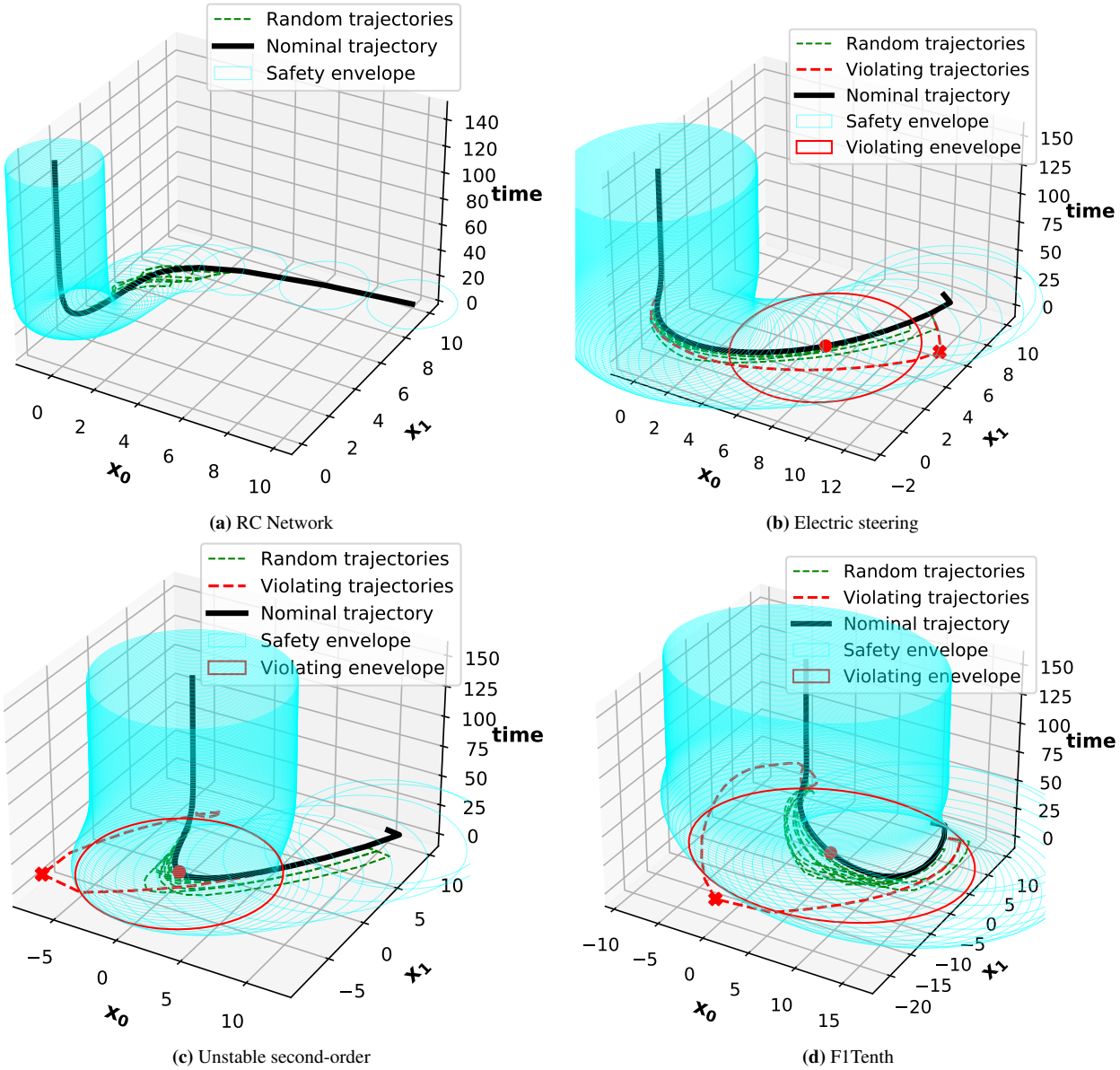
(a) RC network

(b) Electric steering

(c) Unstable second-order

(d) F1Tenth

Figure 4.8: Violin and box plots of computed $\mathbf{d}_{ub}$ values over varying probabilistic guarantee c on the computed bound. All boxes in Fig. 4.8a are single values.



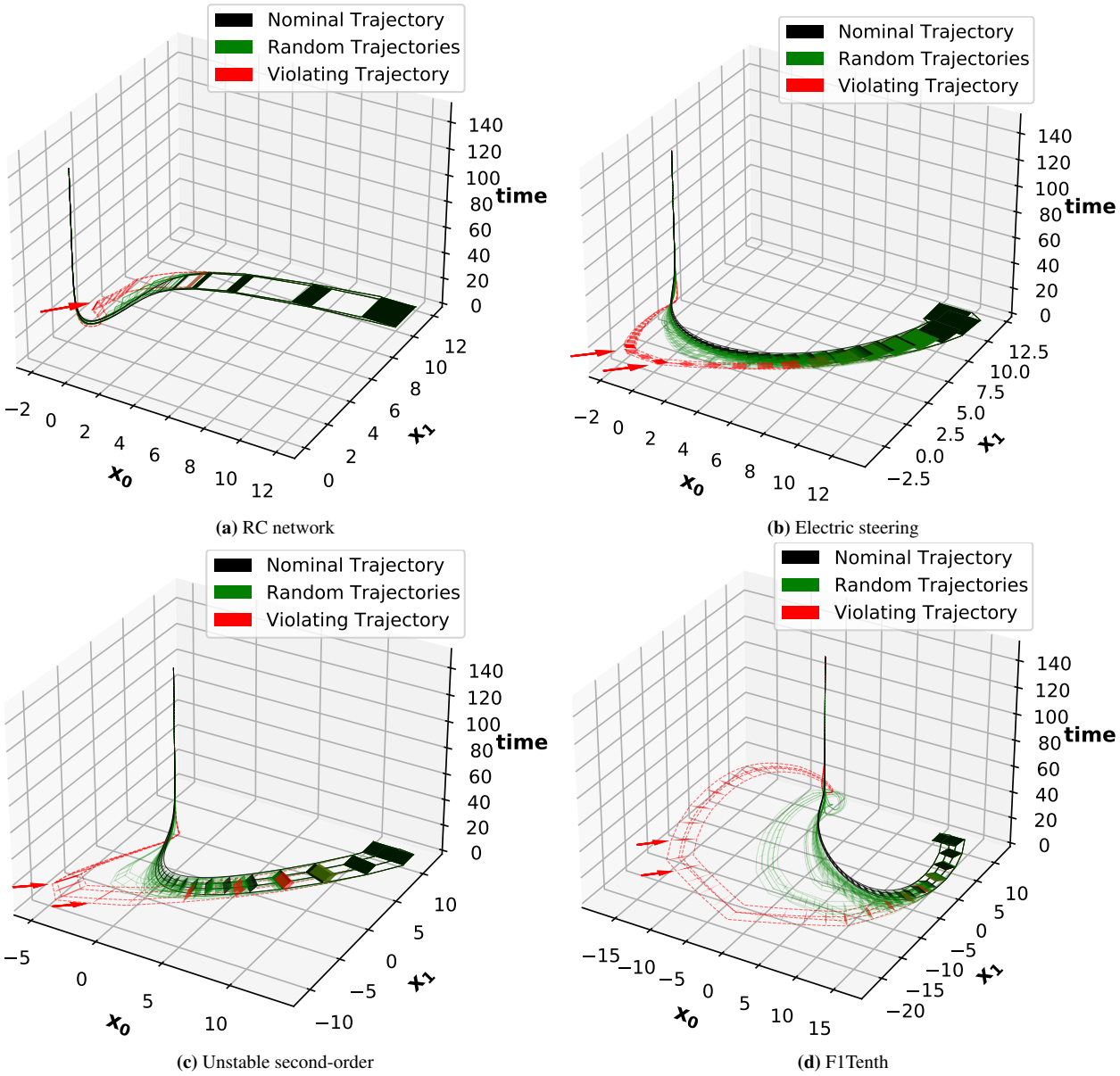
(a) RC Network

(b) Electric steering

(c) Unstable second-order

(d) F1Tenth

Figure 4.9: System behavior with a single initial state [1]. Safety envelopes at a distance of $\mathbf{d}_{ub}$ from the nominal trajectory, with random trajectories with one extra consecutive deadline miss.



(a) RC network

(b) Electric steering

(c) Unstable second-order

(d) F1Tenth

Figure 4.10: System behavior with random trajectories (green) that are within $\mathbf{d}_{ub}$ distance from the nominal trajectory (black), and one violating trajectory (red) with one extra consecutive deadline miss—this trajectory deviates more than $\mathbf{d}_{ub}$ from the nominal trajectory.

4.7.1 Benchmarks

4.7.1.1 RC network

The RC network is given by the following state space equations.

$$x[t+1] = \begin{bmatrix} 0.5495 & 0.07240 \\ 0.01448 & 0.9332 \end{bmatrix} x[t] + \begin{bmatrix} 0.3781 \\ 0.05234 \end{bmatrix} u[t] \tag{4.15}$$

Assuming nominal timing behavior of the control software, the feedback controller computed using LQR is

$$u[t] = \begin{bmatrix} 0.09772 & 0.2504 & 0.07805 \end{bmatrix} \begin{bmatrix} x[t-1] \\ u[t-1] \end{bmatrix}. \tag{4.16}$$

Using the matrices from Eqs. (4.15) and (4.16), we construct a transducer automaton for different deadline miss strategies. We used $\mathcal{T}_3$ to specify the scheduler.

4.7.1.2 Electric steering

The electric steering example is given by the following state space equations.

$$x[t+1] = \begin{bmatrix} 0.996 & 0.075 \\ -0.052 & 0.996 \end{bmatrix} x[t] + \begin{bmatrix} 0.100 & 0.003 \\ -0.003 & 0.083 \end{bmatrix} u[t]. \tag{4.17}$$

Optimal feedback controller for this system under nominal timing behavior computed using LQR is:

$$u[t] = \begin{bmatrix} 0.9067 & 0.07384 & 0 & 0 \\ 0.01041 & 0.9685 & 0 & 0 \end{bmatrix} \begin{bmatrix} x[t-1] \\ u[t-1] \end{bmatrix}. \tag{4.18}$$

As before, using the matrices from Eqs. (4.17) and (4.18), we construct a transducer automaton for different deadline miss strategies. We used $\mathcal{T}_3$ to specify the scheduler.

4.7.1.3 Unstable second-order system

The unstable second-order system example is given by the following state space equations.

$$x[t+1] = \begin{bmatrix} 1.1053 & 0 \\ -0.0209 & 0.99 \end{bmatrix} x[t] + \begin{bmatrix} 0.0526 & 0.0105 \\ 0.0393 & 0.0994 \end{bmatrix} u[t] \tag{4.19}$$

The control input $u[t]$ is computed as

$$u[t] = \begin{bmatrix} 4.7393 & -0.2430 \\ -0.2277 & 0.8620 \end{bmatrix} x[t-1]. \tag{4.20}$$

Using the matrices from Eqs. (4.19) and (4.20), we construct a transducer automaton $\mathcal{T}_1$ to specify the scheduler.

4.7.1.4 F1Tenth

The model of an F1Tenth car [79] was linearized and the following state space equations were obtained.

$$x[t+1] = \begin{bmatrix} 1 & 0.13 \\ 0 & 1 \end{bmatrix} x[t] + \begin{bmatrix} 0.02559 \\ 0.3937 \end{bmatrix} u[t] \tag{4.21}$$

The control input $u[t]$ is computed as

$$u[t] = \begin{bmatrix} 0.2935 & 0.4403 \end{bmatrix} x[t-1]. \tag{4.22}$$

Using the matrices from Eqs. (4.21) and (4.22), we construct a transducer automaton $\mathcal{T}_3$ to specify the scheduler.

4.7.2 Results

The main results addressing (Q1) and (Q3) are summarized in Table 4.1. For the RC network, electric steering and F1Tenth, $\mathcal{T}_3$ (*i.e.*, the constraint "at most 3 consecutive misses") was used with $c = 0.99$. Since the third system is an unstable open loop system, missing too many deadlines would cause the deviation

bound to increase drastically; therefore, we only consider $\mathcal{T}_1$ (*i.e.*, the constraint "at most 1 deadline miss consecutively").

To address (Q2), we fixed $R = 10$ and using $\mathcal{T}_3$ for RC network, electric steering, F1tenth, and $\mathcal{T}_1$ for the unstable second order system, we varied c (the probabilistic guarantee) from 0.9 to 0.9999. We used the *Hold&Skip-Next* strategy for these experiments. The resulting values of $\mathbf{d}_{ub}$ are shown by boxplots and mean values in Fig. 4.8. Below we discuss the results in detail for all the examples.

4.7.2.1 RC network

We now discuss the results obtained for the RC network example.

Q1: Considering at most 3 consecutive deadline misses allowed, we computed the maximum deviation $\mathbf{d}_{ub}$ to be 2.278 (0) for all the scheduling policies. The details are given in Table 4.1. The system behavior is shown in Figs. 4.9a and 4.10a, using the *Hold&Skip-Next* policy, and $c = 0.99$. Fig. 4.9a shows the system behavior with a single initial state, whereas Fig. 4.10a shows the system behavior with an initial set of states. Thus, we see trajectories as a series of points in Fig. 4.9a, and a series of polytopes in Fig. 4.10a. We plot Fig. 4.9a primarily to illustrate the role of safety envelopes, and how trajectories might violate it. The trajectories of the system from a single initial state will have safety envelopes plotted as a circles of radius $\mathbf{d}_{ub}$ (cyan) from the nominal behavior (black), as show in all the subplots of Fig. 4.9. However, this is not the case when the trajectories are resulting from an initial set of states. Thus, we indicate the violating trajectory—one that deviates more the $\mathbf{d}_{ub}$ from the nominal trajectory—in red, with the maximum and minimum violating time steps marked with arrows. The red trajectory shows safety violation (*i.e.* deviates more than $\mathbf{d}_{ub}$ from the nominal trajectory) by increasing the consecutive deadline from 3 to 4. The red arrow shows the point at which the system violates the safety. That is, at this point, the trajectory deviates more than $\mathbf{d}_{ub}$ from the nominal trajectory. Note that we did not find any safety violation with a single initial state (Fig. 4.9a).

Q2: Considering at most 3 consecutive deadline misses allowed and $R = 10$, we gradually varied c from 0.9 to 0.9999. We observe that the mean $\mathbf{d}_{ub}$ does not change with any further increase in c from $c = 0.9$. This is shown in Fig. 4.8a.

Q3: The results show (see Table 4.1) that the stochastic method returns similar bounds as the `RS` method. However, the computation time is higher but not significantly high, *i.e*, under 2 s.

4.7.2.2 Electric steering

We now discuss the results obtained for the electric steering example.

Q1: Similar to our previous example, considering at most 3 consecutive deadline misses allowed, we computed the maximum deviation $\mathbf{d}_{ub}$, with the scheduling policies (detailed results in Table 4.1). The system behavior is shown in Figs. 4.9b and 4.10b, using the *Hold&Skip-Next* policy, and $c = 0.99$. Further, the figures also show a possible safety violation that might occur when the number of consecutive deadline misses just increases by 1, *i.e.*, considering at most 4 consecutive deadline misses. The red trajectories in Figs. 4.9b and 4.10b show a safety violation that might occur if the consecutive deadline misses increases by 1. The red dotted line in Fig. 4.9b shows the safety violation (with the existing safety envelope) that might occur if the consecutive deadline misses increases by 1. The safety envelope highlighted in red shows the violating envelope. The behavior would have been safe if the trajectory (in red) was within the highlighted safety envelope (red), whereas it actually stretches outside to the point marked in '$\times$' (red). In Fig. 4.10b, we indicate the maximum and minimum violating time steps marked with arrows (red).

Q2: Considering at most 3 consecutive deadline misses allowed and $R = 10$, we gradually varied c from 0.9 to 0.9999. The result is given in Fig. 4.8b. We observe that the mean $\mathbf{d}_{ub}$ increases with increase in c, as expected. However, higher outlier deviation values are sometimes returned, unlike in the previous example. Also, note that with low values of c, we witness a wide range, which narrows as c increases.

Q3: As given in Table 4.1, we compute tighter values of $\mathbf{d}_{ub}$. The computation time for the stochastic method is lower for all strategies, except for *Hold&Kill* where it is slightly higher (≈ 0.2 s).

4.7.2.3 Unstable second-order system

We now discuss the results obtained for the unstable second-order system example.

Q1: Unlike our previous examples, Since the third system is an unstable open loop system, missing too many deadlines would cause the deviation bound to increase drastically; therefore, we only consider $\mathcal{T}_1$ (i.e. the constraint "at most 1 deadline miss consecutively"). We computed the maximum deviation $\mathbf{d}_{ub}$ with the scheduling policies (see Table 4.1). The system behavior is shown in Figs. 4.9c and 4.10c, using the *Hold&Skip-Next* policy, along with a violating trajectory when the constraint changes to $\mathcal{T}_2$.

Q2: Considering at most 1 consecutive deadline miss allowed and $R = 10$, we gradually varied c from 0.9 to 0.9999. The result is given in Fig. 4.8c. We observe that the mean $\mathbf{d}_{ub}$ increases with increase in c, as expected. Also, note that with low values of c, we witness a wide range, which narrows (to zero) as c increases—$\mathbf{d}_{ub}$ seems to stabilizing at $c = 0.995$.

Q3: As given in Table 4.1, we compute tighter values of $\mathbf{d}_{ub}$ in less computation time for all the scheduling policies. Moreover, the `RS` method times out ($>$ 1 h) for *Zero&Kill* and *Zero&Skip-Next* policies.

4.7.2.4 F1Tenth

We now discuss the results obtained for the F1tenth example.

Q1: Similar to the first two examples, we computed the maximum deviation $\mathbf{d}_{ub}$ with the scheduling policies, considering at most 3 consecutive deadline misses. The safety envelope is shown in Figs. 4.9d and 4.10d, using the *Hold&Skip-Next* policy, and $c = 0.99$. Further, the figures also show a possible safety violation that might occur when the number of consecutive deadline misses increases by 1 (at most 4 consecutive deadline misses).

Q2: Unlike other examples, where the width of the distribution decreases sharply, in the F1Tenth example the decrease is not as drastic. This is shown in Fig. 4.8d. However, following the method of [8], we computed an upper-bound on the *joint spectral radius* and found the system with at most three deadline consecutive misses is stable. This suggests that even though the overall behavior of the system is stable, the system possibly behaves erratically prior to obtaining stability—that is, the variance of the deviation obtained for various behaviors (w.r.t. deadline hits and misses) is very high. To put it differently, the deviation obtained from two different sequences of deadline hits and misses can be very different.

Q3: The `RS` method, except for *Hold&Kill* was not able to compute a reasonable bound on the deviation. Whereas our proposed method computed reasonable bound, for all policies, under 3.5 s.

4.7.3 General Observations

In this subsection, we draw general observations from our experiments that might help the users to tune this method according to their application. In other words, the following observations are drawn to provide a rule of thumb to choose the parameters and gather insights:

1. Revisit (Q1).

2. Revisit (Q2): The parameter c (confidence on the deviation).

3. Revisit (Q3): When to use a traditional method (RS) over our statistical method.

4. How to choose the parameter R.

Revisit (Q1). We observe (from Table 4.1) that there is no one scheduling policy that performs consistently well (in terms of smaller deviation bounds) across all the benchmarks. This suggests that the choice of scheduling policy should be made according to the given application.

***Revisit (Q2): Choice of* c.** A good strategy is to use a high value of c, so that the standard deviation is low and further increases in c will have minimal effect. We note, however, that the unstable second-order and the F1Tenth example with at most 3 consecutive deadline misses fail to achieve a low standard deviation even with $c = 0.99$.

Revisit (Q3): Choice of method to be used. For small dimensional stable systems, like the RC network, traditional methods might work well. However, for unstable systems they are likely to perform poorly due to coarse over-approximations. Note that our statistical method was able to computer tighter bounds, compared to the RS method, for all the examples. Next, we discuss why the RS method timed out in most cases. By segmenting the time horizon into manageable chunks, the RS technique computes the reachable set by computing all possible trajectories for each segment. It is possible to compute all possible trajectories when each of these parts is small enough. By dividing the time horizon into smaller chunks, we were able to get useful deviation bounds for some of the benchmarks, but not for the majority of them. As a result, we had to increase the number of segments by which we break the time horizon, which made it impossible to compute every possible trajectory in that segment and caused a timeout.

***Choice of* R.** As our method is stochastic, for smaller values of R (and highly chaotic systems), the initial $\mathbf{d}_{ub}$ guess can vary greatly (for different trials). And when such an initial guess is being verified with a low probability c, the chances of the initial guess being accepted is very high. Therefore the SD for low values of c is also high, and the obtained bound $\mathbf{d}_{ub}$ is not guaranteed to be monotonic.

Comments on Fig. 4.8 and Fig. 4.10 Further, we make general comments on Fig. 4.8 and Fig. 4.10 as follows.

Varying the value of c **(Fig. 4.8).** We observe that for lower values of c, the mean values of $\mathbf{d}_{ub}$ are lower but the distributions are wider, however, as c increases the mean values of $\mathbf{d}_{ub}$ increase whereas the width of the distribution decreases sharply, except for F1Tenth where the decrease is not as drastic as other examples. For F1Tenth, the computed *joint spectral radius* showed the system with at most three deadline consecutive misses to be stable, suggesting the system behavior is possibly erratic prior to obtaining stability. At lower values of c, the distribution is wider, therefore the variance on the computed bound is much higher at lower values of c. The computed bound is however observed to be stabilizing with the distribution narrowing at only higher values of c, except for the F1Tenth example. Again, the value of c at which the computed mean bound stabilizes is application specific. For instance, in case of electric steering the value of c at which the bound stabilizes is much higher than the one required for unstable second-order system.

System Behavior **(Fig. 4.10).** In this figure, we show the behavior of the system using the *Hold&Skip-Next* and $c = 0.99$. For RC network, electric steering and F1Tenth, all the trajectories satisfying the $\mathcal{T}_3$ constraint will be within $\mathbf{d}_{ub}$ from the nominal trajectory (with a probabilistic guarantee). Similarly, for the unstable second order system we used the constraint $\mathcal{T}_1$. We observe that for the electric RC network example, the violating trajectory is closer to the random trajectories, compared the other three examples—this might be due to that fact that RC network behaves less erratically than the other three examples. Further, we evaluated the robustness of the system *w.r.t* the computed $\mathbf{d}_{ub}$, by changing the constraint from $\mathcal{T}_3$ to $\mathcal{T}_4$ for RC network, electric steering and F1Tenth, and from $\mathcal{T}_1$ to $\mathcal{T}_2$ for the unstable second order system. The violating trajectory is shown in red, and the arrows mark the minimum and maximum violating time steps.

Using Fig. 4.8 to compute Fig. 4.10. Note that Fig. 4.8 suggests a heuristic to choose a value for c (at which the mean stabilizes with a narrow distribution). Once such a c is chosen, the safety envelope should be computed, as in Fig. 4.10, using that value of c.

4.8 Synthesizing Schedulers in the Presence of Timing Uncertainties

Correctness of control implementations has relied on real-time guarantees that all control tasks would finish execution by the prescribed deadline. However, with increased complexity and heterogeneity in hardware, the worst-case execution time estimates are becoming very conservative. Thus, for efficient usage of hardware resources, some control tasks would have to miss some deadlines. As previous sections argued, a system can still abide by its safety requirements even after missing some of its deadlines. This section briefly discusses an approach to synthesize a scheduler for control tasks that miss some deadlines without compromising any of the safety requirements. Given that the number of possible schedules would increase combinatorially with the number of tasks involved, our scheduler synthesis uses an efficient automata representation to search for the appropriate schedule. We can incorporate statistical verification techniques to construct this automaton and accelerate the search process. Statistical verification is advantageous compared to deterministic verification in the synthesis process in two ways: first, it enables us to synthesize schedules that would not be possible otherwise, and second, it drastically reduces the time taken to synthesize such a schedule.

The ***main contribution*** of this section is to discuss a technique for automatically synthesizing such schedules from a collection of feedback controllers and their associated safety properties. Instead of the safety property being *stability*—where our proposed technique may also be applied—we focus on a more general safety property defined by the maximum deviation from a *nominal behavior*. Given an initial state of the system (plant + controller) we define the nominal behavior as the trajectory in the state space where all the controller inputs meet their deadlines (*i.e.*, all the feedback control inputs are applied before the end of the sampling period). If some of the control inputs cannot be computed (and hence applied) within the sampling period (*i.e.*, there is a deadline miss) then the trajectory followed by the system will deviate from this nominal trajectory. As long as the deviation is not more than a specified bound, the behavior of the system is deemed to be *safe*. This notion of safe behaviors is a natural one. For instance, the nominal trajectory might represent the pre-determined path that an autonomous vehicle should follow and safe behaviors will denote the paths the vehicle can take, without hitting any obstacles.

The proposed schedule synthesis scheme is illustrated in Fig. 4.11. Given a plant + controller and a desired safety property (*i.e.*, the maximum allowed deviation from its nominal behavior), we first derive the pattern of allowed deadline hits and misses. This is captured as a *weakly-hard constraint* that specifies

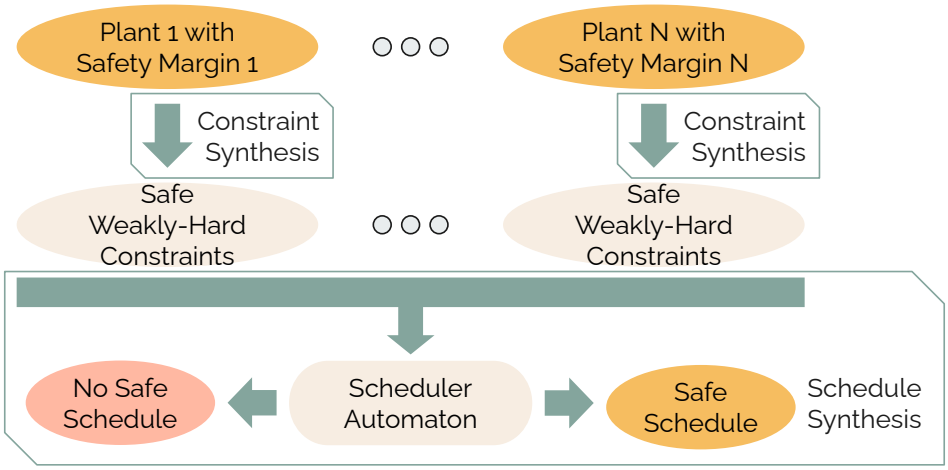


Figure 4.11: Outline of the proposed scheme.

how many deadlines m must be met within any window of k consecutive samples. Note that there could be multiple such weakly-hard constraints that satisfy the safety property of a controller. This done by using the statistical hypothesis testing based framework proposed in previous sections. Given a set of controllers and a set of weakly-hard constraints derived for each controller, we develop an automata-theoretic technique to compute a schedule for them. Such a schedule specifies which controllers should be run in each period (and hence meet their deadlines) and which cannot be run (*i.e.*, they miss their deadlines). It is important to note that such schedules may not be captured using standard scheduling policies like earliest deadline first or fixed priority. The proposed approach in [64] outlines the specifics of the automata theoretic method. It is important to note that the synthesized scheduler is represented as an automaton, with its runs representing different possible schedules. To handle the weakly hard constraints generated through statistical methods, we generate a random run of the automaton and verify if that particular schedule (*i.e.* the run of the automaton) satisfies the safety constraints. For more information, please refer to [66].

CHAPTER 5: DESIGNING AUTONOMOUS SYSTEMS—UNLOCKING PERFORMANCE PERFECTION BY SMART RESOURCE UTILIZATION

Imagine a scenario where a factory robot is required to perform safe navigation in a busy warehouse floor. To perform the navigation, the robot is equipped with a plethora of perception models that trades-off control cost and accuracy. Without loss of generality, such perception models can reside on the robot, or in a central server ("Cloud") which can be invoked over a shared network. In such scenarios, a desired strategy should be to use the high perception cost models, to ensure minimum control cost, *only* when the robot is required to make tricky maneuvers. In such settings, robots can invoke heterogeneous computation resources such as CPUs, perception models, cloud GPU servers, or even human computation for achieving a high-level goal. The problem of invoking an appropriate computation model, at every time step, so that it will successfully complete a task while keeping its compute and energy costs within a budget is called the *model selection problem*. We further break down the problem of model selection in two parts. Firstly, we propose an optimal solution to the model selection problem with two compute models in Section 5.1. Secondly, we propose an optimal model selection, balancing control cost and perception cost, with N perception models in Section 5.2.

About the chapter. The content of this chapter is taken from [47, 89].

5.1 Choosing Between Two Compute Models

Ideally, robotic computation should possess high levels of accuracy, responsiveness, and speed, while also being efficient in terms of compute and power usage. This is crucial for enabling agile and autonomous operation. However, in modern robotics, there is a challenge of choosing the most suitable compute resource from a range of heterogeneous options, each with its own trade-off between accuracy and compute cost. For instance, when faced with a decision, should a factory robot rely on the perception results generated by an on-board deep neural network (DNN), or should it seek assistance from a busy human supervisor? Similarly, should a small drone calculate its motion plan locally or wait for a more detailed plan from a remote server? Fundamentally, these scenarios represent instances of a *compute model selection* problem. In such cases, a

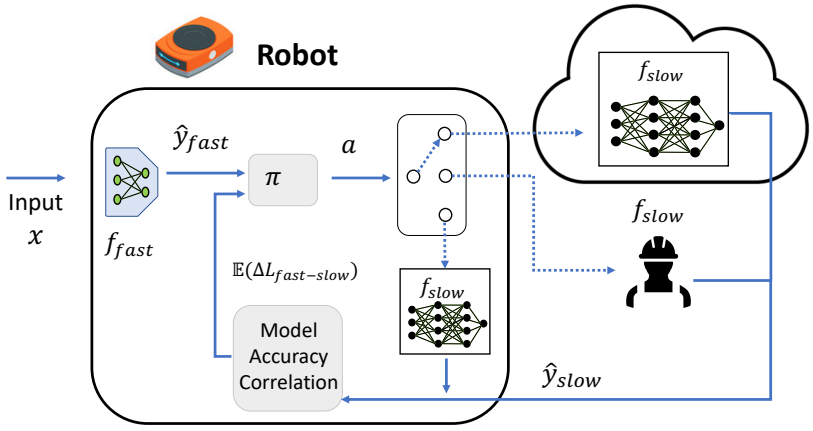


Figure 5.1: The Compute Model Selection Problem: A robot must balance task accuracy and compute cost, such as energy or latency, when choosing between heterogeneous compute resources. Our interpretable model selection policy π leverages the statistical correlation between fast and slow compute models f_{fast} and f_{slow} to dynamically decide which model to invoke.

robot must skillfully balance the level of accuracy required for the task with factors such as compute time, power consumption, network availability, and potential delays in human processing.

Figure 5.1 illustrates the model selection problem addressed in this chapter. Given the sensor observations x at each time step, a robot's model selection policy π must dynamically invoke either a fast, compute-and-power-efficient model (f_{fast}) or a slower, more accurate model (f_{slow}) based on a high-level task's required accuracy. Variants of this problem have been studied for perception tasks in cloud robotics [90, 91] and human-robot collaboration [92, 93, 94]. However, existing works either offer specialized point-solutions (e.g. for perception [95, 96]) that do not readily generalize to other domains, use hand-engineered heuristics, or employ uninterpretable, learning-based algorithms [90, 97]. *The key contribution of this section is to provide a unified, interpretable, and theoretically-grounded framework for compute model selection in robotics.*

The fundamental principle behind selecting an appropriate compute model is to perform a cost-benefit analysis. The key insight, behind the solution proposed in this section, is that a robot's model selection algorithm can leverage the statistical, and often analytical, correlation between the accuracy of the fast and the slow compute models. This correlation can enable us to perform *reliable and interpretable* cost-benefit analysis between compute cost and gain in accuracy for the different models. Crucially, such correlations between fast and slow models are now possible even for state-of-the-art DNNs, due to recent advances that compress large DNNs with provable approximation guarantees [98, 99].

Contributions. Given prior literature, the contributions of this section are three-fold. First, an interpretable model selection algorithm which leverages analytical correlations between fast and slow model performance to dynamically decide which model to invoke. Second, we show how this algorithm can naturally leverage recent advances that compress large DNNs with provable approximation guarantees that relate fast and slow models. Third, we show strong experimental performance of this algorithm on diverse domains ranging from robotic perception to sampling-based reachability analysis for a simulated rover navigating Martian terrain data.

5.1.1 The Compute Model Selection Problem with Two Compute Models: A Formal Definition

In this section, we formally define the problem of model selection depicted in Fig. 5.1 by introducing compute models, an accuracy metric, and a performance criterion.

Compute Model Input: The input to the compute model, at time t, is denoted by $x^t \in \mathbb{R}^n$. We denote the input data distribution by $\mathcal{X}$, that is, $x^t \sim \mathcal{X}$. In practice, x^t could represent a depth-camera image or laser scan.

Compute Models: The compute models are denoted by $f_i : \mathbb{R}^n \to \mathbb{R}^m$, where $i \in \{\text{fast}, \text{slow}\}$. Given an input x^t, the output is denoted by $y_i^t = f_i(x^t)$. The cost associated with f_i is given by $c_i \in \mathbb{R}_+$. The cost is context-dependent, such as battery consumption, compute inference latency, or even communication latency for cloud robotics tasks. For example, the compute models could be a DNN, with image input x^t and corresponding segmentation y^t. The distribution of outputs is denoted by $\mathcal{Y}$, that is, $y_i^t \sim \mathcal{Y}$. The ground-truth output associated with input x^t is denoted by y_{oracle}^t.

Loss Function: Let $\mathcal{L}(y_1^t, y_2^t) : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}_{\geq 0}$ be the loss function, that formally quantifies the quality of the output y_1^t returned by a compute model compared to the ground-truth result of y_2^t. A lower value of $\mathcal{L}(\cdot)$ indicates a more accurate output. In practice, the loss function is context-dependent, such as the cross-entropy loss for image classification.

Model Selection Policy: Given an input x^t, the model selection policy decides whether to use the slow compute model f_{slow}, or the fast model f_{fast}. We assume that the policy has access to the results of the fast model (without this information, the policy would be purely random). Therefore, the problem of model selection is to infer whether or not to *additionally* invoke the slow model to enhance task accuracy if the fast model results are insufficient for a robot's high-level goal. The challenge is that the robot only has access to

the input x^t and the fast model output y^t_{fast} and thus must estimate the accuracy benefit of the slow model *before* invoking it.

Formally, we define the model selection policy as $\pi : \mathbb{R}^n \times \mathbb{R}^m \to \{0,1\}$. Given input x^t and fast model prediction y^t_{fast}, we define the action as $a^t = \pi(x^t, y^t_{\text{fast}})$. The action $a^t = 0$ indicates selecting the fast model f_{fast}, and $a^t = 1$ indicates selecting the slow model f_{slow}. We define the cost associated with each action as $\texttt{cost}(a^t)$:

$$\texttt{cost}(a^t) = \begin{cases} c_{\text{fast}} & \text{if } a^t = 0 \\ c_{\text{fast}} + c_{\text{slow}} & \text{if } a^t = 1 \end{cases}.$$

Reward: To simultaneously achieve high task accuracy while minimizing the cost of compute, we introduce a per-timestep reward. Given input x^t, the output of the fast model y^t_{fast}, and model selection a^t, the corresponding reward is:

$$R^t(a^t) = \begin{cases} -\alpha \texttt{L}(y^t_{\text{fast}}, y^t_{\text{oracle}}) - \beta\, \texttt{cost}(0) & \text{if } a^t = 0 \\ -\alpha \texttt{L}(y^t_{\text{slow}}, y^t_{\text{oracle}}) - \beta\, \texttt{cost}(1) & \text{if } a^t = 1 \end{cases} \tag{5.1}$$

where α, $\beta \in \mathbb{R}_+$ are user-defined weights to balance the emphasis on accuracy and cost. These can be flexibly set by a roboticist given the unique requirements of a high-level task. For example, a fleet of low-power, compute-limited warehouse robots that rarely interact with humans might have a higher emphasis β on cost to minimize how many times they query a shared central server or remote human supervisor. Conversely, robots that operate in safety-critical scenarios will have a much higher emphasis on accuracy given by α.

5.1.1.1 Formal Problem Definition

Given a stream of N inputs, $\{x^1, x^2, \cdots, x^N\}$, our goal is to propose an optimal model selection policy π^*, that provably maximizes the expected cumulative reward:

$$\mathbb{E} \sum_{t=0}^{N} R^t \left(\pi^*(x^t, y^t_{\text{fast}}) \right)$$

Intuitively, π^* achieves the optimal balance between the cost and accuracy over the given period of N time steps. We now formally define the model selection problem.

Problem 2 (Model Selection for Inference)**.** *Given fast model f_{fast}, slow model f_{slow}, loss function $\mathcal{L}(\cdot)$, and model selection cost* $\texttt{cost}(\cdot)$*, find the optimal model selection policy π^*, which maximizes the reward (Equation 5.1) over a finite horizon N:*

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \mathbb{E} \sum_{t=0}^{N} R^t \left(a^t = \pi(x^t, y^t_{fast}) \right).$$

Section 5.1.2 provides our solution to Problem 2.

5.1.1.2 Discussion on the Problem Definition

The model selection problem is broadly applicable in robotics since it is agnostic to the nature of the compute models, the loss function, or even costs. For example, the models could represent small quantized and large, compute-intensive DNNs or even small and large databases or random forests. Further, the costs could represent battery consumption or communication delay or model inference time.

The main challenge of this problem is the limited information available to the selection policy, namely the input x^t, fast model output y^t_{fast}, and the cost function $\texttt{cost}(\cdot)$. The key challenge is to estimate the accuracy of the slow model, f_{slow}, before even invoking it, which motivates our key technical approach to statistically relate both models' accuracy.

5.1.2 An Algorithmic Approach To Model Selection

In this section, we provide an optimal solution to the model selection problem (Problem 2). First, we make the following practically-motivated assumption.

Assumption 1 (Action and State Independence)**.** *Given a model input x^t at any time t, the model selection a^t of policy π does not affect the next robot measurement x^{t+1}.*

Assumption 1 is practical in many robotics scenarios, since a^t is simply a choice of a compute model to process inputs, not a physical *actuation* decision. For example, a robot can run a fast perception DNN on images x^t at every timestep and its choice to optionally consult a slower DNN a^t does not affect the new image observation x^{t+1}, which is instead largely affected by its ego-motion and surroundings. Our assumption

will not hold for fast-moving robots whose control decisions are heavily dependent on the perception model they invoke, which we discuss in our future work.

Theorem 20. *The optimal model selection policy that solves Problem 2 is of the form:*

$$\pi^*(x^t, y^t_{fast}) = 1\left(\frac{\beta}{\alpha} c_{\text{slow}} < \underbrace{\mathbb{E}\left(\mathcal{L}(y^t_{fast}, y^t_{\text{oracle}}) - \mathcal{L}(y^t_{slow}, y^t_{\text{oracle}})\right)}_{\text{task accuracy gain}}\right)$$

Proof. By Assumption 1, the action a^t at every time does not affect the next state x^{t+1}. Thus, given any input x^t, the actions a^t are independent, so to maximize the cumulative reward it suffices to maximize the reward at every time-step independently. Recall from Equation 5.1 that the reward depends on two choices of a^t, that is, $a^t \in \{0,1\}$. Therefore, Problem 2 can be rewritten as:

$$\pi^*(x^t, y^t_{\text{fast}}) = \underset{a^t \in \{0,1\}}{\text{argmax}}\, \mathbb{E}(R^t(a^t))$$

Substituting in the reward definition (Equation 5.1), we see that we should choose the slow model only when the associated reward is higher than continuing with the fast model. Thus, we choose $a^t = 1$ only when:

$$-\alpha\mathbb{E}\big(\mathcal{L}(y^t_{\text{oracle}}, y^t_{\text{slow}})\big) - \beta(c_{\text{slow}} + c_{\text{fast}}) > -\alpha\mathbb{E}\big(\mathcal{L}(y^t_{\text{oracle}}, y^t_{\text{fast}})\big) - \beta c_{\text{fast}}$$

Simplifying, we arrive at the desired result:

$$\pi^*(s^t) = 1\left(\frac{\beta}{\alpha} \underbrace{c_{\text{slow}}}_{\text{extra compute cost}} < \mathbb{E}\big(\underbrace{\mathcal{L}(y^t_{\text{oracle}}, y^t_{\text{fast}}) - \mathcal{L}(y^t_{\text{oracle}}, y^t_{\text{slow}})}_{\text{task accuracy benefit}}\big)\right). \tag{5.2}$$

□

Theorem 20 suggests a simple model selection policy, which estimates the model accuracy gap $\Delta\mathcal{L} = \mathbb{E}\big(\mathcal{L}(y^t_{\text{oracle}}, y^t_{\text{fast}}) - \mathcal{L}(y^t_{\text{oracle}}, y^t_{\text{slow}})\big)$, and only chooses the slow model if the gap is greater than a threshold that depends on the relative compute costs and weights of accuracy via α, β. However, the key challenge is that calculating $\Delta\mathcal{L}$ requires querying the slow model and knowledge of the ground-truth value y^t_{oracle}. We now transition to two practical approaches to directly instantiate the guarantees from Theorem 20 in practice.

First, we note that in many practical deployment scenarios, the ground-truth oracle values are not present. In such practical settings, the more accurate slow model simply serves as the ground-truth, such as when a slow human supervisor makes ground-truth decisions. In the absence of human annotations, a large, compute-intensive DNN can serve as the slow model and ground-truth. Thus, we present the following lemma of Theorem 20.

Lemma 14. *The optimal model selection policy that solves Problem 2, when $y^t_{oracle} = y^t_{slow}$ at all times t is:*

$$\pi^*(x^t, y^t_{fast}) = 1\left(\frac{\beta}{\alpha} c_{\text{slow}} < \mathbb{E}\left[\mathcal{L}(y^t_{fast}, y^t_{slow})\right]\right)$$

Proof. The proof is the same as Theorem 20, where we note that $\mathbb{E}\big(\mathcal{L}(y^t_{\text{oracle}}, y^t_{\text{slow}})\big) = 0$ when the oracle and slow models are identical. □

For evaluation, one can use Lemma 14 as this model selection policy best reflects practical autonomous deployments. The key challenge to directly applying Theorem 20 and Lemma 14 is to accurately estimate the loss between fast and slow models solely using predictions from the fast model. However, we now show one can indeed compute the expected accuracy benefit for a broad class of fast and slow models that are related by provable approximation guarantees. Specifically, in Section 5.1.2.1, we instantiate the guarantees of Lemma 14 to provide a closed-form, analytic model selection policy for linear regression problems. Crucially, we then extend this analysis to DNN inference in Section 5.1.2.2.

Lemma 14 provides a general framework for model selection. For a novel setting, it can be instantiated by selecting the: (i) compute models, (ii) loss function, (iii) compute model cost, and (iv) characterizing the statistical relationship between compute models to derive the selection policy. Section 5.1.2.1 and Section 5.1.2.2 instantiate Lemma 14 for specific cases of Linear Regression and DNN inference.

5.1.2.1 Analytical Results for Linear Regression

We now apply the guarantees from Lemma 14 to an illustrative warm-up example of high-dimensional linear regression. Recall that our challenge is to estimate the expected value of y^t_{slow} from the information available to the selection policy, namely input x^t and fast model prediction y^t_{fast}. To overcome this challenge, we apply results to approximate linear regression models using *coresets* [100], which are importance-ranked

subsets of a large training dataset. Importantly, a model trained on just the coreset will provably approximate the predictions of one trained on the full dataset. For example, a fast model could be trained on only a core-set of local data on-board a robot while a large one could be trained on multiple robots' data in the cloud.

Compute Models: Let the fast and slow compute models f_i be linear regression models $f_i(x) = A_i x + b_i$, where $i \in \{\text{fast}, \text{slow}\}$, $A_i \in \mathbb{R}^{m \times n}$, and $b_i \in \mathbb{R}^{m \times 1}$. We assume the slow model f_{slow} is learned on a full set of training samples from a joint distribution on $\mathcal{X} \times \mathcal{Y}$, while the fast model is only trained on a core-set of the original data.

Loss Function: Let the loss function be the standard L_2 norm loss: $\mathcal{L}(y_1^t, y_2^t) = ||y_1^t - y_2^t||_2^2$; where $y_1^t, y_2^t \in \mathbb{R}^m$. The following coreset guarantees follow from [100]:

Property 1 (Relation between fast and slow models [100])**.** *For all t, given input x^t, denote the compute model outputs as $y_{fast}^t = f_{fast}(x^t)$ and $y_{slow}^t = f_{slow}(x^t)$. Then, there exists an $\varepsilon > 0$ such that:*

$$y_{fast}^t \in \left[y_{slow}^t, (1+\varepsilon) y_{slow}^t \right], \tag{5.3}$$

where $\mathcal{X}$ and $\mathcal{Y}$ are the input and output distributions, meaning $x^t \sim \mathcal{X}$, and $y_{slow}^t, y_{fast}^t \sim \mathcal{Y}$. [100] provides the approximation factor ε based on the relative size of the core-set compared to the full training set.

Property 1 allows us to relate the fast and slow model predictions as:

$$\begin{aligned} y_{\text{slow}}^t \leq y_{\text{fast}}^t \leq (1+\varepsilon) y_{\text{slow}}^t \\ \textit{or,} \quad y_{\text{slow}}^t \in \left[\frac{y_{\text{fast}}^t}{1+\varepsilon}, y_{\text{fast}}^t \right] \end{aligned} \tag{5.4}$$

Thus, the loss function can be upper bounded as:

$$\mathcal{L}(y_{\text{fast}}^t, y_{\text{slow}}^t) \leq \frac{\varepsilon^2 \cdot (y_{\text{fast}}^t)^2}{(1+\varepsilon)^2}. \tag{5.5}$$

Finally, we can use Eq. (5.5) and Lemma 14 to provide a closed-form model selection policy for Problem 2 in the linear regression setting:

$$\pi(x^t, y_{\text{fast}}^t) = 1\left(\frac{\beta}{\alpha} c_{\text{slow}} < \frac{\varepsilon^2 \cdot (y_{\text{fast}}^t)^2}{(1+\varepsilon)^2} \right). \tag{5.6}$$

Note that Lemma 14 provides the *optimal* solution and the above solution is an approximation since we upper-bound the loss function between fast and slow models. However, our subsequent experiments show this is a very tight bound and implementing Section 5.1.2.1 yields very close performance to an unrealizable oracle solution that has perfect knowledge of the fast and slow model predictions.

5.1.2.2 Analytical Results for Deep Neural Networks (DNNs)

We now provide a similar analysis to the linear regression scenario for the important case when a robotic perception DNN has been compressed using recently developed coreset guarantees [98, 99]. Specifically, [98] compresses fully connected DNNs with ReLU activations by targetedly removing weights with low relative importance via coresets. [99] extends this work to convolutional neural networks (CNNs) by using coresets to remove convolutional filters that a prediction is least sensitive to, which enables a compressed DNN to provably approximate its original counterpart.

Compute Models: Let the models f_i: $\mathbb{R}^n \to \mathbb{R}^m$, where $i \in \{\text{fast}, \text{slow}\}$ be DNNs. Both models are trained on a set of samples drawn from a joint data distribution on $\mathcal{X} \times \mathcal{Y}$.

Loss Function: As for linear regression, the loss function is an L_2 norm loss, such as for depth estimation from a perception CNN.

We now use the following guarantees for DNNs.

Property 2 (Relation between fast and slow models)**.** *For all t, given x^t, $y^t_{fast} = f_{fast}(x^t)$, and $y^t_{slow} = f_{slow}(x^t)$, there exist an $\varepsilon, \delta > 0$, such that the following holds [98, 99]:*

$$\mathbb{P}\Big(y^t_{fast} \in \big[(1-\varepsilon)y^t_{slow}, (1+\varepsilon)y^t_{slow}\big]\Big) \geq 1-\delta, \tag{5.7}$$

$$\mathbb{P}\left(y^t_{fast} \in \left[y^t_{slow} - \frac{M}{2}, y^t_{slow} + \frac{M}{2}\right]\right) \leq \delta, \tag{5.8}$$

where $M \geq 0$ is an upper bound on the error described below. ε and δ depend on the extent of DNN compression. Further, input $x^t \sim \mathcal{X}$ and outputs $y^t_{slow}, y^t_{fast} \sim \mathcal{Y}$.

Eq. (5.8) and bound M arise from the observation that in practical engineering scenarios, the outputs of a neural network and thus the loss will be bounded since they have physical meaning. For example, for a regression loss with perception, $M \geq 0$ could be derived from the largest depth-reading a depth sensor can

register. Likewise, for classification, M is naturally bounded by 1 since the outputs y are softmax scores from a cross-entropy loss.

We now use the core-set relationship to analyze Lemma 14 for DNN inference as follows:

$$y^t_{\text{slow}} \in \begin{cases} \left[\frac{y^t_{\text{fast}}}{1+\varepsilon}, \frac{y^t_{\text{fast}}}{1-\varepsilon}\right] & \text{with prob. } \geq 1-\delta \\ \left[y^t_{\text{fast}} - \frac{M}{2}, y^t_{\text{fast}} + \frac{M}{2}\right] & \text{with prob. } \leq \delta. \end{cases} \tag{5.9}$$

Thus, using Eq. (5.9), the loss can be upper bounded as:

$$\mathcal{L}(y^t_{\text{fast}}, y^t_{\text{slow}}) \leq \begin{cases} \frac{\varepsilon^2 \cdot (y^t_{\text{fast}})^2}{(1-\varepsilon)^2} & \text{with probability } (1-\delta) \\ M & \text{with probability } \delta. \end{cases} \tag{5.10}$$

Therefore, the expectation of the loss function is:

$$\mathbb{E}\left[\mathcal{L}(y^t_{\text{fast}}, y^t_{\text{slow}})\right] \leq \delta M + (1-\delta)\left(\frac{\varepsilon^2 \cdot (y^t_{\text{fast}})^2}{(1-\varepsilon)^2}\right). \tag{5.11}$$

Finally, we can apply Equation 5.11 and Lemma 14 to provide a closed-form model selection policy for Problem 2 in the DNN setting:

$$\pi(x^t, y^t_{\text{fast}}) = 1\left(\frac{\beta}{\alpha} c_{\text{slow}} < \delta M + (1-\delta)\frac{\varepsilon^2 \cdot (y^t_{\text{fast}})^2}{(1-\varepsilon)^2}\right) \tag{5.12}$$

As for linear regression, we emphasize Lemma 1 is optimal and Eq. 5.12 is an approximation since we are *bounding* the expectation using the core-set guarantee. However, our experiments show that implementing Eq. 5.12 as a proxy for Lemma 1 works well in practice. More broadly, we emphasize that core-set guarantees are simply one way to instantiate the general policy provided in Lemma 1. For example, a roboticist could also use other practically-relevant models such as random forests or even approximate databases if they can reliably relate fast and slow model accuracy.

5.1.3 Application Scenarios

We now describe example application scenarios of high-dimensional linear regression, DNN inference, and reachable set computation for a simulated Mars Rover to demonstrate the theoretical guarantees from Section 5.1.2.

5.1.3.1 Linear Regression

Using the analytical results from Section 5.1.2.1, we demonstrate our model selection policy by simulating it on a toy example of linear regression. Let $f_{\text{slow}} : \mathbb{R}^4 \to \mathbb{R}^4_{[0,1]}$ be any general-purpose linear regression model. The amount of time f_{slow} takes to generate an output is 2.5 seconds. Using coresets, we compress the linear regression model f_{slow}, to a faster linear regression model f_{fast}. The f_{slow} model takes 1 second to generate an output. The compression is such that the relation in Equation 5.3 holds with $\varepsilon = 0.1$.

We chose $\alpha = 1$ and $\beta = 0.003$ to emphasize accuracy over compute efficiency in the simulations, although α, β can be flexibly set by a user. We implement the model selection policy as in Section 5.1.2.1, and demonstrate its performance in Section 5.1.4 against benchmark policy selection algorithms (discussed in Section 5.1.3.4).

5.1.3.2 Compute Efficient Robotic Perception

We now stress-test our algorithm on a scenario, inspired by [101], where an aircraft must autonomously track a runway center-line using a wing-mounted camera for state estimation. This scenario, henceforth referred to as the TaxiNet scenario as per [101], uses a DNN to map from camera images to an estimate of the aircraft's lateral distance from the runway center-line d and heading angle θ, which are linearly combined to create the aicraft's steering control. We chose the TaxiNet scenario since the central idea is broadly applicable to resource-constrained robotics, such as low-power drones that use efficient vision models to estimate their real-time pose relative to a landing site.

ResNet-18 DNN [102] was trained to serve as the slow perception model f_{slow} using over 50K images from the standard X-Plane simulator [103] using a publicly-available dataset [104]. The ResNet-18 achieved a low MSE loss of 0.038 on an independent test dataset of 18,372 images, where each image took 0.17 seconds for inference on a CPU. f_{slow} was compressed to yield a quantized ResNet-18 as f_{fast}, which was 47.21% faster but had a 64% higher loss, illustrating a clear need for model selection.

For the TaxiNet scenario, we chose the model costs of $c_{\text{slow}} = 0.017$ and $c_{\text{fast}} = 0$ based on their relative inference times and $\alpha = 1$, $\beta = 3 \times 10^{-4}$ to emphasize safety (low loss) over compute costs. Our model selection results are demonstrated in Section 5.1.4 along with example DNN predictions from aircraft images in Fig. 5.4 (Right).

5.1.3.3 Reachable Set Computation

In this section, we apply the model selection policy to safety assessment for robot navigation. Consider a robot, such as a Mars rover, navigating an unexplored environment. The robot has to assess whether its maneuvers are safe while considering environment uncertainties such as the coefficient of friction, wind disturbances, etc.. This is done by computing a *reachable set*, a set that contains all the states a rover can potentially reach. The robot can make the maneuver safely if the reachable set does not overlap with any obstacles. The reachable set computed by the fast compute model has confidence that is an order of magnitude lesser than the reachable set computed by the slow model.

We assume that the closed loop dynamics of the robot is given as a nonlinear system. For computing the reachable sets, we approximate the nonlinear dynamics locally as an uncertain linear system (see Chapter 2), where the coefficients in the dynamics belong to a bounded range.

Recently, a statistical approximation of the reachable set has been presented in [19]. The confidence of the statistical approximation can be tuned by the user according to her performance and accuracy requirements. Leveraging the flexibility of this statistical approach, we generate a fast compute model which has medium confidence and slow model that has high confidence over the computed reachable sets. Given the various constraints on robot resources, the model selection policy should invoke the appropriate compute model to guarantee safety while minimizing the cost.

Formally, consider a linear dynamical system with uncertainties, represented as $x^+ = \Lambda x$, where $\Lambda \subset \mathbb{R}^{n \times n}$ is an uncertain dynamical matrix. The reachable set of the current state x up to a time horizon t, is denoted as $\texttt{RS}(\Lambda, x, t)$. Though the dynamics is given as $x^+ = \Lambda x$, it can encompass the open loop behavior $x^+ = \Lambda_A x + \Lambda_B u$ (where Λ_A, Λ_B are uncertain matrices), if a control sequence u is provided. In such cases, the uncertain matrices Λ_A, Λ_B are combined together to Λ by concatenating the state and open-loop control.

A system is unsafe if the reachable set intersects with the unsafe set, such as obstacles. That is, given an unsafe set $U \subset \mathbb{R}^n$, a system is unsafe if and only if $\texttt{RS}(\Lambda, x, t) \cap U \neq \emptyset$. Given $\mu > 0$ and a set $S \subseteq \mathbb{R}^n$, we denote the uniform expansion (*bloating*) of set S by μ as $B_\mu(S)$.

We now formally present the model selection problem for safety assessment of robot navigation.

Computation Models: The compute models $i \in \{\text{fast}, \text{slow}\}$, denoted as $\texttt{RS}_i(\Lambda, x, t)$, compute approximations of the reachable set of an uncertain linear system defined by Λ [19]. The statistical guarantee $\mathcal{G}_i$ associated with $\texttt{RS}_i$ is as follows:

$$\mathcal{G}_i: \text{ for any } A \in \Lambda,\ \mathbb{P}\big(\texttt{RS}_i(A, x, t) \subseteq \texttt{RS}_i(\Lambda, x, t)\big) \geq p_i$$

$\mathcal{G}_i$ has a *type I error* of δ_i. Here, the confidence $p_i \in \mathbb{R}_{[0,1]}$ and allowable type I error δ_i are user-given parameters to the models. Intuitively, $\mathcal{G}_i$ means the probability that the reachable set of any sample dynamics is contained within the reachable set $\texttt{RS}_i(\cdot)$ is at least probability p_i. Computing high-confidence approximations of the reachable set requires more statistical samples and therefore a higher computational time and cost. In particular, the required confidence p_i set by a user for statistical guarantee $\mathcal{G}_i$ is directly proportional to the required number of samples. Thus, we set the slow model to be a high-confidence reachable set and the fast model to be a lower-confidence approximation, so $p_{\text{slow}} > p_{\text{fast}}$, $\delta_{\text{slow}} \leq \delta_{\text{fast}}$, and therefore $c_{\text{slow}} > c_{\text{fast}}$. We denote the outputs of the fast and slow models as $y^t_{\text{fast}} = \texttt{RS}_{\text{fast}}(\Lambda, x, t)$ and $y^t_{\text{slow}} = \texttt{RS}_{\text{slow}}(\Lambda, x, t)$.

The crux of our selection policy is that we can relate the reachable sets returned by both models by a factor of ε:

Property 3 (Relationship between fast and slow models)**.** *Given Λ and θ, for all t, there exists an ε such that:*

$$B_{1-\varepsilon}\big(\mathcal{RS}_{slow}(\Lambda, x, t)\big) \subseteq \mathcal{RS}_{fast}(\Lambda, x, t) \tag{5.13}$$

$$\textit{or,} \quad \mathcal{RS}_{\text{slow}}(\Lambda, x, t) \subseteq B_{\frac{1}{1-\varepsilon}}\big(\mathcal{RS}_{\text{fast}}(\Lambda, x, t)\big). \tag{5.14}$$

In a calibration dataset, we can compute the fast and slow model reachable sets for all time steps. Then, we can set ε to be the minimum factor to bloat the robot's set such that the bloated version over-approximates the slow model's reachable set at all times. Thus, a robot can quickly run the fast model, bloat it by $\frac{1}{1-\varepsilon}$, and continue planning if the *bloated* set does not intersect an unsafe region, as formalized below.

Loss Function: Given the safety-critical nature of navigation, the loss is 0 when the reachable set doesn't intersect an unsafe set and ∞ otherwise. Defining the reachable sets used to compute intersections with obstacles as $\Theta_{\text{fast}} = B_{\frac{1}{1-\varepsilon}}(y^t_{\text{fast}})$ and $\Theta_{\text{slow}} = y^t_{\text{slow}} = \mathtt{RS}_{\text{slow}}(\Lambda, x, t)$, the loss for any model $i \in \{\text{fast}, \text{slow}\}$ is:

$$\mathcal{L}(\Theta_i) = \begin{cases} 0 & \Theta_i \cap U = \emptyset \\ \infty & \text{otherwise.} \end{cases} \tag{5.15}$$

Finally, using Property 3 and Lemma 14, the model selection policy that solves Problem 2 for safety assessment during robot navigation is:

$$\pi^*(y^t_{\text{fast}}) = 1\left(B_{\frac{1}{1-\varepsilon}}(y^t_{\text{fast}}) \cap U \neq \emptyset\right). \tag{5.16}$$

Intuitively, the above policy exploits the relationship between fast and slow models by first bloating the fast model's reachable set by a factor of $\frac{1}{1-\varepsilon}$ to create a guaranteed over-approximation of the slow model's reachability computation. If the over-approximation does not intersect obstacles, we are guaranteed safety and simply proceed. If not, we need to invoke the slow model to assess its higher-fidelity reachable set and re-plan a trajectory if it indicates unsafety. While we implemented our policy with $\alpha = 0.7, \beta = 0.3$ to prioritize safety, safety is also heavily emphasized in the loss function (Eq. 5.15) since the penalty is ∞ for collisions.

5.1.3.4 Benchmark Algorithm

We evaluate the performance of our model selection policy against the following benchmark policies:

`Fast`: This policy always uses the fast model with prediction y^t_{fast} for all t.

`Slow`: This policy always uses the slow model with prediction y^t_{slow} for all t.

`Random`: The robot randomly chooses between the fast and slow model with equal probability.

`Our Selector`: This represents our model selection policy from Section 5.1.2.1, Eq. (5.12), and Section 5.1.3.3.

`Oracle`: This strategy assumes that the slow model's output is available to the model selection function at the time of inference. Thus, this strategy only selects the slow model when that decision has a better reward

than using the fast model. The oracle is an upper-bound, unrealizable strategy since it assumes privileged knowledge of the slow model.

5.1.4 Evaluation

The principal objective of our evaluation is to show that our model selection policies from Lemma 14 and Section 5.1.2.1, Eq. (5.12), and Section 5.1.3.3 achieve a significantly higher reward than benchmark model selection policies. Further, we show how thus policy achieves better accuracy with a lower cost than competing benchmarks on simulations of linear regression, aircraft taxiing with state-of-the-art DNN perception models, and rover navigation with real Martian terrain data. All the code (in Python) and models are publicly available at [105].

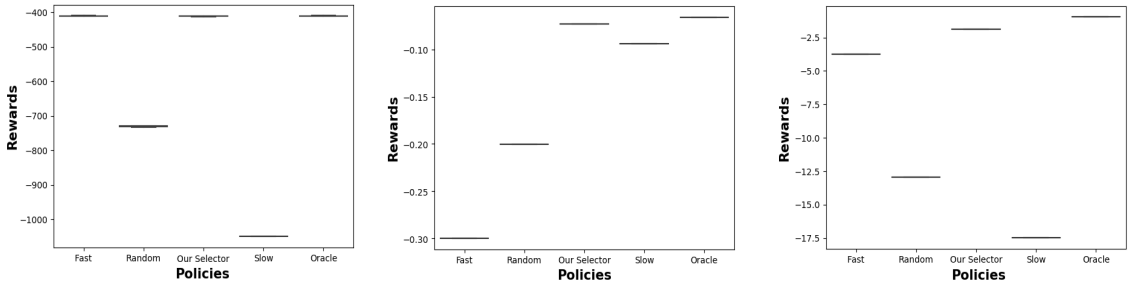


Figure 5.2: Rewards. (*Left: Linear Regression, Center: DNN, Right: Mars Rover Reachable Set*). We illustrate the cumulative rewards (Eq. 5.1) gathered by various policies on the Linear Regression, DNN perception (TaxiNet), and Mars Reachable Set scenarios, respectively. Clearly, our policy (`Our Selector`) achieves the maximum reward compared to other realizable benchmarks and is close to the oracle in all cases.

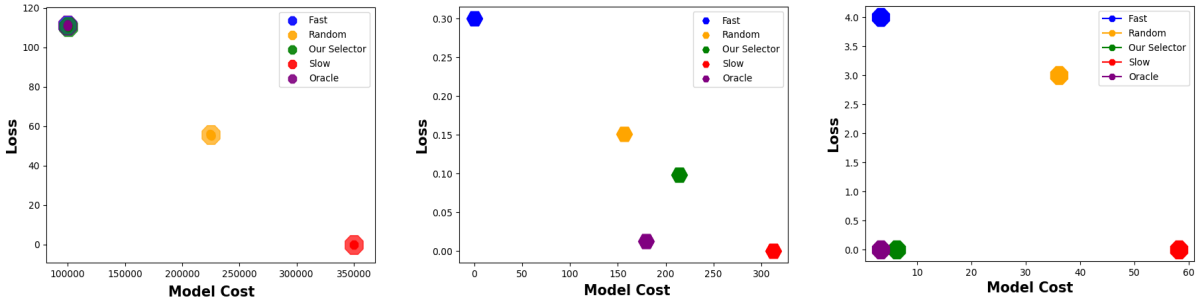


Figure 5.3: DNN Cost vs. Accuracy. (*Left: Linear Regression, Center: DNN, Right: Mars Rover Reachable Set*). Cost vs. Loss trade-off achieved by various model selection policies on all scenarios. In all cases, we observe that the `Fast` policy has low cost but low accuracy, `Slow` has high accuracy but high cost, and `Random` lies sub-optimally in the middle (with a high variance). Only the selection policy proposed in this paper (`Our Selector`) achieves a delicate balance by exploiting the statistical relationship between models to intelligently consult the slow model.

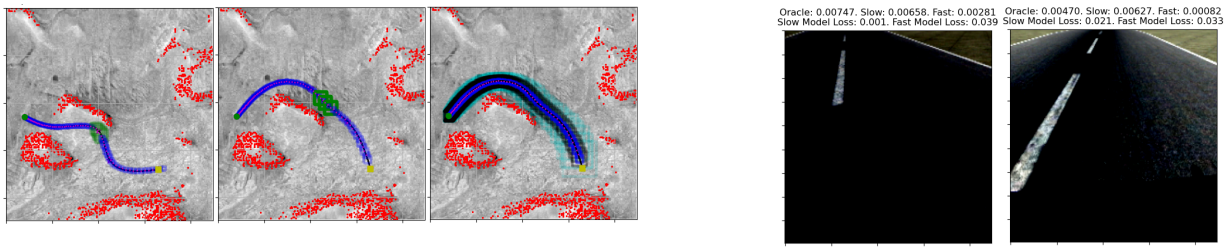


Figure 5.4: Left: (Safe Navigation of a Mars Rover). We consider navigation of a simulated rover on real Mars HiRise terrain data [2], where the red point clouds are obstacles indicating regions of high elevation, as processed in [3]. *First two images:* We show the reachable sets of two different possible routes taken by the Mars Rover. The reachable set in green is computed by the slow model, whereas the one in blue comes from the fast model. Path safety is determined by assessing if the reachable set (accounting for uncertainties), intersects red obstacles. Clearly, our model selection policy uses the slow model only when the rover makes tricky maneuvers. Otherwise, when the rover is far from obstacles, the fast model is sufficient to determine safety, allowing us to maintain high safety with a lower compute time. *Third image:* We show the relationship beween reachable sets computed by the fast model (in blue) and the slow model (in black). Crucially, our policy uses the over-approximated reachable set of the slow model as computed by the fast model (in cyan), that is, $B_{\frac{1}{1-\varepsilon}}(y^t_{\text{fast}})$. Clearly, our model selection policy only consults the slow model when it suspects a possible collision when the set represented in cyan intersects with a red obstacle. Our policy always led to safe, collision-free, efficient navigation by exploiting the relationship between fast and slow models. **Right: (Aircraft TaxiNet DNN Output)**. The output of the fast and slow DNN models for a ResNet-18 TaxiNet model. Given an image, the final output shown is the rudder control.

5.1.4.1 Linear Regression Results

We now evaluate our selection policy for linear regression, as described in Section 5.1.2.1 and Section 5.1.3.1. The key highlight is that our policy achieves 245.4% higher reward than benchmarks in 100 trials, each of duration $N = 10^5$ timesteps with stochastic Gaussian inputs x^t. Fig. 5.2 (Left) and Fig. 5.3 (Left) show the cumulative rewards and trade-off between accuracy and cost, respectively, of all algorithms.

5.1.4.2 Deep Neural Networks (DNN)

We now evaluate the model selection policy for the TaxiNet aircraft taxiing scenario from Section 5.1.3.2. The key result on 18,372 *test* images is shown in Figure 5.2 (Center), where the proposed policy (`Our Selector`) achieves 22.22% higher reward than competing benchmarks and is within 10.18% the performance of an upper-bound `Oracle`. Moreover, Fig. 5.3 (Center) shows that our model selection policy achieves low loss with low cost unlike competing policies. This is because our policy leverages the statistical correlation between models to mostly rely on the fast model to reduce cost, but also opportunistically queries the slow model for higher accuracy. However, the proposed policy is careful to only invoke the slow, accurate model when there is a substantial accuracy gain, leading it to be queried only 68.6% of the time.

5.1.4.3 Reachable Set Computation

We now demonstrate the performance of the model selection policy (Section 5.1.3.3) to determine the safety of a simulated Mars Rover navigating steep obstacles on terrain from NASA's HiRise Dataset [2, 3]. A low-power rover must always be safe, but also fast and compute-and-power-efficient while accounting for reachable sets while planning.

The rover is assumed to follow a linearized bicycle model with bounded perturbations in the dynamics matrix for yaw angle. Given an intended path, we use the model selection policy (Section 5.1.3.3) to determine safety given uncertain dynamics while minimizing compute time. Specifically, given a start set, desired goal, and a set of way-points, a reference trajectory is computed using a cubic spline planner, which is followed using Model Predictive Control (MPC). Using the planned states x and controls u at every time, our model selection policy must determine the trajectory's safety by invoking either a fast or slow reachable set computation model as described in Section 5.1.3.3.

Fig. 5.4 (Left, first two images) shows how the proposed policy (Section 5.1.3.3) safely, but efficiently, follows two different paths near a red obstacle indicating an unsafe terrain gradient above 20 degrees. The key benefit of the proposed approach is that the robot mostly uses the fast reachable set computation (blue) for high-efficiency and only *intelligently* consults the higher-fidelity slower model during tricky turns close to an obstacle. Indeed, Fig. 5.4 (Left, third image) precisely shows how the proposed policy (Equation 5.16) exploits the relationship between fast and slow models to selectively query the slow model only when required during key turns. The fast model's reachable set result is in blue, the slow model's result is in black, and the *over-approximation* from bloating the fast model's result by $B_{\frac{1}{1-\varepsilon}}(y^t_{\text{fast}})$ is in cyan.

Clearly, even the over-approximation rarely intersects unsafe obstacles and it is only necessary to consult a fine-grained result from the slow model (black) when the over-approximation is too conservative and needs to be refined. In all scenarios, we rigorously verified the simulated rover is safe and never hits an obstacle despite dynamics uncertainties. Fig. 5.2 (Right) and 5.3 (Right) quantitatively illustrates the superior efficiency and accuracy (safety) of our policy, since it achieves the highest reward, never hits an obstacle, and efficiently only queries the slow model on-demand near critical obstacles.

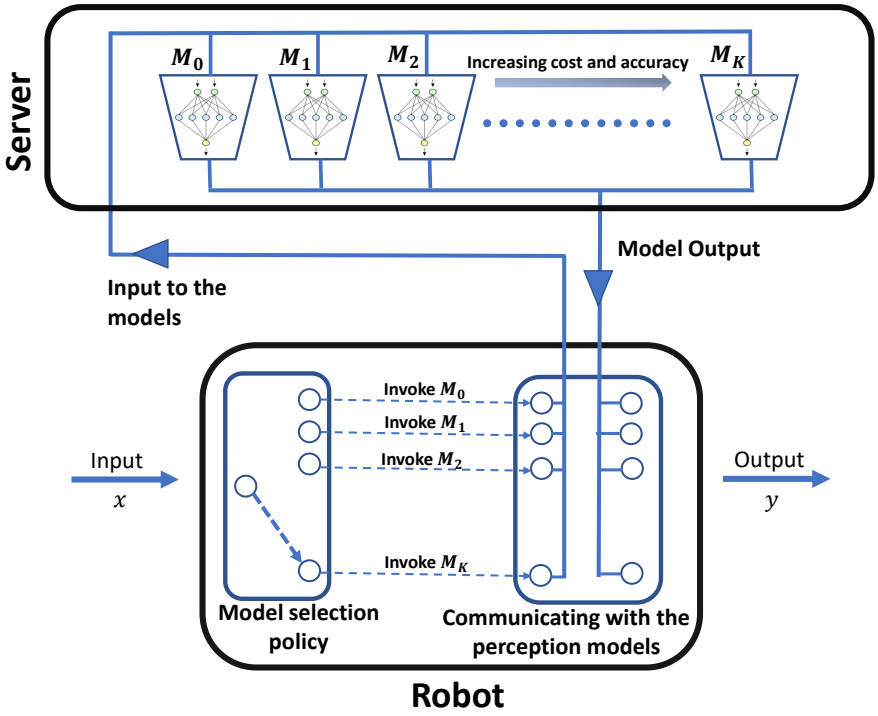


Figure 5.5: Model Selection Problem: In a setting where a robot has access to a plethora of models (namely M_0 to M_K)—with varying cost and accuracy—the goal of the robot is to perform a control task such that it strikes an optimal balance between compute cost and control cost.

5.2 Decision Making in Perception-Based Robots: Selecting from a Variety of Perception Models

A resource constrained robot, required to perform a desired control task, is generally equipped with a low accuracy and computationally fast perception model (local model). However, today different perception models with accuracy and latency trade-offs are being trained. For example, the EfficientNet [106] suite of vision models provides 8 model variants, namely EfficientNet 0 to EfficientNet 7, that trade-off accuracy with model size and latency (ResNet [102] suite is yet another such example). These models increase inference latency and energy consumption, requiring a robot to gracefully balance control cost with perception cost, as shown in Fig. 5.13a. The key desideratum for the robot in a multi-step control task is to leverage the available models to perform the task efficiently. In other words, the robot should accomplish the task by optimally balancing the perception time and control cost. In the rest of the chapter, we refer to perception time as perception cost since it can represent various metrics (such as number of bit-operations performed, time taken to run on a device, *etc*.).

With recent advancements in model training (with accuracy and latency trade-off), a model selection strategy that can accomplish a task by optimally balancing the perception cost and control cost is a necessity in modern day robotic applications. Imagine a scenario where a factory robot is required to perform safe navigation in a busy warehouse floor. To perform the navigation, the robot is equipped with a plethora of perception models that trades-off control cost and accuracy. Without loss of generality, such perception models can reside on the robot, or in a central server ("Cloud") which can be invoked over a shared network. In such scenarios, a desired strategy should be to use the high perception cost models, to ensure minimum control cost, *only* when the robot is required to make tricky maneuvers.

The solution we proposed in this section provably realizes such a policy. Consider a solution using the computationally fastest perception model among all the available models (say EfficientNet 0) at all time steps: this will lead to a solution that takes minimum possible perception cost but results in worst possible control cost (as decisions based on a low-accuracy model might lead to states far away from the optimal states *vis-à-vis* control cost). Similarly, a solution using the most computationally expensive perception model (say EfficientNet 7) can result in a least control cost solution but maximum perception cost. Clearly, for any practical scenario, such simple solutions do not work. A desired solution, on the other hand, uses computationally expensive perception models only when computationally faster models fail to produce the desired result. Further, the relationship, with respect to compute cost and accuracy, between the available plethora of models is highly nonlinear [106]. We refer to this as the *model selection problem*, illustrated in Fig. 5.5.

Adding to the challenges, the choice of perception model at the current step impacts the *future* states reached by the robot. In other words, selecting a particular perception model at this step may cause the robot to reach a state from which, regardless of the perception model choices made in the future, all states the robot reaches will have a high control cost—resulting in a sub-optimal outcome. Clearly, a greedy approach that only considers the information available at the current step (as in [107]) does not result in an optimal solution in a general setting; a setting considered in this section. We note that obtaining information about the future states would require information about the future environment states (*i.e.,* inputs to the perception models). While obtaining information about the exact future environment is impossible, in a practical setting, an empirical observation of the past data can however give us *some idea* of the future environment states. Formally, a probability distribution of the future environment states can be obtained using empirical evidence (such as historical data). Note that provable optimality cannot be guaranteed in a general

setting that assumes *zero-knowledge* of future environment states (unless assumptions are imposed). In this section, we propose a provable optimal solution that outperforms current model selection approaches and achieves a performance that is very close to the (unrealistic) theoretical bound, assuming only the knowledge of probability distribution (*i.e.* mean and variance) of the future environment states.

Most optimal control only penalizes control cost and tracking error, while remaining oblivious to the perception cost. The key idea behind our solution is to compute the sensitivity of control cost to perception errors, which in turn guides which model to invoke since we not only care about conventional control cost but also perception latency. In particular, we provide a provably optimal solution to the model selection problem with a plethora of perception models (any number of models) with varying control cost and perception cost. That is, we compute a sequence of models that should be invoked, at each time step, to achieve optimality with respect to control cost and perception cost. This is achieved by encoding the model selection sequence as a Mixed Integer Quadratic Program (MIQP) that minimizes a linear combination of the control cost and the perception cost, leveraging results from [108]. Intuitively, the integer variables, in the MIQP formulation, model the discrete available choices of the perception models, and the objective function is quadratic primarily due to the control cost being a quadratic function of the robot states. Further, we prove a polynomial time complexity for our proposed solution for special subcases. The solution also indicates the importance of variance (not just mean accuracy) of the perception models when considering a multi-step decision problem; where in contrast, most works on model latency accuracy trade-offs largely look at the single step vision tasks and report mean accuracy.

We demonstrate the proposed solution on a drone landing sequence, using visual navigation, simulated on a photo-realistic simulator named AirSim [109]. In the experiments, the drone is equipped with all the EfficientNet models to compute the altitude of the drone from the helipad. These models have associated perception errors (in estimating the altitude) due to uncertainty in DNNs. The drone landing controller issues input based on the altitude computed by the given EfficientNet model (where the model, at a given time step, is selected as per the policy). The experiments have shown that we can achieve 38.04% lower control cost in 79.1% less perception time than other available policies. For the rest of the section, we refer to perception models simply as models.

Contributions: Given prior works, mainly [90, 107], the contributions proposed in this section are as follows:

1. Prior works have only considered the model selection problem with only two models. [90] uses an RL based approach, and [107] (discussed in Section 5.1) provides an interpretable solution when some restrictive assumptions hold true. In contrast, this section provides a provable optimal solution to the model selection problem, with any number of perception models, for multistep decision making to achieve a desired a control task, even when the input to the perception models are not independent and identically distributed.

2. [90] uses RL based approach, and [107] uses statistical correlation between the models primarily using model compression. In contrast, our solution is fundamentally different from both these approaches. Leveraging recent results from [108], we cast the model selection problem as an optimization problem (MIQP), and provide a provable optimal solution.

3. This solution indicates the importance of variance, not just mean accuracy of the models. Further, our work also allows the user to use a plethora of compute models, such as EfficientNets [106] and ResNets [102], to achieve a goal in an optimal fashion.

4. We show that a special sub-case of the model selection problem has polynomial time complexity.

5. We illustrate this approach on a photo-realistic drone landing sequence simulated in AirSim. The results of our experiments clearly show how our approach outperforms other policies.

5.2.1 Problem Statment

In this section, we define the problem statement formally. Consider the following dynamics:

$$x_{t+1} = Ax_t + Bu_t + Cs_t, \tag{5.17}$$

where, at time step t, x_t denotes the state reached by the robot, u_t denotes the controller input to the robot, and s_t denotes the *perception input*. We note that, in the dynamics given in Eq. (5.17), the state of the robot x_t is therefore dependent on the perception inputs to the robot at previous time steps. Though a linear dynamics, as in Eq. (5.17), is restrictive, but it provides a good first step especially with linearized robot dynamics for common robots. Further, the additive Cs_t term can seem restrictive but indeed practical, especially while encoding exogenous sensor measurements. Imagine a drone landing sequence that relies on a DNN based perception model to compute the altitude of the drone which is used by the controller to issue control inputs to the drone. The DNN estimates the altitude of the drone with some error due to the

inherent uncertainties of the model. In such cases, the Cs_t term is used to model the measurement errors from the DNN. It is also worth noting that since the states of the robots are dependent on the perception inputs, it is extremely important to choose perception models that estimates the inputs appropriately. An improper estimation of the current perception input might lead the system to a state from which all the states reached by the system, regardless of the choices of future perception inputs, will have a very high control cost.

Robot State: Given the dynamics as in Eq. (5.17), the set of states reached by the robot from an initial set $x_0 \in \mathbb{R}^n$ at time step t is given by $x_t \in \mathbb{R}^n$, where $A \in \mathbb{R}^{n \times n}$.

Perception Input: The perception input to the robot at time step t is given by $s_t \in \mathbb{R}^p$, where $C \in \mathbb{R}^{n \times p}$. Let $\mathbf{s} \in \mathbb{R}^{pH \times 1}$ denote the vector of all perception inputs up-to time step $H-1$; that is: $\mathbf{s} = \begin{bmatrix} s_0 & s_1 & \dots & s_{H-1} \end{bmatrix}^\top$.

Controller Input: Given a perception input vector $\mathbf{s}$, the controller input to the robot at time step t is given by $u_t = \pi\big(x_t, s_{t:H-1}, \theta_c\big)$, where π is the policy providing the control inputs, with the given parameter θ_c (such as dynamics matrix, feedback matrix *etc.*) [108]. Let $\mathbf{u} = \mu(\mathbf{s})$ denote the vector of all control inputs, for a given perception input vector $\mathbf{s}$, up-to time step $H-1$; that is: $\mathbf{u} = \mu(\mathbf{s}) = \begin{bmatrix} u_0 & u_1 & \dots & u_{H-1} \end{bmatrix}^\top$, where $u_t = \pi\big(x_t, s_{t:H-1}, \theta_c\big)$. We note that the control input u_t is dependent on the current state (x_t) and the future perception inputs ($s_{t:H-1} = \{s_t, s_{t+1}, \cdots, s_{H-1}\}$). To be able to compute the control inputs that would make the system stable, it needs the information about what it sees as perception input in the future as well. The closed form expression for this can be found in [108].

5.2.1.1 Perception Model

The true perception input to the robot, at time step t, is given as s_t. But obtaining the true perception input s_t is practically impossible, therefore we use practically implementable perception models to estimate s_t as $\hat{s}_t$. The loss between the true perception input s_t and the estimated perception input is given by $\mathcal{L}(s_t, \hat{s}_t)$. Consider a drone landing sequence where the true perception is the true altitude of the drone (s_t). Since it is impossible to obtain the true altitude of the drone, one can use DNN that takes the image of the helipad from the drone and estimates the altitude ($\hat{s}_t$); this is denotes the estimation of the true altitude. The perception input, at time step t, to the robot comes from the perception model $g_{\text{per}} : \mathbb{Z}_{[0,W-1]} \times \mathbb{Z}_{\geq 0} \to \mathbb{R}^p$, where W is the total number of perception models available. Intuitively, the perception model $g_{\text{per}}(w,t)$ selects one of the available models $0 \leq w \leq W-1$, and returns its estimated perception at time step t. The cost incurred to the robot for invoking the perception model is given by $cost_{\text{per}} : \mathbb{Z}_{[0,W-1]} \to \mathbb{R}$. We note that our cost is

dependent only on the model selected, and not on time step. That is, the perception input at time step t is given by $\hat{s}_t = g_{\text{per}}(w,t)$ for a given *choice of the model* w, and the cost for invoking the model is given by $cost_{\text{per}}(w)$. Intuitively, the model choice w represents the desired quality of output by the user. For example, higher value w incurs a higher cost $\big($*i.e.* higher value of $cost_{\text{per}}(w)\big)$ and better quality of perception input $\big($*i.e.* lower value of $\mathcal{L}(g_{\text{per}}(w,t), s_t)\big)$. Note that the indices of the model, $0 \leq w \leq W-1$, are not arbitrary; they are sorted in a non-decreasing order of their costs. That is, model w_i will have a lower cost than model w_j, if and only if $w_i < w_j$.

We note that $\hat{s}_t$ is used to represent the perception input at time step t; where $\hat{s}_t = g_{\text{per}}(w,t)$ means the perception input is chosen from the w^{th} perception model as per the function $g_{\text{per}}(w,t)$. However, the function $g_{\text{per}}(\cdot)$, for all t and all $w \in \mathbb{Z}_{[0,W-1]}$, is defined independent of s_t as follows:

$$g_{\text{per}}(w,t) = \Gamma_t^w, \tag{5.18}$$

$$cost_{\text{per}}(w) = w \cdot \Upsilon, \tag{5.19}$$

where $w \in \mathbb{Z}_{[0,W-1]}$ represents the model choice variable, $\Upsilon \in \mathbb{R}_{>0}$, and $\Gamma_t^w \in \mathbb{R}^p$. The constant Υ represents the ratio by which the perception cost of the models rises with an increase in index; the value of the constant therefore depends on the application. Though we assume a linear dependence of the perception cost of a model to its index, our results will hold true for any convex quadratic cost function. However, for the sake of simplicity, and since it also models most real-world scenarios, we will be using a linear cost function. Intuitively, Γ_t^0 represents the lowest-quality perception input $\big($*i.e.* highest value of $\mathcal{L}(\Gamma_t^0, s_t)\big)$ with minimum perception cost 0 (at time step t), and Γ_t^{W-1} represents the highest-quality perception input with maximum perception cost $cost_{\text{per}}(W-1) = (W-1)\cdot\Upsilon$ (at time step t). We note that the cost of invoking the model is not merely restricted to its computation cost (*i.e.*, the time it takes to generate the output). The cost can be determined by the application, such as the cost of invoking the server or the bandwidth demand it imposes *etc*. Consider a drone landing scenario that needs to estimate its altitude from the available choices of models that reside in a remote server. In such a case, the cost of invoking a model need not be restricted to its compute cost, it can also mean the amount of data that needs to be sent over the network, or the value assigned to the server depending on its demand, or a rental server, *etc*. In order to maintain the generality of the problem, with regards to it compute cost, we assume that the perception cost has no impact on the control frequency,

and therefore the control performance. While it is worth considering a setup where the perception cost is specifically the compute cost, and thus impacts the control frequency, we are not considering such a setup in our current version due to the following challenges. A recent result that offers a closed form solution of the sensitivity of control cost to perception errors is used in our work [90]. However, no such results exist that considers the sensitivity of control cost to perception errors while taking into account the impact on control frequency. It is also worthwhile to note that if such a closed form can be found, our solution can be readily modified to accommodate such a configuration. Due to the aforementioned problems, we shall defer it for the time being and leave it as a future work.

Let $\mathbf{C}$ be the set of all possible choices. Let $\mathbf{c} = \{w_0, w_1, \cdots, w_{H-1}\} \in \mathbf{C}$ be a set of choices up-to time step $H-1$. Intuitively, $\mathbf{C}$ represents the decision variables in the optimization problem encoding the choice of the models at every time step. For instance, an optimal solution $\mathbf{c}_{\text{opt}} \in \mathbf{C}$ to the optimization problem with $w_t = M$ implies model M (where $0 \leq M \leq W-1$) should be invoked at time step t. Let the perception input vector, corresponding to $\mathbf{c}$, be given as follows $\hat{\mathbf{s}}_{\mathbf{c}} = \begin{bmatrix} s_0^c & s_1^c & \dots & s_{H-1}^c \end{bmatrix}^\top$, where $s_t^c = g_{\text{per}}(w_t, t)$, and the corresponding control vector is given as $\hat{\mathbf{u}}_{\mathbf{c}} = \mu(\mathbf{s}_{\mathbf{c}})$. The optimal control vector $\mathbf{u}^* = \mu(\mathbf{s})$. Intuitively, $\hat{\mathbf{s}}_{\mathbf{c}}$ denotes the (estimated) perception inputs corresponding to the model choices $\mathbf{c}$ up to time step $H-1$. The control vector $\hat{\mathbf{u}}_{\mathbf{c}}$ is the set of control inputs to be applied to the system that is computed using the perception inputs $\hat{\mathbf{s}}_{\mathbf{c}}$, while $\mathbf{u}^*$ denotes the optimal control inputs (that would have been applied if the true perceptions inputs were known).

5.2.1.2 Perception Cost and Control Cost

Given a set of choices $\mathbf{c} = \{w_0, w_1, \cdots, w_{H-1}\}$, and an initial set x_0, the control cost $\mathcal{J}^{ctrl}(\mathbf{c})$ is given as [108]:

$$\mathcal{J}^{\text{ctrl}}(\mathbf{c}) = J^c(\hat{\mathbf{u}}_{\mathbf{c}}, \mathbf{s}, x_0) = c_{H-1}(x_{H-1}) + \sum_{t=0}^{H-1} c(x_t, \hat{u}_t), \tag{5.20}$$

where $c(\cdot)$ is the stage cost and $c_{H-1}(x_{H-1})$ is the terminal cost. Further, we define the perception cost $\mathcal{J}^{per}(\mathbf{c})$ as follows:

$$\mathcal{J}^{\text{per}}(\mathbf{c}) = \sum_{t=0}^{H-1} cost_{\text{per}}(w_t). \tag{5.21}$$

Given a set of (model) choices $\mathbf{c}$, $\mathcal{J}^{\text{ctrl}}(\mathbf{c})$ (Eq. (5.20)) denotes the cumulative control costs of the states traversed by the system due to the model choices $\mathbf{c}$, while the $\mathcal{J}^{\text{per}}(\mathbf{c})$ (Eq. (5.21)) is the cumulative cost

incurred by the system for invoking the models as per $\mathbf{c}$. Intuitively, α and β denotes the user's priority on control cost and perception cost as per the application. If the application prioritizes control cost more than perception cost, α and β will be chosen such that $\alpha > \beta$ (say, $\alpha = 0.8$ and $\beta = 0.2$). In other words, $\mathcal{J}^{\text{tot}}(\mathbf{c})$ represents the weighted sum of the cumulative control cost and the cumulative perception cost as per the model choices $\mathbf{c}$.

5.2.1.3 Model Selection Problem

Using the aforementioned artifacts, we now formally state the model selection problem as follows:

Problem 3 (Model Selection Problem)**.** *Given a system with dynamics as in Eq.* (5.17)*, and perception model* $g_{per}(\cdot)$ *with cost* $cost_{per}(\cdot)$*, compute a set of choices* $\mathbf{c}_{opt}$ *for H time steps, such that, for a given* α, β *and* $x_0 \in \mathbb{R}^n$*:*

$$\mathbf{c}_{opt} = \arg\min \mathbf{c} \in \mathbf{C}\Big\{\mathcal{J}^{tot}(\mathbf{c})\Big\}. \tag{5.22}$$

Intuitively, we want to find a set of choices, $\mathbf{c}_{\text{opt}}$, such that the total cost $\mathcal{J}^{\text{tot}}$ (an weighted sum of the control cost and the perception cost) is minimized.

Reformulating Problem 3. Now we reformulate Problem 3 using results from [108]. The reason to reformulate Problem 3 is three-fold:

1. In Problem 3, the solution is tied to an initial set x_0. In this reformulation, we propose a formulation that is independent of the initial state.
2. Leveraging prior results, we will propose a solution that can be formulated as an optimization problem. Such a formulation helps us in computing the optimal model selection sequence efficiently using the off the self optimization solvers.
3. We will further show that for special cases we can solve the problem in polynomial time complexity.

A re-formulation of Problem 3 can be given as follows:

$$\min_{\mathbf{c}\in\mathbf{C}} \quad \alpha \cdot \mathcal{J}^{\text{ctrl}}(\mathbf{c}) + \beta \cdot \mathcal{J}^{\text{per}}(\mathbf{c}) \tag{5.23}$$

$$\iff \quad \min_{\mathbf{c}\in\mathbf{C}} \quad \underbrace{\alpha \cdot \big(J^c(\hat{\mathbf{u}}_{\mathbf{c}}, \mathbf{s}, x_0)\big)}_{\text{control term}} + \underbrace{\beta \cdot \mathcal{J}^{\text{per}}(\mathbf{c})}_{\text{perception cost term}} \tag{5.24}$$

The minimum possible control cost is achieved when the control inputs are computed with the true perception input $\mathbf{s}$. In other words, the control term in Eq. (5.24) is minimal when $\hat{\mathbf{s}} = \mathbf{s}$ (therefore, $\hat{\mathbf{u}}_{\mathbf{c}} = \mathbf{u}^*$). Since it is impossible to know the true perception inputs ($\mathbf{s}$), we want to estimate them ($\hat{\mathbf{s}}$) as closely as feasible to the control cost that corresponds to the true perception inputs. Formally, for any set of choices $\mathbf{c} \in \mathbf{C}$, the following relation is assumed to hold: $J^c\left(\mathbf{u}^*, \mathbf{s}, x_0\right) < J^c(\hat{\mathbf{u}}_{\mathbf{c}}, \mathbf{s}, x_0)$. Therefore, the re-formulation given in Eq. (5.23) can be further rewritten as:

$$\min_{\mathbf{c} \in \mathbf{C}} \quad \alpha \cdot \left(J^c(\hat{\mathbf{u}}_{\mathbf{c}}, \mathbf{s}, x_0) - J^c\left(\mathbf{u}^*, \mathbf{s}, x_0\right) \right) + \beta \cdot \mathcal{J}^{\text{per}}(\mathbf{c}) \tag{5.25}$$

Intuitively, we can rewrite Eq. (5.24) as Eq. (5.25) due to the following reasons. Note that, the control cost is minimum when the optimal set of control inputs $\mathbf{u}^*$ is used. Therefore, in this optimization formulation, instead of finding a set of models ($\mathbf{c}$) that minimizes the control cost directly (corresponding to $\mathbf{c}$), we rewrite it to find a set of models such that the difference between the control cost pertaining to the optimal control and the control cost pertaining to the models ($\mathbf{c}$) is minimized. Clearly, Eq. (5.24) and Eq. (5.25) are equivalent. The reason for this algebraic manipulation is, however, to be able to use a recent result that provides a closed form of the manipulated term—this shall be discussed next.

Given an arbitrary perception input vector $\hat{\mathbf{s}}$, [108] provides a closed form solution to $J^c\left(\mu(\hat{\mathbf{s}}), \mathbf{s}, x_0\right) - J^c\left(\mathbf{u}^*, \mathbf{s}, x_0\right)$. More specially, [108] provides a closed form expression for $J^c\left(\mu(\hat{\mathbf{s}}), \mathbf{s}, x_0\right) - J^c\left(\mathbf{u}^*, \mathbf{s}, x_0\right)$ in terms of $\hat{\mathbf{s}}$ and $\mathbf{s}$ (as a quadratic function of the difference between $\hat{\mathbf{s}}$ and $\mathbf{s}$, *i.e.*, a quadratic function of $\hat{\mathbf{s}} - \mathbf{s}$):

$$J^c\left(\mu(\hat{\mathbf{s}}), \mathbf{s}, x_0\right) - J^c\left(\mathbf{u}^*, \mathbf{s}, x_0\right) = (\hat{\mathbf{s}} - \mathbf{s})^\top \times \Psi \times (\hat{\mathbf{s}} - \mathbf{s}), \tag{5.26}$$

where $\Psi = L^\top K^{-1} L$ is a positive semi-definite matrix. [108] provides closed form solutions to compute the matrices L,K, where L and K are only dependent on the dynamics (matrices A, B and C) and cost matrices Q and R. We note that the above term (Eq. (5.26)) is independent of x_0, using this we propose a reformulation of Problem 4 that is independent of the initial set x_0 in the rest this section. Given a set of choices, $\mathbf{c} \in \mathcal{C}$, recall that the perception input vector corresponding to $\mathbf{c}$ is $\hat{\mathbf{s}}_{\mathbf{c}}$. Using Eq. (5.26), the optimization formulation in Eq. (5.25) can be equivalently rewritten as:

Problem 4 (Model Selection Problem)**.** *Given a system with dynamics as in Eq.* (5.17)*, and perception model* $g_{per}(\cdot)$ *with cost* $cost_{per}(\cdot)$*, compute a set of choices* $\mathbf{c}_{opt}$ *for H time steps, such that, for a given* α, β*:*

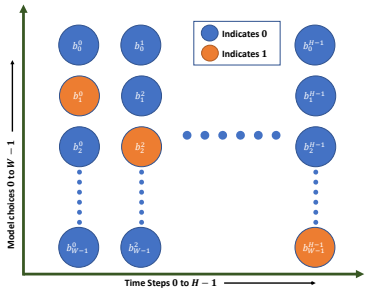
$$\min_{\mathbf{c}\in\mathbf{C}} \quad \alpha \cdot \Big((\hat{\mathbf{s}}_{\mathbf{c}} - \mathbf{s})^{\top} \times \Psi \times (\hat{\mathbf{s}}_{\mathbf{c}} - \mathbf{s}) \Big) + \beta \cdot \mathcal{J}^{per}(\mathbf{c}). \tag{5.27}$$

Theorem 21 (Equivalence)**.** *Problem 3 and Problem 4 are equivalent. [Proof in Section 5.2.6.1]*

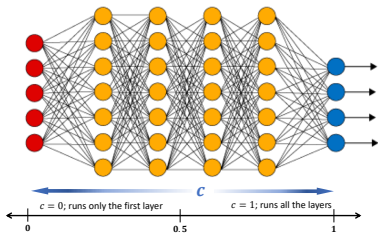
Note that the reformulation in Problem 4 is independent of the initial set x_0. That is, the set of optimal choices $\mathbf{c}_{opt}$, as computed by the solution, is independent of x_0. It is worthwhile to mention that a model selection sequence, that is independent of the initial state, enables an engineer/practitioner to compute the model selection sequence just once and use it anywhere regardless of its initial set. In most practical scenarios, where the initial state cannot be known apriori, our solution is extremely effective as the model selection sequence need not be solved from scratch when the initial state changes.

5.2.2 Model Selection Problem When All the Perception Outputs are Known

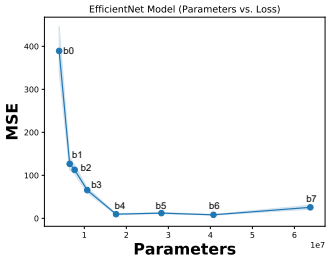
Note that Problem 4 assumes that the difference between the perception model's output $g_{\text{per}}(\cdot)$ and the true perception is known. But in practice, this might not hold true for many cases. In such cases, one can only know a distribution, not the exact difference between the true and perception models' output. In this section, we will propose an exact solution to Problem 4 (assuming the true difference is known), but in subsequent sections we will relax this assumption to handle practical scenarios. Since the solution to the problem is based on an optimization formulation, before we put forward the solution to Problem 4, we wish to layout the model selection encoding. That is, how the perception $\hat{s}_t$ and the cost $cost_t$, at a time step t, can be encoded using binary variables. Recall, that the final output of the model selection problem, encoded as an optimization problem in Eq. (5.27), is the set of model choices ($\mathbf{c}$) such that the total cost (a weighted sum of the control cost and the perception cost) is minimized. To encode the model choices, among 0 to $W-1$ many models, up to time step $H-1$, we will use $W \times H$ many binary variables. For each time step $t \in [0, H-1]$, we will use W many binary variables, denoted as b_i^t, where $i \in [0, W-1]$. If at a given time step t, $b^w = 1$, it implied that model w should be used at time step t. Note that, the encoding will ensure that only one model is chosen at a given time step t, *i.e.*, all other $b^w = 0$, where $i \neq w$. This is illustrated in Fig. 5.6a, and formally described next.



(a) Encoding Binary Variables. The model choices are encoded using $W \times H$ many binary variables, where $H-1$ is time horizon and W is the number of models. For each time step t, W many binary variables are used. At any time step t, if a binary variable $b_w^t = 1$, then model w should be selected at time step t.



(b) Approximating Early-Exit DNN. Approximating the output of an early exit DNN with a continuous variable c that acts as a proxy to represent the number of layers to be run of the DNN.



(c) Accuracy of All the Trained EfficientNet Models. The x and the y axis represents number of parameters in the DNN and the MSE accuracy respectively. This plot shows the average accuracy and 95% confidence interval of all the trained EfficientNet models.

Figure 5.6: Model Selection Encoding and DNN Training

$$\hat{s}_t = \sum_{i=0}^{W-1} b_i^t \cdot g_{\text{per}}(i,t), \tag{5.28}$$

$$cost_t = \sum_{i=0}^{W-1} b_i^t \cdot \Upsilon, \tag{5.29}$$

such that $\sum_{i=0}^{W-1} b_i^t = 1$, where $\forall_{b_i^t \in \mathbb{Z}_{[0,1]}}$. Intuitively, if $b_w^t = 1$ (and rest all 0), it implies that the model $0 \leq w \leq W-1$ has been selected at time step t; *i.e.*, $\hat{s}_t = g_{\text{per}}(w,t)$ with $cost_t = cost_{\text{per}}(w)$. Let $\mathbf{b}$ denote the set of Boolean variables required to encode the model choices up-to time step H. $\mathbf{b} = \{b_0^0, b_1^0, \ldots, b_{W-1}^0, b_0^1, \ldots b_{W-1}^1, \ldots, b_{W-1}^{H-1}\}$. Note that, clearly, $\mathbf{c}$ and $\mathbf{b}$ are equivalent (Section 5.2.6.2). Using this idea, one can encode $\hat{\mathbf{s}}_c$ as follows (simply by replacing the elements $\hat{s}_t$ in $\hat{\mathbf{s}}_c$):

$$\hat{\mathbf{s}}_c = \left[\sum_{i=0}^{W-1} b_i^0 \cdot g_{\text{per}}(i,0) \;\cdots\; \sum_{i=0}^{W-1} b_i^t \cdot g_{\text{per}}(i,t) \;\cdots\; \sum_{i=0}^{W-1} b_i^{H-1} \cdot g_{\text{per}}(i,H-1) \right]^\top. \tag{5.30}$$

Therefore, $\hat{\mathbf{s}}_c - \mathbf{s}$ can encoded and rewritten, in terms of $\mathbf{b}$, as follows:

$$\hat{\mathbf{s}}_c - \mathbf{s} = \mathcal{V}(\mathbf{b}) = \left[\sum_{i=0}^{W-1} b_i^0 \cdot g_{\text{per}}(i,0) - s_0 \;\ldots \right.$$
$$\left. \sum_{i=0}^{W-1} b_i^t \cdot g_{\text{per}}(i,t) - s_t \;\ldots \sum_{i=0}^{W-1} b_i^{H-1} \cdot g_{\text{per}}(i,H-1) - s_{H-1} \right]^\top \tag{5.31}$$

such that $\forall_{0\leq t\leq H-1} \sum_{i=0}^{W-1} b_i^t = 1$. Similarly, $\mathcal{J}^{\text{per}}(\mathbf{c})$ can be encoded and overloaded with $\mathbf{b}$ as follows:

$$\mathcal{J}^{\text{per}}(\mathbf{c}) = \mathcal{J}^{\text{per}}(\mathbf{b}) = \sum_{t=0}^{H-1} \left(\sum_{i=0}^{W-1} b_i^t \cdot i \cdot \Upsilon \right). \tag{5.32}$$

Therefore, using Eq. (5.31) and Eq. (5.32), we provide an optimal solution to Problem 4 in Solution 1 using a Boolean optimization problem.

Solution 1 (Solution to Problem 4)**.** *The following Boolean optimization problem provides an optimal solution to Problem 4.*

$$\begin{aligned} \min_{\mathbf{b}} \quad & \alpha \cdot \Big(\mathcal{V}(\mathbf{b})^\top \times \Psi \times \mathcal{V}(\mathbf{b}) \Big) + \beta \cdot \mathcal{J}^{per}(\mathbf{b}) \\ s.t. \quad & \forall_t \sum_{i=0}^{W-1} b_i^t = 1 \bigwedge \forall_{t,i} b_i^t \in \mathbb{Z}_{[0,1]}. \end{aligned} \tag{5.33}$$

The Boolean optimization formulation in Solution 1, using $H \cdot W$ Boolean variables, is NP-Hard. [110, 111]

Though the proposed solution is in general NP-Hard, but for most practical cases, using off the shelf tools, one can solve such optimization problems very efficiently using several heuristics. In particular, optimization based solutions have been effectively used in embedded hybrid model predictive control [112], optimal coordination of vehicles at intersections [113] *etc*. The same will also been witnessed in our experiments.

5.2.3 An expectation based approach to the Model Selection Problem

Consider a drone landing scenario, where the drone is expected to land in minimal control cost and perception cost using the perception models available to the drone. The perception models take an image of the helipad as an input and estimates the altitude form that image—this represents the perception input. Though it is impossible to know the exact altitude estimated by a model, at a given time, a probability distribution of the estimated-altitude (*i.e.*, the expected estimated-altitude and the variance by a model) can still be empirically calculated (using past data and dynamics of the drone). In other words, though we cannot know the exact estimated-altitude of the drone by a model, at a certain time, we can still empirically compute the expected estimated-altitude (and the variance) after the landing has been initiated, using past data and the dynamics of the drone. It is worth recalling that provable optimality cannot be ensured in a general scenario, that assumes zero knowledge of future environment states, without imposing any assumption. However, in this paper, we provide a provably optimal solution to the model selection problem that outperforms current

model selection techniques and achieves performance that is very close to the (unrealistic) theoretical bound utilizing only the information from distribution of perception models. The technique will be described in detail in the rest of the section.

In Section 5.2.2 we proposed a solution to Problem 4 assuming that $(\hat{\mathbf{s}}_c - \mathbf{s})$ is known. However, in reality, this assumption might not always hold true, for reasons such as unavailability of perception models $\big(g_{\text{per}}(\cdot)$ in Eq. (5.18)$\big)$ or true perception input $\mathbf{s}$. Whereas, a probabilistic distribution of $(\hat{\mathbf{s}}_c - \mathbf{s})$ can still be known [114, 115, 116]. In other words, instead of various perception models $\big($as in Eq. (5.18)$\big)$, we have various distributions available to us at varying model costs. Formally, we assume the knowledge of the following:

$$\big(g_{\text{per}}(w,t) - s_t\big) \sim \mathcal{T}_t^w, \tag{5.34}$$

$$cost_{\text{per}}(w) = w \cdot \Upsilon, \tag{5.35}$$

where $w \in \mathbb{Z}_{[0,W-1]}$ represents the model choice variable, $\Upsilon \in \mathbb{R}_{>0}$, and $\mathcal{T}_t^w$ is a probability distribution represented with the random variable $\mathcal{D}_t^w$. Intuitively, $\mathcal{D}_t^0$ represents the lowest-quality perception input distribution (*i.e.* high variance and mean farther away from 0) with minimum perception cost $cost_{\text{per}}(c_t) = 0$ (at time step t). $\mathcal{D}_t^{W-1}$ represents the highest-quality perception input distribution (*i.e.* low variance and mean closer to 0) with maximum perception cost $cost_{\text{per}}(W-1) = (W-1) \cdot \Upsilon$ (at time step t). Note that we do not impose any assumption on the distributions. However, in practice, the distributions are generally uniform or normal as the models estimate the target variable fairly well for most of the inputs, while performing relatively poor for some outlier cases. Given a random variable $\mathcal{D}_t^w$, let the expected value be given as $\mathbb{E}[\mathcal{D}_t^w]$, and variance as $Var(\mathcal{D}_t^w)$. For a given choice of model w, we define the following functions:

$$g_{\text{exp}}(w,t) = \mathbb{E}[\mathcal{D}_t^w], \tag{5.36}$$

$$g_{\text{var}}(w,t) = Var(\mathcal{D}_t^w). \tag{5.37}$$

Informally, given a model choice w, Eq. (5.36) returns the expected difference between output of the perception model and the true output $\big($*i.e.* $\mathbb{E}\big[g_{\text{per}}(w,t) - s_t\big]\big)$, and Eq. (5.37) returns the variance $\big($*i.e.* $Var\big(g_{\text{per}}(w,t) - s_t\big)\big)$. Similarly, we extend the operator $\mathbb{E}[\cdot]$ and $Var(\cdot)$ to the vector $(\hat{\mathbf{s}}_c - \mathbf{s})$ in the obvious manner, denoted as $\mathbb{E}[\hat{\mathbf{s}}_c - \mathbf{s}]$ and $Var(\hat{\mathbf{s}}_c - \mathbf{s})$ respectively (Section 5.2.6.3). Naturally, in the setting described

above, Problem 4 recasts itself from minimizing true total loss (see Problem 3) to minimizing expected total loss. We formally state the problem as follows:

Problem 5 (Model Selection Problem)**.** *Given a system with dynamics as in Eq.* (5.17)*, and perception model distribution* $\left(g_{exp}(\cdot)\text{ and }g_{var}(\cdot)\right)$ *with cost* $cost_{per}(\cdot)$*, compute a set of choices* $\mathbf{c}_{opt}$ *for H time steps, such that, for a given* α, β*:*

$$\min_{\mathbf{c}\in\mathbf{C}} \quad \mathbb{E}\left[\alpha\cdot\left((\hat{\mathbf{s}}_{\mathbf{c}}-\mathbf{s})^{\top}\times\Psi\times(\hat{\mathbf{s}}_{\mathbf{c}}-\mathbf{s})\right)+\beta\cdot\mathcal{J}^{per}(\mathbf{c})\right]. \tag{5.38}$$

Given $\mathbb{E}[\hat{\mathbf{s}}_c-\mathbf{s}]=\mathbb{E}[\mathcal{V}(\mathbf{b})]$ and $Var(\hat{\mathbf{s}}_c-\mathbf{s})=Var(\mathcal{V}(\mathbf{b}))$, Problem 5 can be solved by the following optimization problem, proposed in Solution 2

Solution 2 (Solution to Problem 5)**.** *The following Boolean optimization problem provides an optimal solution to Problem 5.*

$$\begin{aligned}\min_{\mathbf{b}} \quad & \alpha\cdot\left(\mathbb{E}[\mathcal{V}(\mathbf{b}])^{\top}\times\Psi\times\mathbb{E}[\mathcal{V}(\mathbf{b})]\right)+\\ & \alpha\cdot\mathbb{V}\cdot Var(\mathcal{V}(\mathbf{b}))+\beta\cdot\mathcal{J}^{per}(\mathbf{b})\\ s.t. \quad & \forall_t\sum_{i=0}^{W-1}b_i^t=1\bigwedge\forall_{t,i}b_i^t\in\mathbb{Z}_{[0,1]},\end{aligned} \tag{5.39}$$

where $\mathbb{V}=\begin{bmatrix}\Psi[0,0] & \Psi[1,1] & \dots & \Psi[K-1,K-1]\end{bmatrix}$*, where* $K=p\cdot H$ *is the dimension of* Ψ*.*

Intuitively, the only difference between Eq. (5.33) (Solution 1) and Eq. (5.39) (Solution 2) is that, instead of minimizing the total cost, it minimizes the *expected* total cost. Since it uses the expectation operator ($\mathbb{E}[\cdot]$) on the random variables, the extra term with variance ($\mathbb{V}$) arises in Eq. (5.39) after algebraic manipulations of random variables with the expectation operator.

Theorem 22. *Solution 2 solves Problem 5, assuming the random variables* $\mathcal{D}_t^w$*, for all t and w, are independent. [The proof, detailed in Section 5.2.6.4, is by algebraic manipulation of Eq.* (5.38)*]*

The Boolean optimization formulation in Solution 2, using $H\cdot W$ Boolean variables, is NP-Hard. [110, 111] We note that the proposed solution is NP-Hard; we do not not claim that the problem is NP-Hard. The complexity class of the model selection problem is currently unknown, and is left as a future work. Note that though we consider a more general version in Problem 5 than Problem 4, we still do not introduce any additional variables to the optimization formulation. The only additional term is added in the objective function, which is again linear.

5.2.4 A Special Subcase: Model Selection in Polynomial Time Complexity

The models that we have been considering so far were discrete, *i.e.*, the choice of the model w in Eq. (5.18) and Eq. (5.19) in Solution 1, are given by integer choices $w \in \mathbb{Z}_{[0,W-1]}$. Similarly the probabilistic models, given in Eq. (5.34) and Eq. (5.35), in Solution 2 consider discrete choices of models, *i.e.*, $w \in \mathbb{Z}_{[0,W-1]}$. In this section, contrary to our models in Eq. (5.36), Eq. (5.37) and Eq. (5.35), we consider models in a continuous setting. That is, we consider a continuous set of models to solve Problem 5 in polynomial time. Formally, we consider the following models.

$$g^c_{\text{exp}}(c,t) = \qquad (1-c)\cdot\mathbb{E}\left[\mathcal{D}_t^0\right] + c\cdot\mathbb{E}\left[\mathcal{D}_t^{W-1}\right], \tag{5.40}$$

$$g^c_{\text{var}}(c,t) = \qquad (1-c)\cdot Var\left(\mathcal{D}_t^0\right) + c\cdot Var\left(\mathcal{D}_t^{W-1}\right), \tag{5.41}$$

where $c \in \mathbb{R}_{[0,1]}$, $\mathcal{D}_t^0$ and $\mathcal{D}_t^{W-1}$ are random variables representing the least and the most accurate distributions, as mentioned before. The cost model is given as: $cost^c_{\text{per}}(c) = c\cdot\Upsilon$. Intuitively, this setting assumes a linear combination of the best and the worst possible models are available. In other words, with $c = 1$, we get the best possible output $\left(i.e.\ \mathbb{E}\left[\mathcal{D}_t^0\right]\right.$ and $Var\left(\mathcal{D}_t^0\right)$ closer to $0\left.\right)$ with cost Υ, and vice-versa with $c = 0$. Despite the fact that there are no continuous set of models available for performing perception, this approach can be useful for approximating early-exit DNN [117, 118] and perform model selection on the resulting DNN. In other words, we now have a DNN with N layers instead of N perception models from which to perform model selection. Running higher number of layers would increase accuracy (at a greater cost) and vice versa. Clearly, such a setting can also be used in model selection. Consider a scenario in which we use a linear function to approximate the output of an early-exit DNN, with the input being a continuous variable (say, $c \in [0,1]$) that serves as a proxy for the number of layers to be run (*i.e.*, the higher the value of c, the more layers are run) and the output being the DNN's output after running the number of layers indicated by the proxy variable. By choosing the input variable appropriately, one may empirically obtain such functions—this is also illustrated in Fig. 5.6b. Though no such provable approximation techniques exist, this can however make the problem of model selection computationally very efficient. The computational complexity, for instance, will be exponential in the number of layers times the time horizon when performing model selection using an early-exit DNN with 1000 layers, however applying the aforementioned approach as a rough approximation will decrease the complexity to polynomial time.

Next, we restate Problem 5 in this setting and propose a polynomial time solution. But before we proceed to restating Problem 5, we are required to put forward the necessary artifacts. Let $\mathbf{C}^c$ be the set of all possible choices. Let $\mathbf{c} = \left\{c_0, c_1, \cdots, c_{H-1}\right\} \in \mathbf{C}^c$ be a set of choices up-to time step $H-1$, where $c_i \in \mathbb{R}_{[0,1]}$. Using this, we rewrite $\mathbb{E}\left[\hat{\mathbf{s}}_c - \mathbf{s}\right]$ by replacing the $H \cdot W$ binary variables with H real variables as follows:

$$\begin{aligned}\mathbb{E}\left[\hat{\mathbf{s}}_c - \mathbf{s}\right] = \mathbb{E}\left[\mathcal{V}(\mathbf{c})\right] = \Bigg[& (1-c_0)\cdot\mathbb{E}\left[\mathcal{D}_0^0\right] + c_0 \cdot \mathbb{E}\left[\mathcal{D}_0^{W-1}\right] \\ & \ldots\ (1-c_t)\cdot\mathbb{E}\left[\mathcal{D}_t^0\right] + c_t \cdot \mathbb{E}\left[\mathcal{D}_0^{W-1}\right] \\ & \ldots\ (1-c_{H-1})\cdot\mathbb{E}\left[\mathcal{D}_{H-1}^0\right] + c_{H-1} \cdot \mathbb{E}\left[\mathcal{D}_{H-1}^{W-1}\right] \Bigg]^\top \end{aligned} \tag{5.42}$$

Now we restate Problem 5 in a continuous setting as follows:

Problem 6 (Model Selection with Continuous Choices)**.** *Given a system with dynamics as in Eq.* (5.17)*, and perception model distribution* $\left(g_{exp}^c(\cdot)\right.$ *and* $\left.g_{var}^c(\cdot)\right)$ *with cost* $cost_{per}^c(\cdot)$*, compute a set of choices* $\mathbf{c}_{opt}$ *for H time steps, such that, for a given* α, β*:*

$$\min_{\mathbf{c}\in\mathbf{C}} \quad \mathbb{E}\left[\alpha \cdot \left((\hat{\mathbf{s}}_\mathbf{c} - \mathbf{s})^\top \times \Psi \times (\hat{\mathbf{s}}_\mathbf{c} - \mathbf{s})\right) + \beta \cdot \mathcal{J}^{per}(\mathbf{c})\right] \tag{5.43}$$

Given $\mathbb{E}\left[\hat{\mathbf{s}}_c - \mathbf{s}\right] = \mathbb{E}\left[\mathcal{V}(\mathbf{c})\right]$ and $Var\left(\hat{\mathbf{s}}_c - \mathbf{s}\right) = Var\left(\mathcal{V}(\mathbf{c})\right)$, Problem 6 can be solved by the following optimization problem, proposed in Solution 3. Intuitively, Problem 6 is very similar to Problem 5, except that Problem 6 uses continuous set of perception models instead of a discrete set of perception models as in Problem 5.

Solution 3 (Solution to Problem 5)**.** *The following Boolean optimization problem provides an optimal solution to Problem 6.*

$$\begin{aligned} \min_{\mathbf{c}\in\mathbf{C}^c} \quad & \alpha \cdot \left(\mathbb{E}\left[\mathcal{V}(\mathbf{c})\right]^\top \times \Psi \times \mathbb{E}\left[\mathcal{V}(\mathbf{c})\right]\right) + \alpha \cdot \mathbb{V} \cdot Var\left(\mathcal{V}(\mathbf{c})\right) \\ & + \beta \cdot \mathcal{J}^{per}(\mathbf{c}) \\ s.t. \quad & \forall_t c_t \in \mathbb{R}_{[0,1]}, \end{aligned} \tag{5.44}$$

where $\mathbb{V} = \Big[\Psi[0,0] \quad \Psi[1,1] \quad \ldots \quad \Psi[K-1,K-1],\Big]$ *$K = p \cdot H$ is the dimension of Ψ.*

Solution 3 is again very similar to Solution 2; it replaces the $W \times H$ binary variables (where W is the number of models and H is the time horizon) with H continuous variables, where each continuous variable (between 0 to 1) represents the desired perception model at a given time step.

Theorem 23. *The quadratic optimization formulation in Solution 3, using H real variables, can be solved in polynomial time. [The proof hinges on casting the quadratic optimization problem in Solution 3 to a semidefinite program [119]. The details of the proof is given in Section 5.2.6.5]*

5.2.5 Experiments

We evaluate this approach on a photo-realistic drone landing model using visual navigation, where the state variables are altitude, rate of change of altitude, pitch angle, and the rate of change of pitch angle, with the input being the elevator angle. The control task is to land the aircraft using LQR control as in [108]. We consider the model for our drone landing as in [120]. Next, we discretize the given model, in [120], and rewrite it in form of Eq. (5.17) where the Cs_t part denotes the estimation error in altitude. Note that the state of drone obtained from the mathematical model is almost never accurate, which could be due to various uncertainties, such as wind, the motion of the landing pad (say in a water-body). Note that the difference in state, obtained from the mathematical model and reality, is encoded by the Cs_t part of the dynamics as in Eq. (5.17). To be able to estimate the true altitude (perception input) of the drone, one can use various perception models. In the following set of experiments, we will evaluate our proposed solution to the model selection problem. Recall that information of future perception inputs (*i.e.*, the distributions) is necessary since our technique guarantees provable optimality. This information in our experiments is the discrepancy between the true altitude and the estimated altitude (as determined by the perception models) at a given time step. The dynamics of the drone and historical data is used to experimentally determine the mean difference and the variance at a particular time step of the perception models, even though it is impossible to determine the precise difference (which is also not required in our setup). We follow this idea in our subsequent experiments. A choice of perception model to be invoked, at a given step, obtained using our proposed approach strikes an optimal balance between the control cost and the perception cost, whereas other benchmark policies fail. To this end, we will perform the following two experiments to evaluate or proposed optimal solution to the model selection problem:

1. Simulate the availability of a plethora of perception models that can estimate the true altitude of the drone, with varying compute cost and accuracy, and evaluate our optimal model selection algorithm (Problem 5).

2. Similar to the above case, instead of simulating the perception models, we use AirSim [109] to gather dataset of drone landing to train EfficientNet models b0 to b7 [106] to accurately estimate the altitude of the drone. Once we train the models, similar to above experimental setup, we use our proposed optimal model selection solution to land the drone by balancing control cost and perception cost.

We recall that the main purpose of this experiment is to evaluate the proposed solution to check if it indeed strikes the optimal balance between control cost and perception cost, against the following benchmarks:

1. `All Small`: The model with the least accuracy and compute cost is used at all time steps.

2. `All Large`: The model with the highest accuracy and compute cost is used at all time steps.

3. `Random`: At a given time step, a random model is chosen to perform the task.

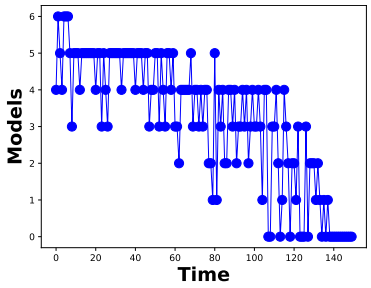
4. `Optimal`: Our proposed model selection policy.

5. `Oracle`: A strategy that assumes the results of all the models are known a priori, even without invoking the model. Note that this is an infeasible policy, we use this policy just to benchmark our proposed policy—this also serves as a theoretical best case bound.

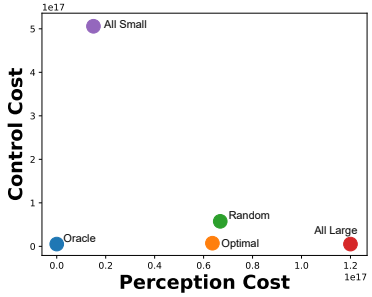
Using the proposed model selection policy, we want to reduce the total control cost for landing the drone with minimal perception cost.

5.2.5.1 Drone Landing Simulation

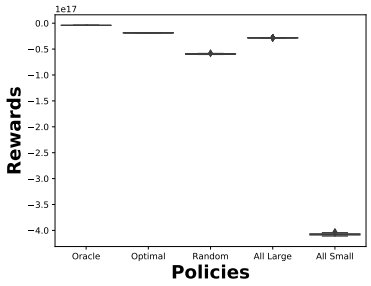
In this subsection, we solve Problem 5 on an drone landing simulation. We perform a numerical simulation of a drone landing from a height of 1000 meters. In addition, we numerically simulate eight perception models that estimate the altitude of the drone. The expected accuracy of the models are numerically simulated to follow a sigmoidal pattern as in Fig. 5.13a. In particular, we simulate distributions of perception models as in Eq. (5.34). Note that, we train real perception models and use its distribution instead of numerical simulation in the subsequent experiment.



(a) Model Selection Sequence. The x and the y axis represents time steps and the perception models invoked respectively. That is, this plot shows the perception models to be invoked at a every time step, up-to 150 time steps.



(b) Perception Model Cost vs. Control Cost. The control cost and perception model cost for all the policies are shown. Clearly, our proposed policy (`Optimal`) achieves an optimal balance between control cost and perception model cost compared to other policies.



(c) Rewards. Total reward obtained by various policies. Clearly, our proposed strategy (`Optimal`) obtains highest reward, and very close to the infeasible `Oracle` policy.

Figure 5.7: Results evaluating our proposed model selection strategy (`Optimal`) on a numerical aircraft landing simulation.

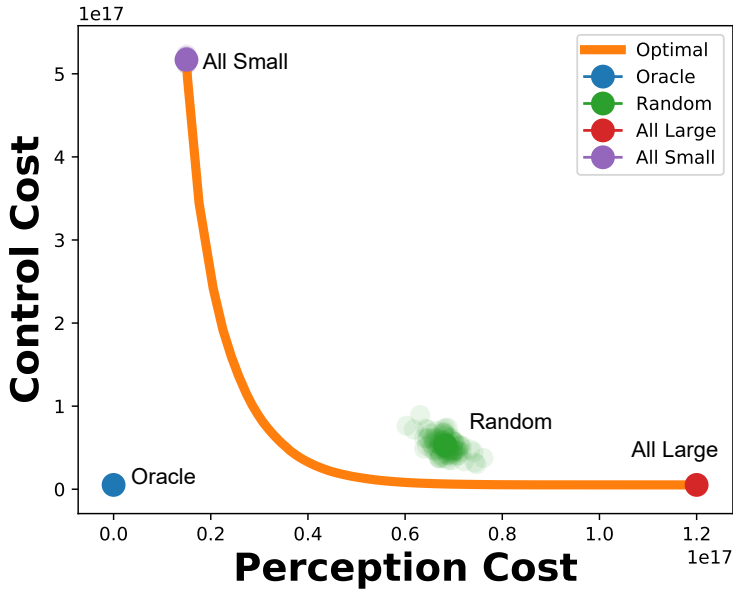


Figure 5.8: Pareto (Numerical Simulation). This plot shows how the behavior of our optimal policy changes with varying values of α and β. Recall, higher values of α implies the model selection sequence should prioritize control cost over perception and model cost. Whereas higher values of β implies the model selection sequence should prioritize perception model cost over control cost.

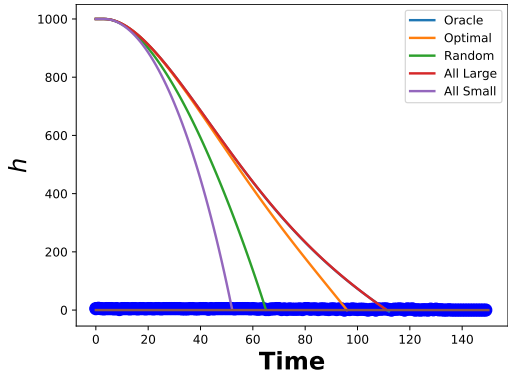


Figure 5.9: Landing Trajectories (Numerical Simulation). Plot showing the change altitude happening, for landing the aircraft, with time for various policies.

Once the required information is gathered, we solve Problem 5, and obtain the model selection sequence given in Fig. 5.7a, for 150 time steps, taking 2.88 s. Note that, in this instance, model b7 was never invoked. Since this is a safety-critical scenario of drone landing, we adjusted the parameters in a way that the synthesized model selection sequence prioritizes the control cost above the perception cost. In order to avoid compromising on control cost, the method assigns a large budget of perception cost (*i.e.*, computationally costly and accurate models may be called more frequently). In particular, we set $\alpha = 0.8$ and $\beta = 0.2$, which suggests that the control cost should receive 80% of the overall effort and the perception cost should receive the remaining 20%. Additionally, we selected $\Upsilon = 1e14$, which stands for the factor that determines how much the perception cost increases as the model index grows.

Now recall that this model selection sequence maximizes the reward in expectation. Therefore, to evaluate the obtained model selection sequence, we instantiated the distribution, for all the perception models, for 100 trials. In other words, we assign concrete values to the measurement errors according to the assumed distribution. Once concrete values to the measurement errors are assigned, for all perception models, we evaluate our model selection sequence for the following.

- Total perception cost and total control cost for all the polices are given in Fig. 5.7b. We clearly observe that our model selection strategy (`Optimal`) outperforms all other policies, achieving low control and perception cost. In particular, using our policy one can achieve 567.18% lower control cost in 80.72% less perception cost than the `Random` policy.

- Total reward gathered by all the polices is given in Fig. 5.7c. Our policy achieves 53.26% better reward than the second best policy (`All Large`), and 77% far from the infeasible `Oracle` policy—which also indicates a theoretical best case bound.

- Fig. 5.8 shows the perception cost vs. control cost obtained by various policies with varying values of α and β. Recall, a higher value of α implies the model selection sequence should prioritize control cost over perception cost. Whereas higher values of β implies the model selection sequence should prioritize perception model cost over control cost. In other words, the costs indicated by the orange curve (`Optimal`) on the bottom-right part of Fig. 5.8 indicates a very high priority is assigned to the control cost (tends to invoke expensive models frequently, ensuring low control cost), while the top-left part of the plot indicates costs with higher priority on the perception cost.

- Finally, the trajectories of landing (change in altitude with time) obtained using various policies are given in Fig. 5.9.

5.2.5.2 Photo-realistic Drone Landing Simulation in AirSim

Similar to the previous set of experiments, except for numerically-simulating the distribution of the perception models, we simulate a photo-realistic drone landing sequence in AirSim using visual navigation. Contrary to our previous set of experiments (Section 5.2.5.2), we simulate the drone landing in AirSim and gather a dataset to train EfficientNet models b0 to b7. Our dataset contains the images captured from the camera attached to the drone and the altitude of the drone with respect to the helipad. Further, we compute the distribution, as in Eq. (5.34), of the trained EfficientNet models using a different dataset (than the one used for training). Once the distribution is computed, we perform same steps as our previous experiments to solve Problem 5 and the evaluate the solution. Next, we perform the following steps to compute the distribution of the perception models using AirSim.

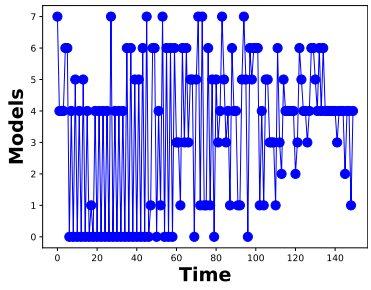
1. We setup an aircraft landing environment using a drone simulation. The drone is expected to land on a helipad. Moreover, the environment is cluttered, there are moving objects near the helipad that might make the perception difficult. The setup environment is shown in Fig. 5.12b (Appendix).

2. Using this setup, we gathered 2093 images as our training dataset, and 897 images as our testing dataset, by equipping the drone with a camera. See Fig. 5.12a for an example image.

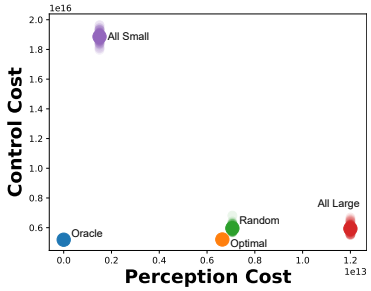
3. We then trained EfficientNet b0 to b7 on the training dataset to predict the difference in height of the drone to the helipad. See Fig. 5.13b for training accuracy plot of EfficientNet b7. The average accuracy of all the trained models, along with their 95% confidence interval, is given in Fig. 5.6c.

4. Once all the models are trained, we use the testing dataset to compute the distribution, as in Eq. (5.34), for all the models. An example distribution of b7 can be found in Fig. 5.13c.

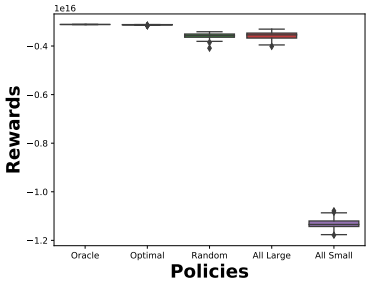
Once the distributions are obtained, we perform similar steps as before. We now solve Problem 5, and obtain the model selection sequence given in Fig. 5.10a, for 150 time steps, taking 4.16 s. Similar to our previous experiment, the parameters were adjusted to assign more priority to the control cost than perception cost. In particular, we set $\alpha = 0.6$, $\beta = 0.4$ (suggests that the control cost receives 60% of the overall effort



(a) Model Selection Sequence. The x and the y axis represents time steps and the perception models invoked respectively. That is, this plot shows the perception models to be invoked at a every time step, up-to 150 time steps.



(b) Perception Model Cost vs. Control Cost. The control cost and perception model cost for all the policies are shown. Clearly, our proposed policy (`Optimal`) achieves an optimal balance between control cost and perception model cost compared to other policies.



(c) Rewards. Total reward obtained by various policies. Clearly, our proposed strategy (`Optimal`) obtains highest reward, and very close to the infeasible `Oracle` policy.

Figure 5.10: Results evaluating our proposed model selection strategy (`Optimal`) on an aircraft landing simulation in AirSim.

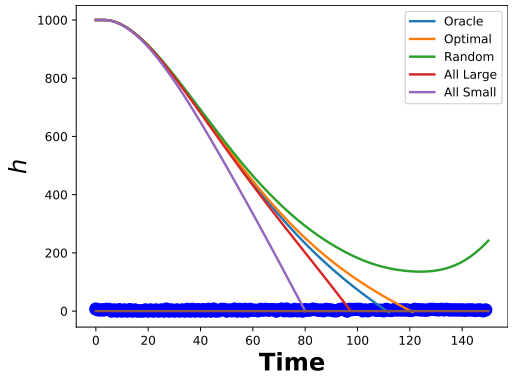


Figure 5.11: Landing Trajectories (AirSim Simulation). Plot showing the change altitude happening, for landing the aircraft, with time for various policies.

and the remaining by the perception cost), and $\Upsilon = 1e10$. Similar to our previous experiments we now evaluate our model selection sequence over 100 trials, and observe the following:

- Total perception cost and total control cost for all the polices are given in Fig. 5.10b. We clearly observe that our model selection strategy (`Optimal`) outperforms all other policies, achieving low control and perception cost. In particular, using our policy one can achieve 38.04% lower control cost in 79.1% less perception model cost than the `Random` policy.

- Total reward gathered by all the polices is given in Fig. 5.10c. Our policy achieves 13.97% better reward than the second best policy (`All Large`), and just 0.3% far from the infeasible `Oracle` policy—it demonstrates that the performance of our suggested optimal strategy is extremely close to the theoretical best case bound.

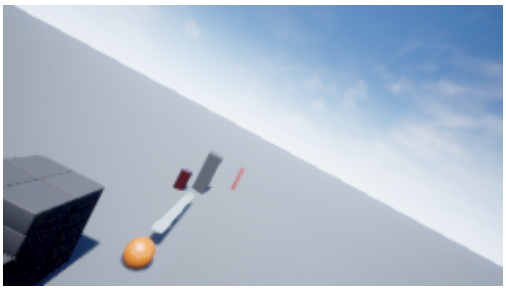
- Finally, the trajectory of landing, change in altitude with time, obtained using various policies is given in Fig. 5.11.

Limitations: In our formulation, the dynamics of the system is considered to be linear with additive uncertainties. In our future work, we wish to extend our techniques to non-linear dynamics with more intricate uncertainties. We note that such an extension is not straightforward (one needs to extended the closed loop control cost formulation and encode the optimization problem in a manner that is easy for the solvers to solve efficiently). Further, we have currently focused on LQR control using some analytical results on LQR from a previous work. We wish to extend our technique to more control cost formulation, taking motivation from several robotic control applications. Advancing the multiple step decision problem to swarm robotics is also on our agenda (A scenario, where instead of a single robot, a network of robots each having access to the same models). In a such a setup, how to allocate perception models to the robots such that optimality is ensured. Finally, we also intend to deploy the synthesized model selection policy on a real robots in a factory floor.

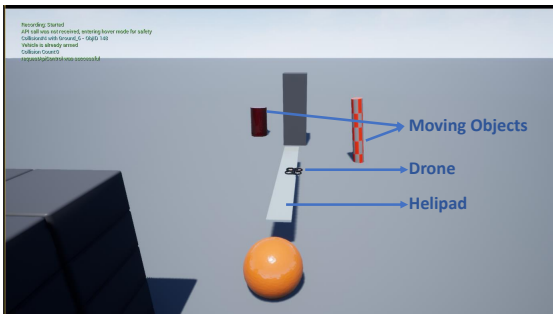
5.2.6 Detailed Analysis

5.2.6.1 Proof of Equivalence

In this section we provide a proof of Theorem 21



(a) Dataset. The dataset is gathered by equipping the drone with a camera in the given environment.



(b) AirSim Environment. The environment emulates a drone landing on a helipad, where the environment also has a lot of other fixed and moving objects.

Figure 5.12: AirSim environment and data gathering.

Proof. Problem 3 is stated as the following minimization problem:

$$\min_{\mathbf{c}\in\mathbf{C}} \quad \mathcal{J}^{\text{tot}}(\mathbf{c}). \tag{5.45}$$

The above equation can be equivalently restated as:

$$\min_{\mathbf{c}\in\mathbf{C}} \quad \alpha\cdot\mathcal{J}^{\text{ctrl}}(\mathbf{c})+\beta\cdot\mathcal{J}^{\text{per}}(\mathbf{c}). \tag{5.46}$$

That is, $\mathbf{c}_{opt}\in\mathbf{C}$ minimizing the above optimization problem is a solution to Problem 4.

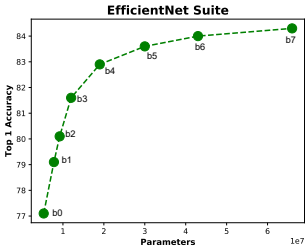
Next, we show that minimization problem in Eq. (5.27) is equivalent to the minimization problem in Eq. (5.46).

Consider the following equivalent formulations:

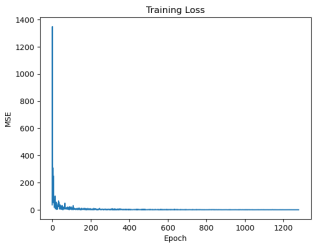
$$\min_{\mathbf{c}\in\mathbf{C}} \quad \alpha\cdot\Big((\hat{\mathbf{s}}_{\mathbf{c}}-\mathbf{s})^{\top}\times\Psi\times(\hat{\mathbf{s}}_{\mathbf{c}}-\mathbf{s})\Big)+\beta\cdot\mathcal{J}^{\text{per}}(\mathbf{c}) \tag{5.47}$$

$$\iff \min_{\mathbf{c}\in\mathbf{C}} \quad \alpha\cdot\Big(J^{c}\big(\mu(\hat{\mathbf{s}}),\mathbf{s},x_0\big)-J^{c}\big(\mathbf{u}^{*},\mathbf{s},x_0\big)\Big)+\beta\cdot\mathcal{J}^{\text{per}}(\mathbf{c}) \quad [\because Eq.\ (5.26)] \tag{5.48}$$

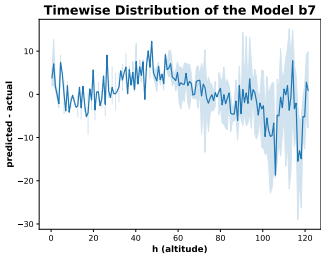
$$\iff \min_{\mathbf{c}\in\mathbf{C}} \quad \alpha\cdot\Big(\mathcal{J}^{\text{ctrl}}(\mathbf{c})-J^{c}\big(\mathbf{u}^{*},\mathbf{s},x_0\big)\Big)+\beta\cdot\mathcal{J}^{\text{per}}(\mathbf{c}) \tag{5.49}$$



(a) EfficientNet Suite. (The data is taken from [106]). What model should we dynamically invoke based on a robot's real-time operating context?



(b) EfficientNet b7 Training Loss. The x axis represents the epoch, and the y axis represents the MSE loss at that epoch.



(c) EfficientNet b7 Distribution. The x axis represents the altitude of the drone, and the y axis represents the altitude, and the shaded area (in blue) represents the corresponding distribution at that altitude.

(d) Data Gathering using AirSim

Figure 5.13: Left. EfficientNet suite accuracy trend. **Right.** Data Gathering using AirSim

$$\min_{\mathbf{c}\in\mathbf{C}} \quad \underbrace{\alpha\cdot\mathcal{J}^{\text{ctrl}}(\mathbf{c})+\beta\cdot\mathcal{J}^{\text{per}}(\mathbf{c})}_{Eq.\ (5.46)} - \underbrace{\alpha\cdot J^{c}\left(\mathbf{u}^{*},\mathbf{s},x_0\right)}_{\text{Oracle control cost}}. \tag{5.50}$$

$\Longleftrightarrow$

The term $\alpha\cdot J^{c}\left(\mathbf{u}^{*},\mathbf{s},x_0\right)$ encoding the oracle control cost is a constant term in the objective function of the above optimization problem, as it is independent of the optimization variable $\mathbf{c}$. Note that the objective function of the optimization formulation, in Eq. (5.50), is obtained by subtracting the constant term $\alpha\cdot J^{c}\left(\mathbf{u}^{*},\mathbf{s},x_0\right)$ from the objective function of the optimization problem in Eq. (5.46). Therefore, $\mathbf{c}_{opt}\in\mathbf{C}$ minimizing the objective function in Eq. (5.46) also minimizes the objective function in Eq. (5.50). □

5.2.6.2 Encoding with Binary Variables

In this section we show that $\mathbf{c}$ and $\mathbf{b}$ are equivalent.

Given a set of choices $\mathbf{c}=\left\{w_0,w_1,\cdots,w_{H-1}\right\}$, one can represent $\mathbf{c}$ using $\mathbf{b}=\{b_0^0,b_1^0,\ldots,b_{W-1}^0,b_0^1,\ldots b_{W-1}^1,\ldots b_{W-1}^{H-1}\}$, such that, $\forall_{0\le t\le H-1}b_{w_t}^t=1$, and $\forall_{0\le t\le H-1}\sum_{i=0}^{W-1}b_i^t=1$.

5.2.6.3 Expectation and Variance of the Difference Vector

In this section we provide the expectation and variance of $(\hat{\mathbf{s}}_c-\mathbf{s})$

$$\mathbb{E}\left[\hat{\mathbf{s}}_c - \mathbf{s}\right] = \mathbb{E}\left[\mathcal{V}(\mathbf{b})\right] = \begin{bmatrix} \sum_{i=0}^{W-1} b_i^0 \cdot \mathbb{E}\left[g_{\text{per}}(i,0) - s_0\right] \\ \vdots \\ \sum_{i=0}^{W-1} b_i^t \cdot \mathbb{E}\left[g_{\text{per}}(i,t) - s_t\right] \\ \vdots \\ \sum_{i=0}^{W-1} b_i^{H-1} \cdot \mathbb{E}\left[g_{\text{per}}(i,H-1) - s_{H-1}\right] \end{bmatrix}. \tag{5.51}$$

$$Var\left(\hat{\mathbf{s}}_c - \mathbf{s}\right) = Var\left(\mathcal{V}(\mathbf{b})\right) = \begin{bmatrix} \sum_{i=0}^{W-1} b_i^0 \cdot Var\left(g_{\text{per}}(i,0) - s_0\right) \\ \vdots \\ \sum_{i=0}^{W-1} b_i^t \cdot Var\left(g_{\text{per}}(i,t) - s_t\right) \\ \vdots \\ \sum_{i=0}^{W-1} b_i^{H-1} \cdot Var\left(g_{\text{per}}(i,H-1) - s_{H-1}\right) \end{bmatrix}. \tag{5.52}$$

5.2.6.4 Minimizing Expectation

If the random variables are $\mathcal{D}_t^w$ (for all t and w) are independent, the objective function given in Eq. (5.38) can be rewritten as (see Section 5.2.6.4):

$$\begin{aligned} \mathbb{E}\left[\alpha \cdot \left((\hat{\mathbf{s}}_\mathbf{c} - \mathbf{s})^\top \times \boldsymbol{\Psi} \times (\hat{\mathbf{s}}_\mathbf{c} - \mathbf{s})\right) + \beta \cdot \mathcal{J}^{\text{per}}(\mathbf{c})\right] = \\ \alpha \cdot \left(\mathbb{E}\left[(\hat{\mathbf{s}}_\mathbf{c} - \mathbf{s})\right]^\top \times \boldsymbol{\Psi} \times \mathbb{E}\left[(\hat{\mathbf{s}}_\mathbf{c} - \mathbf{s})\right]\right) + \\ \alpha \cdot \mathbb{V} \cdot Var\left((\hat{\mathbf{s}}_\mathbf{c} - \mathbf{s})\right) + \beta \cdot \mathcal{J}^{\text{per}}(\mathbf{c}). \end{aligned}$$

Since $\hat{\mathbf{s}}_c - \mathbf{s} = \mathcal{V}(\mathbf{b})$, with constraints $\forall_t \sum_{i=0}^{W-1} b_i^t = 1$ and $\forall_{t,i} b_i^t \in \mathbb{Z}_{[0,1]}$, the above equation evaluates to the following:

$$\begin{aligned} \left(\mathbb{E}\left[\mathcal{V}(\mathbf{b})\right]^\top \times \boldsymbol{\Psi} \times \mathbb{E}\left[\mathcal{V}(\mathbf{b})\right]\right) + \\ \alpha \cdot \mathbb{V} \cdot Var\left(\mathcal{V}(\mathbf{b})\right) + \beta \cdot \mathcal{J}^{\text{per}}(\mathbf{b}). \end{aligned}$$

Finally the optimization in Eq. (5.39) is formulated by replacing the objective function with the above equivalent function, and the added constraints.

Reformulation

$$\hat{\mathbf{s}}_c - \mathbf{s} = \begin{bmatrix} \hat{s}_0 - s_0 \\ \hat{s}_1 - s_1 \\ \vdots \\ \hat{s}_{H-1} - s_{H-1} \end{bmatrix}. \tag{5.53}$$

Note that each element in the vector given in Eq. (5.53) is a vector of p dimension. Therefore, the vector $(\hat{\mathbf{s}}_c - \mathbf{s})$ can be rewritten as follows:

$$\hat{\mathbf{s}}_c - \mathbf{s} = \begin{bmatrix} \hat{s}_0[0] - s_0[0] \\ \vdots \\ \hat{s}_0[p-1] - s_0[p-1] \\ \hat{s}_1[0] - s_1[0] \\ \vdots \\ \hat{s}_{H-1}[p-1] - s_{H-1}[p-1] \end{bmatrix}, \tag{5.54}$$

where the r-th ($0 \leq r \leq p-1$) element of the vector $\hat{s}_t$ is given as $\hat{s}_t[r]$. As mentioned in Section 5.2.3, each element in the vector Eq. (5.54) comes from a probability distribution; let the random variable corresponding to $\hat{s}_t[r] - s_t[r]$ be denoted as d_{pt+r}, for given choice of model w. Therefore Eq. (5.54) can be further rewritten as

$$\hat{\mathbf{s}}_c - \mathbf{s} = \begin{bmatrix} \hat{s}_0[0] - s_0[0] \\ \vdots \\ \hat{s}_0[p-1] - s_0[p-1] \\ \hat{s}_1[0] - s_1[0] \\ \vdots \\ \hat{s}_{H-1}[p-1] - s_{H-1}[p-1] \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{p \cdot H-1} \end{bmatrix} = \mathbf{d}. \tag{5.55}$$

Assumption 2. *Given any two random variable d_i, d_j, for $0 \leq i, j \leq p \cdot H - 1$, they are independent.*

$$\mathbb{E}\left[\alpha \cdot \left((\hat{\mathbf{s}}_\mathbf{c} - \mathbf{s})^\top \times \Psi \times (\hat{\mathbf{s}}_\mathbf{c} - \mathbf{s})\right) + \beta \cdot \mathcal{J}^{\text{per}}(\mathbf{c})\right] \tag{5.56}$$

$$= \mathbb{E}\left[\alpha \cdot \left(\mathbf{d}^\top \times \Psi \times \mathbf{d}\right)\right] + \beta \cdot \mathcal{J}^{\text{per}}(\mathbf{c}) \tag{5.57}$$

$$= \alpha \cdot \sum_{j=0}^{p \cdot H-1} \mathbb{E}\left[\sum_{i=0}^{p \cdot H-1} d_j \cdot d_i \cdot \Psi[i][j]\right] + \beta \cdot \mathcal{J}^{\text{per}}(\mathbf{c}) \tag{5.58}$$

For any j in the first summation, using Assumption 2, we have:

$$\mathbb{E}\left[\sum_{i=0}^{p\cdot H-1} d_j \cdot d_i \cdot \Psi[i][j]\right] =$$

$$\sum_{i=0}^{p\cdot H-1} \mathbb{E}[d_j]\,\mathbb{E}[d_i] + Var(d_j)\cdot\Psi[j][j]. \tag{5.59}$$

Using Eq. (5.59), we rewrite Eq. (5.58) as follows:

$$\alpha \cdot \sum_{j=0}^{p\cdot H-1} \mathbb{E}\left[\sum_{i=0}^{p\cdot H-1} d_j \cdot d_i \cdot \Psi[i][j]\right] + \beta \cdot \mathcal{J}^{\text{per}}(\mathbf{c}) \tag{5.60}$$

$$= \sum_j \sum_i \Psi[i][j] \cdot \mathbb{E}[d_j] \cdot \mathbb{E}[d_i] +$$

$$\sum_i \Psi[i][j] \cdot Var(d_j) +$$

$$\beta \cdot \mathcal{J}^{\text{per}}(\mathbf{c}) \tag{5.61}$$

$$= \alpha \cdot \Big(\mathbb{E}[(\hat{\mathbf{s}}_{\mathbf{c}} - \mathbf{s})]^{\top} \times \Psi \times \mathbb{E}[(\hat{\mathbf{s}}_{\mathbf{c}} - \mathbf{s})]\Big) +$$

$$\alpha \cdot Var((\hat{\mathbf{s}}_{\mathbf{c}} - \mathbf{s})) +$$

$$\beta \cdot \mathcal{J}^{\text{per}}(\mathbf{c}). \tag{5.62}$$

5.2.6.5 Polynomial Time Proof

Note that each element in the vector $\mathbb{E}[\mathcal{V}(\mathbf{c})]$ is a p dimensional vector. Therefore, the vector $\mathbb{E}[\mathcal{V}(\mathbf{c})]$ can be unwrapped and rewritten with equivalent choice variables

$$\mathbf{c}' = \begin{bmatrix} c'_0 & c'_1 & \dots & c'_{p\cdot H-1}, \end{bmatrix}^{\top} \tag{5.63}$$

where $\forall_{t=0}^{H-1}\ c_t = c'_{p\cdot t} = \cdots = c'_{p\cdot t-1}$. Let $\mathbb{E}[\mathcal{V}(\mathbf{c})] = \mathbb{E}[\mathcal{V}(\mathbf{c}')]$ (see Section 5.2.6.6 for details). Further, we rewrite $\mathcal{J}^{\text{per}}(\mathbf{c})$ $\big($Eq. (5.21)$\big)$ in terms of $\mathbf{c}'$ as $\mathfrak{J}\mathbf{c}'$; where $\mathfrak{J} \in \mathbb{R}^{1\times p\cdot H}$ can be computed easily *s.t.* $\mathcal{J}^{\text{per}}(\mathbf{c}) = \mathfrak{J}\mathbf{c}'$.

Using the above idea, we rewrite the optimization problem in Eq. (5.44) as follows:

$$\begin{aligned}\min_{\mathbf{c}\in\mathbf{C}^c} \quad & \alpha\cdot\Big(\mathbb{E}\left[\mathcal{V}(\mathbf{c}')\right]^\top\times\Psi\times\mathbb{E}\left[\mathcal{V}(\mathbf{c}')\right]\Big)\\ & +\alpha\cdot\mathbb{V}\cdot Var\left(\mathcal{V}(\mathbf{c}')\right)+\beta\cdot\mathcal{J}^{\text{per}}(\mathbf{c}')\\ \text{s.t.} \quad & \forall_{0\le t\le H-1}\forall_{1\le r\le p-1} \quad W_{t,r}^\top\mathbf{c}'=0\\ & \forall_{0\le t\le H-1} \qquad 0\le E_t^\top\mathbf{c}'\le 1,\end{aligned} \tag{5.64}$$

where: $W_{t,r}$ is column matrix of size pH with all zeros, except $W_{t,r}[pt+(r-1)]=1$ and $W_{t,r}[pt+r]=-1$; E_t is column matrix of size pH with all zeros, except $E_t[t]=1$. Intuitively, the first constraint encodes $\forall_{t=0}^{H-1}$ $c'_{p\cdot t}=\cdots=c'_{p\cdot t-1}$, and the second constraint ensures that choice variables are in $\mathbb{R}_{[0,1]}$.

Note that for a given t, $\mathbb{E}\left[\mathcal{D}_t^0\right]$ and $\mathbb{E}\left[\mathcal{D}_t^{W-1}\right]$ are also a p dimensional vector. Let the r-th element of these vectors be denoted as $\mathbb{E}\left[\mathcal{D}_t^0\right][r]$ and $\mathbb{E}\left[\mathcal{D}_t^{W-1}\right][r]$. Let $\mathbb{E}\left[\mathcal{D}_t^0\right][r]=\Gamma^0_{pt+r}$ and $\mathbb{E}\left[\mathcal{D}_t^{W-1}\right][r]=\Gamma^1_{pt+r}$. Now we can rewrite $\mathbb{E}\left[\mathcal{V}(\mathbf{c})\right]$ with $\mathbf{c}'$ as follows:

$$\mathbb{E}\left[\hat{\mathbf{s}}_c-\mathbf{s}\right]=\mathbb{E}\left[\mathcal{V}(\mathbf{c}')\right]=\begin{bmatrix}(1-c'_0)\cdot\Gamma^0_0+c'_0\cdot\Gamma^1_0\\ \vdots\\ (1-c'_{pt+r})\cdot\Gamma^0_{pt+r}+c'_t\cdot\Gamma^1_{pt+r}\\ \vdots\\ (1-c'_{pH-1})\cdot\Gamma^0_{pH-1}+c'_{pH-1}\cdot\Gamma^1_{pH-1}\end{bmatrix}.$$

Note that the term, $\alpha\cdot\Big(\mathbb{E}\left[\mathcal{V}(\mathbf{c}')\right])^\top\times\Psi\times\mathbb{E}\left[\mathcal{V}(\mathbf{c}')\right]\Big)$, in Eq. (5.64), has quadratic, linear and constant terms in $\mathbf{c}'$. Following simple algebraic steps to separate the quadratic, linear and constant terms, we can restate the optimization formulation in Eq. (5.64), in its canonical form, as follows (the steps are provided in Section 5.2.6.7):

$$\begin{aligned}\min_{\mathbf{c}'} \quad & \mathbf{c}'^\top\Psi'\mathbf{c}'+(\mathcal{L}+\mathcal{R}+\beta\cdot\mathfrak{I})\mathbf{c}'+\mathcal{K}+\mathcal{K}_2\\ \text{s.t.} \quad & \forall_{0\le t\le H-1}\forall_{0\le r\le p-1} \quad W_{t,r}^\top\mathbf{c}'=0\\ & \forall_{0\le t\le H-1} \quad 0\le E_t^\top\mathbf{c}'\le 1,\end{aligned} \tag{5.65}$$

where Ψ' is a matrix with elements: $\Psi'[i][j]=\alpha\cdot\left(\Gamma^0_i\Gamma^0_j+\Gamma^0_i\Gamma^1_j+\Gamma^1_i\Gamma^0_j+\Gamma^1_i\Gamma^1_j\right)\cdot\Psi[i][j]$. $\mathcal{L}$ is the linear term (in variables $\mathbf{c}'$) after evaluating the term $\alpha\cdot\Big(\mathbb{E}\left[\mathcal{V}(\mathbf{c}')\right]^\top\times\Psi\times\mathbb{E}\left[\mathcal{V}(\mathbf{c}')\right]\Big)$, and $\mathcal{K}$ is the constant term.

Let $\alpha \cdot \mathbb{V} \cdot Var(\mathcal{V}(\mathbf{c}')) = \mathcal{R} \cdot \mathbf{c}' + \mathcal{K}_2$. Note that, we only evaluated the quadratic terms Ψ' (and not the linear and constant terms) because the complexity of a quadratic programming is dependent on the quadratic term [119, 110, 111].

Theorem 24. *Ψ' is a positive semidefine matrix.*

Proof. Let, $\Psi' = \Phi \circ \Psi$, where $\Phi[i][j] = \Gamma^0_i\Gamma^0_j + \Gamma^0_i\Gamma^1_j + \Gamma^1_i\Gamma^0_j + \Gamma^1_i\Gamma^1_j$, and $A \circ B$ denotes the Hadamard product of the two matrices A and B.

We note that Ψ is a positive semi-definite matrix [108]. Further, Ψ' is obtained by multiplying each element of the matrix Ψ with a factor constituting of terms from from the perception model. Imposing some practical assumptions of Φ (formally stated in Section 5.2.6.8), we observe that Ψ' does not differ sufficiently from Ψ' to lose its semi-definite property. That is, when the factors, coming from the perception model, are multiplied with the elements of Ψ it does not lose its semi-definite property.

In other words, imposing some practical assumptions on Φ (formally stated in Section 5.2.6.8), using perturbation theory, we get Φ to be a positive semi-definite matrix. Therefore, using *Schur Product Theorem* [121, p. 479, Theorem 7.5.3], if Φ and Ψ are positive semi-definite matrices, $\Psi' = \Phi \circ \Psi$ is also positive semi-definite.

□

Since Ψ' is a positive semidefinite matrix (from Theorem 24), the optimization problem $\big($as in Eq. (5.65)$\big)$ cannot be solved in (weakly) polynomial time [122](using the well-known ellipsoid method), and thus we propose a different technique to solve the optimization problem in polynomial time in the rest of the section.

Let $\Psi' = M^\top M$ [from Theorem 24]. Next, we convert the quadratic programming in Eq. (5.65) to a semidefinite programming [123], therefore be able to solve it in polynomial time.

We rewrite the optimization problem in Eq. (5.65) as follows (Detailed steps in Section 5.2.6.9):

$$
\begin{aligned}
\min_{\mathbf{c}',\theta} \quad & \theta \\
\text{s.t.} \quad & \forall_{0\le t\le H-1}\forall_{1\le r\le p-1} \quad \begin{bmatrix} 1 & 0 \\ 0 & -W_{t,r}^{\top}\mathbf{c}' \end{bmatrix} \succeq 0 \\
& \forall_{0\le t\le H-1}\forall_{1\le r\le p-1} \quad \begin{bmatrix} 1 & 0 \\ 0 & W_{t,r}^{\top}\mathbf{c}' \end{bmatrix} \succeq 0 \\
& \forall_{0\le i\le pH} \quad \begin{bmatrix} 1 & 0 \\ 0 & -1-E_t^{\top}\mathbf{c}' \end{bmatrix} \succeq 0 \\
& \forall_{0\le i\le pH} \quad \begin{bmatrix} I & 0 \\ 0 & E_t^{\top}\mathbf{c}' \end{bmatrix} \succeq 0 \\
& \begin{bmatrix} I & M\mathbf{c}' \\ \mathbf{c}'^{\top}M^{\top} & -(\mathcal{K}+\mathcal{K}_2)-(\mathcal{L}+\mathcal{R}+\beta\cdot\mathfrak{I})\mathbf{c}'+\theta \end{bmatrix} \succeq 0,
\end{aligned}
\tag{5.66}
$$

where, $R \succeq 0$ means R is a positive semidefinite matrix.

The above semidefinite programming, Eq. (5.66)*, can be solved in polynomial time (worst case)* [119]. This concludes our polynomial time proof.

5.2.6.6 Transformation of Choice Variables

Let $\mathbf{C}^c$ be the set of all possible choices. Let $\mathbf{c} = \{c_0, c_1, \cdots, c_{H-1}\} \in \mathbf{C}^c$ be a set of choices up-to time step $H-1$, where $c_i \in \mathbb{R}_{[0,1]}$.

$$
\mathbb{E}\left[\hat{\mathbf{s}}_c - \mathbf{s}\right] = \mathbb{E}\left[\mathcal{V}(\mathbf{c})\right] = \begin{bmatrix} (1-c_0)\cdot\mathbb{E}\left[\mathcal{D}_0^0\right]+c_0\cdot\mathbb{E}\left[\mathcal{D}_0^{W-1}\right] \\ \vdots \\ (1-c_t)\cdot\mathbb{E}\left[\mathcal{D}_t^0\right]+c_t\cdot\mathbb{E}\left[\mathcal{D}_0^{W-1}\right] \\ \vdots \\ (1-c_{H-1})\cdot\mathbb{E}\left[\mathcal{D}_{H-1}^0\right]+c_{H-1}\cdot\mathbb{E}\left[\mathcal{D}_{H-1}^{W-1}\right] \end{bmatrix}. \tag{5.67}
$$

Note that for a given t, $\mathbb{E}\left[\mathcal{D}_t^0\right]$ and $\mathbb{E}\left[\mathcal{D}_t^{W-1}\right]$ are also a p dimensional vector. Let the r-th element of these vectors be denoted as $\mathbb{E}\left[\mathcal{D}_t^0\right][r]$ and $\mathbb{E}\left[\mathcal{D}_t^{W-1}\right][r]$. Let $\mathbb{E}\left[\mathcal{D}_t^0\right][r] = \Gamma_{pt+r}^0$ and $\mathbb{E}\left[\mathcal{D}_t^{W-1}\right][r] = \Gamma_{pt+r}^1$. Now we can rewrite Eq. (5.67) with $\mathbf{c}'$ as follows:

$$
\mathbb{E}\left[\hat{\mathbf{s}}_c - \mathbf{s}\right] = \mathbb{E}\left[\mathcal{V}(\mathbf{c})\right] = \begin{bmatrix} (1-c_0)\cdot\Gamma_0^0+c_0\cdot\Gamma_0^1 \\ \vdots \\ (1-c_0)\cdot\Gamma_{p-1}^0+c_0\cdot\Gamma_{p-1}^1 \\ \vdots \\ (1-c_{H-1})\cdot\Gamma_{p(H-1)}^0+c_{H-1}\cdot\Gamma_{p(H-1)}^1 \\ \vdots \\ (1-c_{H-1})\cdot\Gamma_{pH-1}^0+c_{H-1}\cdot\Gamma_{pH-1}^1 \end{bmatrix}. \tag{5.68}
$$

We further note that a choice scalar variable c_t (Or, $1-c_t$) is multiplied with the vector Γ_t^k in Eq. (5.68). Similar to the above process, we can create copies of the scalar variable c_t as follows:

$$c_t \cdot \Gamma_t^k = \begin{bmatrix} c_t \cdot \Gamma_t^k & c_t \cdot \Gamma_{t+1}^k & \cdots & c_t \cdot \Gamma_{t+p-1}^k \end{bmatrix}^\top \tag{5.69}$$

$$= \begin{bmatrix} c'_t \cdot \Gamma_t^k & c'_{t+1} \cdot \Gamma_{t+1}^k & \cdots & c'_{t+p-1} \cdot \Gamma_{t+p-1}^k \end{bmatrix}^\top, \tag{5.70}$$

where $c_t = c'_t = \cdots = c'_{t+p-1}$. Using this process of unwrapping of vectors Γ_t^k and c_t to scalars, we rewrite Eq. (5.68) as follows:

$$\mathbb{E}\left[\hat{\mathbf{s}}_c - \mathbf{s}\right] = \mathbb{E}\left[\mathcal{V}(\mathbf{c})\right] = \begin{bmatrix} (1-c'_0)\cdot\Gamma_0^0 + c'_0 \cdot \Gamma_0^1 \\ \vdots \\ (1-c'_{pt+r})\cdot\Gamma_{pt+r}^0 + c'_t \cdot \Gamma_{pt+r}^1 \\ \vdots \\ (1-c'_{pH-1})\cdot\Gamma_{pH-1}^0 + c'_{pH-1} \cdot \Gamma_{pH-1}^1 \end{bmatrix}, \tag{5.71}$$

where $\forall_{t=0}^{H-1}\ c'_{p\cdot t} = \cdots = c'_{p\cdot t-1}$, and $\forall_t c'_t \in \mathbb{R}_{[0,1]}$.

5.2.6.7 Canonical Form

The objective function in Eq. (5.64) formulation can be rewritten as:

$$\alpha \cdot \left(\mathbb{E}\left[\mathcal{V}(\mathbf{c}')\right]^\top \times \Psi \times \mathbb{E}\left[\mathcal{V}(\mathbf{c}')\right]\right) + \alpha \cdot \mathbb{V} \cdot Var\left(\mathcal{V}(\mathbf{c}')\right) + \beta \cdot \mathcal{J}^{\text{per}}(\mathbf{c}') = \tag{5.72}$$

$$\alpha \cdot \left(\mathbb{E}\left[\mathcal{V}(\mathbf{c}')\right]^\top \times \Psi \times \mathbb{E}\left[\mathcal{V}(\mathbf{c}')\right]\right) + \mathcal{R} \cdot \mathbf{c}' + \mathcal{K}_2 + \beta \cdot \mathfrak{I}\mathbf{c}' = \tag{5.73}$$

$$\underbrace{\left(\sum_{i=0}^{pH}\sum_{j=0}^{pH} \mathbb{E}\left[\mathcal{V}(\mathbf{c}')\right][i] \cdot \mathbb{E}\left[\mathcal{V}(\mathbf{c}')\right][j] \cdot \alpha \cdot \Psi[i][j]\right)}_{\text{c-term}} + \mathcal{R} \cdot \mathbf{c}' + \mathcal{K}_2 + \beta \cdot \mathfrak{I}\mathbf{c}', \tag{5.74}$$

where $\alpha \cdot \mathbb{V} \cdot Var(\mathcal{V}(\mathbf{c}')) = \mathcal{R} \cdot \mathbf{c}' + \mathcal{K}_2$, and $\mathcal{J}^{\text{per}}(\mathbf{c}) = \mathfrak{I}\mathbf{c}'$. Note that, the c-term in Eq. (5.74) has quadratic, linear and constant terms in variables $\mathbf{c}'$. Therefore, we rewrite Eq. (5.74) as follows:

$$\left(\sum_{i=0}^{pH}\sum_{j=0}^{pH} c'_i \cdot c'_j \cdot \alpha \cdot \left(\Gamma_i^0\Gamma_j^0 + \Gamma_i^0\Gamma_j^1 + \Gamma_i^1\Gamma_j^0 + \Gamma_i^1\Gamma_j^1\right) \cdot \Psi[i][j]\right) + \mathcal{L}\mathbf{c}' + \mathcal{K} + \mathcal{R}\cdot\mathbf{c}' + \mathcal{K}_2 + \beta\cdot\Im\mathbf{c}', \tag{5.75}$$

where $\mathcal{L}\mathbf{c}'$ is the linear term (in variables $\mathbf{c}'$) after evaluating the c-term in Eq. (5.74), and $\mathcal{K}$ is the constant term. Note that we only evaluated the quadratic terms (and not the linear and constant term) because the complexity of a quadratic programming is dependent on the quadratic term [119, 111, 110].

5.2.6.8 The Quadratic Matrix is a Positive Semidefinite

Let, $\Phi \in \mathbb{R}^{p\cdot H\times p\cdot H}$ be as follows:

$$\Phi[i][j] = \Gamma_i^0\Gamma_j^0 + \Gamma_i^0\Gamma_j^1 + \Gamma_i^1\Gamma_j^0 + \Gamma_i^1\Gamma_j^1. \tag{5.76}$$

Therefore, $\Psi' = \Phi \circ \Psi$, where $A \circ B$, denotes the Hadamard product of two matrices A and B.

Let $\delta = mean_{i,j}\{\Phi[i][j]\}$. We define a matrix, $\Delta \in \mathbb{R}^{p\cdot H\times p\cdot H}$, of same element, *i.e.*, $\forall_{i,j}\Delta[i][i] = \delta$. Therefore, we write Φ as follows:

$$\Phi = \underbrace{\tilde{\varepsilon}I + \Delta}_{\Phi_c} + \underbrace{\begin{bmatrix} \varepsilon_{0,0}-\tilde{\varepsilon} & \varepsilon_{1,1} & \cdots & \varepsilon_{0,p\cdot H-1} \\ \varepsilon_{1,0} & \varepsilon_{1,1}-\tilde{\varepsilon} & \cdots & \varepsilon_{1,p\cdot H-1} \\ \vdots & \vdots & \ddots & \vdots \\ \varepsilon_{p\cdot H-1,0} & \varepsilon_{p\cdot H-1,1} & \cdots & \varepsilon_{p\cdot H-1,p\cdot H-1}-\tilde{\varepsilon} \end{bmatrix}}_{\Phi_\delta}, \tag{5.77}$$

where $I \in \mathbb{R}^{p\cdot H\times p\cdot H}$ is an identity matrix, $\tilde{\varepsilon} > 0$, $\varepsilon_{i,j} = \left(\Gamma_i^0\Gamma_j^0 + \Gamma_i'^0\Gamma_j^1 + \Gamma_i^1\Gamma_j'^0 + \Gamma_i^1\Gamma_j^1\right) - \delta$.

Note that Δ is singular but Φ_c is non-singular; this being the main reason to include the term $\tilde{\varepsilon}I$ in Eq. (5.77), as it will be used in our further proofs.

We assume the following on the perception model:

Assumption 3.

$$\frac{||\Phi_\delta||_2}{||\Phi_c||_2} \leq \frac{1}{\kappa(\Phi_c)} \tag{5.78}$$

Where $\kappa(\Phi_c)$ is defined as the condition number of a matrix, $\kappa(\Phi_c) = ||\Phi_c||_2 \cdot ||\Phi_c^{-1}||_2$.

If the assumption in Assumption 3 is satisfied, we get Φ to be a positive semi-definite matrix, from perturbation theory.

5.2.6.9 Casting to a Semidefinite Program

The optimization formulation given in Eq. (5.65) can be rewritten as:

$$\begin{aligned} \min_{\mathbf{c}',\theta} \quad & \theta \\ \text{s.t.} \quad & \forall_{0\le t\le H-1}\forall_{1\le r\le p-1} \quad W_{t,r}^{\top}\mathbf{c}' = 0 \\ & \forall_{0\le t\le H-1} \quad 0 \le E_t^{\top}\mathbf{c}' \le 1 \\ & \mathbf{c}'^{\top}\Psi'\mathbf{c}' + (\mathcal{L}+\mathcal{R}+\beta\cdot\mathfrak{I})\mathbf{c}' + \mathcal{K}+\mathcal{K}_2 - \theta \le 0. \end{aligned} \tag{5.79}$$

We further rewrite the above optimization as follows:

$$\begin{aligned} \min_{\mathbf{c}',\theta} \quad & \theta \\ \text{s.t.} \quad & \forall_{0\le t\le H-1}\forall_{1\le r\le p-1} \quad W_{t,r}^{\top}\mathbf{c}' \le 0 \\ & \forall_{0\le t\le H-1}\forall_{1\le r\le p-1} \quad -W_{t,r}^{\top}\mathbf{c}' \le 0, \\ & \forall_{0\le t\le H-1} \quad E_t^{\top}\mathbf{c}' - 1 \le 0 \\ & \forall_{0\le t\le H-1} \quad -E_t^{\top}\mathbf{c}' \le 0 \\ & \mathbf{c}'^{\top}\Psi'\mathbf{c}' + (\mathcal{L}+\mathcal{R}+\beta\cdot\mathfrak{I})\mathbf{c}' + \mathcal{K}+\mathcal{K}_2 - \theta \le 0. \end{aligned} \tag{5.80}$$

Let $\Psi' = M^{\top}M$ [$\because$ Theorem 24].

Next, we express the optimization formulation in Eq. (5.80) as a standard semidefinite programming [123].

$$
\begin{aligned}
\min_{\mathbf{c}',\theta} \quad & \theta \\
\text{s.t.} \quad & \forall_{0\le t\le H-1}\forall_{1\le r\le p-1} \quad \begin{bmatrix} 1 & 0 \\ 0 & -W_{t,r}^{\top}\mathbf{c}' \end{bmatrix} \succeq 0 \\
& \forall_{0\le t\le H-1}\forall_{1\le r\le p-1} \quad \begin{bmatrix} 1 & 0 \\ 0 & W_{t,r}^{\top}\mathbf{c}' \end{bmatrix} \succeq 0 \\
& \forall_{0\le i\le pH} \quad \begin{bmatrix} 1 & 0 \\ 0 & -1-E_t^{\top}\mathbf{c}' \end{bmatrix} \succeq 0 \\
& \forall_{0\le i\le pH} \quad \begin{bmatrix} I & 0 \\ 0 & E_t^{\top}\mathbf{c}' \end{bmatrix} \succeq 0 \\
& \begin{bmatrix} I & M\mathbf{c}' \\ \mathbf{c}'^{\top}M^{\top} & -(\mathcal{K}+\mathcal{K}_2)-(\mathcal{L}+\mathcal{R}+\beta\cdot\mathfrak{I})\mathbf{c}'+\theta \end{bmatrix} \succeq 0,
\end{aligned}
\tag{5.81}
$$

where, $R \succeq 0$ means R is a positive semidefinite matrix.

Bibliography

Bineet Ghosh, Clara Hobbs, Shengjie Xu, Parasara Sridhar Duggirala, James H. Anderson, P. S. Thiagarajan, and Samarjit Chakraborty. Statistical hypothesis testing of controller implementations under timing uncertainties. In *2022 IEEE 28th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 11–20, 2022.

Gary Doran, Steven Lu, Lukas Mandrake, and Kiri Wagstaff. Mars orbital image (HiRISE) labeled data set version 3, 2019.

Manabu Nakanoya, Sandeep Chinchali, Alexandros Anemogiannis, Akul Datta, Sachin Katti, and Marco Pavone. Task-relevant representation learning for networked robotic perception. *CoRR*, abs/2011.03216, 2020.

Naghmeh Niknejad, Waidah Binti Ismail, Abbas Mardani, Huchang Liao, and Imran Ghani. A comprehensive overview of smart wearables: The state of the art literature, recent advances, and future challenges. *Eng. Appl. Artif. Intell.*, 90(C), 2020.

Veton Këpuska and Gamal Bohouta. Next-generation of virtual personal assistants (microsoft cortana, apple siri, amazon alexa and google home). In *IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, 2018.

Mengyu Fu, Alan Kuntz, Robert J. Webster, and Ron Alterovitz. Safe motion planning for steerable needles using cost maps automatically extracted from pulmonary images. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

Clara Hobbs, Debayan Roy, Parasara Sridhar Duggirala, F. Donelson Smith, Soheil Samii, James H. Anderson, and Samarjit Chakraborty. Perception computing-aware controller synthesis for autonomous systems. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021.

M. Maggio et al. Control-System Stability Under Consecutive Deadline Misses Constraints. In *32nd Euromicro Conference on Real-Time Systems (ECRTS)*, 2020.

Petter Nilsson, Omar Hussien, Ayca Balkan, Yuxiao Chen, Aaron D. Ames, Jessy W. Grizzle, Necmiye Ozay, Huei Peng, and Paulo Tabuada. Correct-by-construction adaptive cruise control: Two approaches. *IEEE Transactions on Control Systems Technology*, 24(4):1294–1307, 2016.

Victor Gan, Guy Albert Dumont, and Ian Mitchell. Benchmark problem: A PK/PD model and safety constraints for anesthesia delivery. In *ARCH@CPSWeek*, volume 34 of *EPiC Series in Computing*, pages 1–8. EasyChair, 2014.

Ratan Lal and Pavithra Prabhakar. Bounded error flowpipe computation of parameterized linear systems. In Alain Girault and Nan Guan, editors, *Proceedings of the 2015 International Conference on Embedded Software (EMSOFT 2015)*, pages 237–246. IEEE, 2015.

Masaki Waga, Étienne André, and Ichiro Hasuo. Model-bounded monitoring of hybrid systems. *ACM Transactions on Cyber-Physical Systems*, 6(4):30:1–30:26, November 2022.

Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *Proceedings of the 8th International Conference on Hybrid Systems: Computation and Control*, HSCC'05, page 291–305, Berlin, Heidelberg, 2005. Springer-Verlag.

Goran Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, pages 258–273, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

Stanley Bak and Parasara Sridhar Duggirala. Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, HSCC '17, page 173–178, New York, NY, USA, 2017. Association for Computing Machinery.

Colas Guernic and Antoine Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4, 05 2010.

Bineet Ghosh and Parasara Sridhar Duggirala. Robust reachable set: Accounting for uncertainties in linear dynamical systems. *ACM Trans. Embed. Comput. Syst.*, 18(5s), oct 2019.

Bineet Ghosh and Parasara Sridhar Duggirala. Robustness of safety for linear dynamical systems: Symbolic and numerical approaches. Technical Report 2109.07632, arXiv, 2021.

Bineet Ghosh and Parasara Sridhar Duggirala. Reachability of linear uncertain systems: Sampling based approaches, 2021.

Raena Farhadsefat, Jirı Rohn, and Taher Lotfi. Norms of interval matrices. 2011.

Matthias Althoff, Colas Le Guernic, and Bruce H Krogh. Reachable set computation for uncertain time-varying linear systems. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 93–102, 2011.

Pierre-Jean Meyer, Alex Devonport, and Murat Arcak. Tira: Toolbox for interval reachability analysis. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, page 224–229, New York, NY, USA, 2019. Association for Computing Machinery.

Wikipedia contributors. Singular value decomposition — Wikipedia, the free encyclopedia, 2021. [Online; accessed 27-March-2021].

Torsten Söderström. Perturbation results for singular values. Technical Report 1999-001, Department of Information Technology, Uppsala University, April 1999.

Bo Kågström. Bounds and perturbation bounds for the matrix exponential. *BIT Numerical Mathematics*, 17(1):39–57, 1977.

Charles Van Loan. The sensitivity of the matrix exponential. *SIAM Journal on Numerical Analysis*, 14(6):971–981, 1977.

Bineet Ghosh and Étienne André. Offline and online monitoring of scattered uncertain logs using uncertain linear dynamical systems. In Mohammad Reza Mousavi and Anna Philippou, editors, *Formal Techniques for Distributed Objects, Components, and Systems*, pages 67–87, Cham, 2022. Springer International Publishing.

Bineet Ghosh and Étienne André. Offline and online energy-efficient monitoring of scattered uncertain logs using a bounding model, 2022.

Bineet Ghosh and Étienne André. `MoULDyS`: Monitoring of autonomous systems in the presence of uncertainties, 2023.

Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *International Workshop on Hybrid Systems: Computation and Control*, pages 291–305. Springer, 2005.

Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*, pages 379–395. Springer, 2011.

Stanley Bak and Parasara Sridhar Duggirala. Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, pages 173–178, 2017.

Debayan Roy, Licong Zhang, Wanli Chang, Dip Goswami, Birgit Vogel-Heuser, and Samarjit Chakraborty. Tool integration for automated synthesis of distributed embedded controllers. *ACM Trans. Cyber-Phys. Syst.*, 6(1), nov 2021.

Sumana Ghosh, Arnab Mondal, Debayan Roy, Philipp H. Kindt, Soumyajit Dey, and Samarjit Chakraborty. Proactive feedback for networked cps. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, SAC '21, page 164–173, New York, NY, USA, 2021. Association for Computing Machinery.

Hao Liu, Ben Niu, and Jiahu Qin. Reachability analysis for linear discrete-time systems under stealthy cyber attacks. *IEEE Transactions on Automatic Control*, 66(9):4444–4451, 2021.

Jiangnan Cheng, Marco Pavone, Sachin Katti, Sandeep Chinchali, and Ao Tang. Data sharing and compression for cooperative networked control. *Advances in Neural Information Processing Systems*, 34, 2021.

Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM review*, 45(1):3–49, 2003.

Aditya Zutshi, Sriram Sankaranarayanan, and Ashish Tiwari. Timed relational abstractions for sampled data control systems. In *International Conference on Computer Aided Verification*, pages 343–361. Springer, 2012.

Parasara Sridhar Duggirala and Mahesh Viswanathan. Parsimonious, simulation based verification of linear systems. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification (CAV)*, 2016.

Stanley Bak and Parasara Sridhar Duggirala. Simulation-equivalent reachability of large linear systems with inputs. In *International Conference on Computer Aided Verification*, pages 401–420. Springer, 2017.

Sadra Sadraddini and Russ Tedrake. Linear encodings for polytope containment problems, 2019.

Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

Fredrik Johansson et al. *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 0.18)*, December 2013. `http://mpmath.org/`.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020.

Pythonrobotics. `https://pythonrobotics.readthedocs.io/en/latest/`.

Bineet Ghosh, Sandeep Chinchali, and Parasara Sridhar Duggirala. Interpretable trade-offs between robot task accuracy and compute efficiency. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5364–5371, 2021.

Olaf Stursberg and Bruce H. Krogh. Efficient representation and computation of reachable sets for hybrid systems. In *Hybrid Systems: Computation and Control (HSCC)*, 2003.

R. Diwakaran et al. Analyzing neighborhoods of falsifying traces in cyber-physical systems. In *8th International Conference on Cyber-Physical Systems (ICCPS)*, 2017.

G. C. Calafiore and M. C. Campi. The scenario approach to robust control design. *IEEE Transactions on Automatic Control*, 51(5):742–753, 2006.

Bai Xue et al. Pac model checking of black-box continuous-time dynamical systems, 2020.

André Platzer and Edmund M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In *Formal Methods (FM)*, 2009.

Nicole Chan and Sayan Mitra. Verifying safety of an autonomous spacecraft rendezvous mission. In *ARCH17. 4th International Workshop on Applied Verification of Continuous and Hybrid Systems*.

Nikolaos Kekatos et al. Lane change maneuver for autonomous vehicles (benchmark proposal). In *ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems*.

S. Kaynama and C. Tomlin. Benchmark: Flight envelope protection in autonomous quadrotors. 2014.

Hongxu Chen et al. Motor-transmission drive system: a benchmark example for safety verification. In *ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems*, 2015.

Thomas Heinz et al. Benchmark: Reachability on a model with holes. In *ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems*, 2015.

Anna-Kathrin Kopetzki, Bastian Schürmann, and Matthias Althoff. Methods for order reduction of zonotopes. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 5626–5633, 2017.

Xuejiao Yang and Joseph K. Scott. A comparison of zonotope order reduction techniques. *Automatica*, 95:378–384, 2018.

`MoULDyS` installation guide. `https://www.github.com/bineet-coderep/MoULDyS/blob/main/documentation/installation_guide.md`, 2022.

`MoULDyS` user guide. `https://www.github.com/bineet-coderep/MoULDyS/blob/main/documentation/user_guide.pdf`, 2022.

Bineet Ghosh, Clara Hobbs, Shengjie Xu, Parasara Sridhar Duggirala, James H. Anderson, P. S. Thiagarajan, and Samarjit Chakraborty. Statistical verification of autonomous system controllers under timing uncertainties. 2023.

Clara Hobbs, Bineet Ghosh, Shengjie Xu, Parasara Sridhar Duggirala, and Samarjit Chakraborty. Safety analysis of embedded controllers under implementation platform timing uncertainties. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11):4016–4027, 2022.

Shengjie Xu, Bineet Ghosh, Clara Hobbs, P. S. Thiagarajan, and Samarjit Chakraborty. Safety-aware flexible schedule synthesis for cyber-physical systems using weakly-hard constraints. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, ASPDAC '23, page 46–51, New York, NY, USA, 2023. Association for Computing Machinery.

Shengjie Xu, Bineet Ghosh, Clara Hobbs, P.S Thiagarajan, Prachi Desai, and Samarjit Chakraborty. Safety-aware implementation of control tasks via scheduling with period boosting and compressing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Shengjie Xu, Bineet Ghosh, Clara Hobbs, Enrico Fraccaroli, Parasara Sridhar Duggirala, and Samarjit Chakraborty. Statistical approach to efficient and deterministic schedule synthesis for cyber-physical systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

M. Fisher et al. Verifying autonomous systems. *Commun. ACM*, 56(9):84–93, 2013.

J. Wing. Trustworthy AI. *Commun. ACM*, 64(10):64–71, 2021.

L. Dennis and M. Fisher. Verifiable self-aware agent-based autonomous systems. *Proc. IEEE*, 108(7):1011–1026, 2020.

Alejandro Masrur, Sebastian Drössler, Thomas Pfeuffer, and Samarjit Chakraborty. VM-based real-time services for automotive control applications. In *16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2010.

P. Axer et al. Building timing predictable embedded systems. *ACM Trans. Embedded Comput. Syst.*, 13(4):82:1–82:37, 2014.

L. Thiele and R. Wilhelm. Design for timing predictability. *Real-Time Systems*, 28(2-3):157–177, 2004.

R. Wilhelm. Real time spent on real time. *Commun. ACM*, 63(10):54–60, 2020.

P. Pazzaglia et al. Beyond the weakly hard model: Measuring the performance cost of deadline misses. In *30th Euromicro Conference on Real-Time Systems (ECRTS)*, 2018.

D. Soudbakhsh et al. Co-design of arbitrated network control systems with overrun strategies. *IEEE Trans. Control. Netw. Syst.*, 5(1):128–141, 2018.

R. Blind and F. Allgöwer. Towards networked control systems with guaranteed stability: Using weakly hard real-time constraints to model the loss process. In *54th IEEE Conference on Decision and Control (CDC)*, 2015.

D. Liberzon. *Switching in systems and control*. Springer, 2003.

W. Zhang et al. Stability of networked control systems. *IEEE Control Systems Magazine*, 21(1):84–99, 2001.

Matthew O'Kelly, Hongrui Zheng, Dhruv Karthik, and Rahul Mangharam. F1tenth: An open-source evaluation environment for continuous control and reinforcement learning. *Proceedings of Machine Learning Research*, 123:77–89, 2020.

R. Kass and A. Raftery. Bayes factors. *Journal of the American Statistical Association*, 90(430):773–795, 1995.

O. Bernardi and O. Giménez. A linear algorithm for the random sampling from regular languages. *Algorithmica*, 62:130–145, 2010.

Karl J. Åström and Björn Wittenmark. *Computer-Controlled Systems (3rd Ed.)*. Prentice-Hall, Inc., USA, 1997.

Willem Hagemann. Reachability analysis of hybrid systems using symbolic orthogonal projections. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification (CAV)*, 2014.

Sadra Sadraddini and Russ Tedrake. Linear encodings for polytope containment problems. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019.

Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *Proceedings of the 8th International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2005.

Komei Fukuda. From the zonotope construction to the minkowski addition of convex polytopes. *Journal of Symbolic Computation*, 38(4):1261–1272, 2004. Symbolic Computation in Algebra and Geometry.

P. Flajolet et al. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1):1–35, 1994.

Robert A Gabel and Richard A Roberts. *Signals and linear systems*. John Wiley & Sons, USA, 1991.

Bineet Ghosh, Masaad Khan, Adithya Ashok, Sandeep Chinchali, and Parasara Sridhar Duggirala. Dynamic selection of perception models for robotic control, 2022.

Sandeep Chinchali, Apoorva Sharma, James Harrison, Amine Elhafsi, Daniel Kang, Evgenya Pergament, Eyal Cidon, Sachin Katti, and Marco Pavone. Network offloading policies for cloud robotics: A learning-based approach. In *Proceedings of Robotics: Science and Systems*, Freiburg im Breisgau, Germany, June 2019.

Akhlaqur Rahman, Jiong Jin, Antonio Cricenti, Ashfaqur Rahman, and Manoj Panda. Motion and connectivity aware offloading in cloud robotics via genetic algorithm. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–6. IEEE, 2017.

Maya Cakmak and Andrea L Thomaz. Designing robot learners that ask good questions. In *2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 17–24. IEEE, 2012.

David Whitney, Eric Rosen, James MacGlashan, Lawson LS Wong, and Stefanie Tellex. Reducing errors in object-fetching interactions through social feedback. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1006–1013. IEEE, 2017.

Krishnanand N Kaipa, Akshaya S Kankanhalli-Nagendra, Nithyananda B Kumbla, Shaurya Shriyam, Srudeep Somnaath Thevendria-Karthic, Jeremy A Marvel, and Satyandra K Gupta. Enhancing robotic unstructured bin-picking performance by enabling remote human interventions in challenging perception scenarios. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 639–645. IEEE, 2016.

Amir Erfan Eshratifar and Massoud Pedram. Runtime deep model multiplexing for reduced latency and energy consumption inference. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pages 263–270. IEEE, 2020.

Nicolai Dorka, Johannes Meyer, and Wolfram Burgard. Modality-buffet for real-time object detection. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10543–10549, 2020.

Thinh Quang Dinh, Quang Duy La, Tony QS Quek, and Hyundong Shin. Learning for computation offloading in mobile edge computing. *IEEE Transactions on Communications*, 66(12):6353–6367, 2018.

Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. In *International Conference on Learning Representations*, 2019.

Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Christos Boutsidis, Petros Drineas, and Malik Magdon-Ismail. Near-optimal coresets for least-squares regression. *IEEE Transactions on Information Theory*, 59(10):6880–6892, 2013.

Kyle D. Julian, Ritchie Lee, and Mykel J. Kochenderfer. Validation of image-based neural network controllers through adaptive stress testing. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–7, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

X-plane 11. `https://www.x-plane.com/`.

Sydney M. Katz, Anthony Corso, Sandeep Chinchali, Amine Elhafsi, Apoorva Sharma, Marco Pavone, and Mykel J. Kochenderfer. Nasa uli aircraft taxi dataset, 2021. Stanford Research Data, `https://purl.stanford.edu/zz143mb4347`.

Bineet Ghosh, Sandeep Chinchali, and P. S. Duggirala. Interpretable trade-offs between robot task accuracy and compute efficiency. Technical report, University of North Carolina at Chapel Hill, 2021.

Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

Bineet Ghosh et al. Interpretable trade-offs between robot task accuracy and compute efficiency. In *IROS*, 2021.

Jiangnan Cheng et al. Data sharing and compression for cooperative networked control. *Advances in Neural Information Processing Systems*, 34, 2021.

Shital Shah et al. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*. Springer International Publishing, 2018.

Sartaj Sahni. Computationally related problems. *SIAM J. Comput.*, 3, 1974.

P. M. Pardalos and S. A. Vavasis. Quadratic programming with one negative eigenvalue is np-hard. *Journal of Global Optimization*, 1, 1991.

Alberto Bemporad and Vihangkumar V. Naik. A numerically robust mixed-integer quadratic programming solver for embedded hybrid model predictive control. 2018. NMPC.

Robert Hult et al. An miqp-based heuristic for optimal coordination of vehicles at intersections. In *IEEE Conference on Decision and Control (CDC)*, 2018.

Cenk Baykal et al. Data-dependent coresets for compressing neural networks with applications to generalization bounds. In *ICLR*, 2019.

Lucas Liebenwein et al. Provable filter pruning for efficient neural networks. In *ICLR*, 2020.

Christos Boutsidis et al. Near-optimal coresets for least-squares regression. *IEEE Transactions on Information Theory*, 59, 2013.

Surat Teerapittayanon et al. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, 2016.

Priyadarshini Panda et al. Conditional deep learning for energy-efficient and enhanced pattern recognition. DATE '16, 2016.

Lieven Vandenberghe et al. *SIAM Review*, 38, 1996.

F. Ellert and C. Merriam. Synthesis of feedback controls using optimization theory–an example. *IEEE Transactions on Automatic Control*, 1963.

Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 2 edition, 2012.

M.K. Kozlov et al. The polynomial solvability of convex quadratic programming. *USSR Computational Mathematics and Mathematical Physics*, 20, 1980.

Robert M. Freund. Introduction to semidefinite programming (sdp). MIT Lecture Notes.