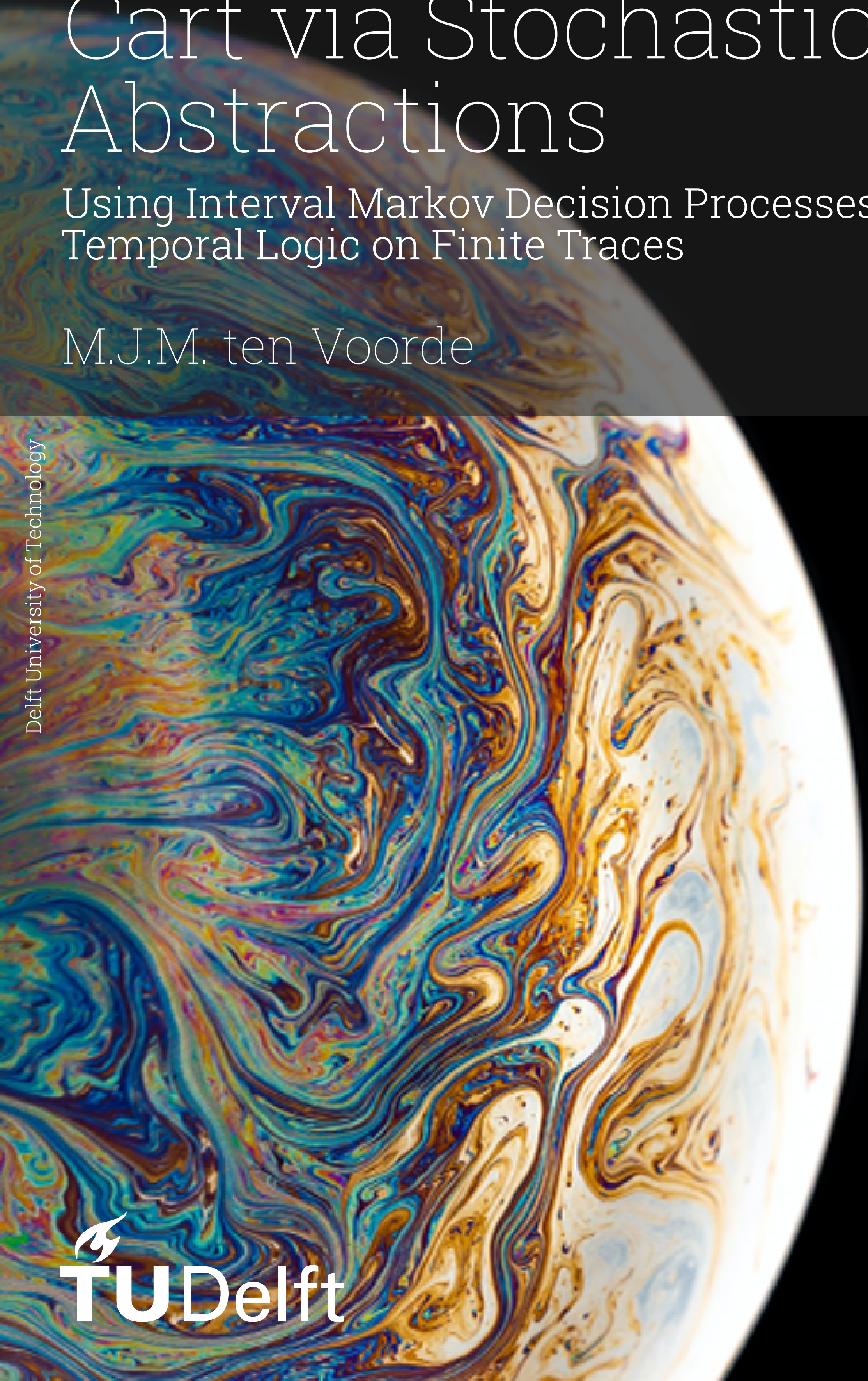
Formal Control of an Inverted Pendulum on a Cart via Stochastic Abstractions

Using Interval Markov Decision Processes and Linear Temporal Logic on Finite Traces

M.J.M. ten Voorde

Delft University of Technology



TU Delft

DCSC

Formal Control of an Inverted Pendulum on a Cart via Stochastic Abstractions

Using Interval Markov Decision Processes and Linear Temporal Logic on Finite Traces

by

M.J.M. ten Voorde

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Friday March 22, 2024 at 2:00 PM.

Student number:	4590740	
Project duration:	September 1, 2023 – March 22, 2024	
Thesis committee:	S.J.L. Adams,	TU Delft, daily supervisor
	Dr. L. Laurenti,	TU Delft, other supervisor

Cover: Olah, D. (2018). Oil drop. Unsplash. photograph. Retrieved from https://unsplash.com/photos/04RhrsalOmU. Under Unsplash License (Modified)

Style: TU Delft Report Style, with modifications by Daan Zwaneveld (Modified)

TU Delft

Preface

I would like to thank my supervisors, Steven and Luca, for their expert guidance and support throughout this work. I also extend my thanks to the committee members for their engagement and insights during my thesis defense.

The completion of this project has been a long and challenging journey, one that I could not have navigated without the support of the many wonderful people that I hold dear. It is here that I deeply acknowledge and cherish those who have been there for me every step of the way.

My dear parents, Hans and Dinette, and my brother, Floris—I cannot express enough how grateful I am for your love, support, patience, and belief in me. It has been a constant from the day I arrived here in Delft to the very end of this journey and is something that will undoubtedly extend far beyond.

Above all mentioned here, there is one person without whom this project might never have come to fruition. A person who has made it her life goal to see me succeed. My beloved Josipa, the countless moments of stress, sadness, and sometimes hopelessness could not have been navigated without you. Whether it was through a stern but motivating talk or, in the deepest moments, through affection, love, and mental support, you constantly pushed me forward towards an end which I often could not see myself.

You always strived to understand what I was working on and what was going on inside my head. We made plans and checklists together, celebrating every little thing we could check off. Staying up with me even when the nights became long. When the big milestones began happening, your jumping and screaming of happiness showed your deep care and passion for me, giving me the courage and motivation to push through until the end.

I could not have done it without you and can’t wait for our next chapter.

Volim te puno, Josipa.

M.J.M. ten Voorde
Delft, March 2024

Abstract

The use of machine learning (ML), especially neural networks, in modeling control systems has shown promise, particularly for systems with complex physics. However, applying these models in safety-critical areas requires reliable verification and control synthesis methods due to their inherent complexity. Formal methods, using stochastic finite state models like interval Markov decision processes (IMDPs), provide a way to analyze and verify these systems against detailed safety and performance specifications defined using linear temporal logic over finite traces (LTLf). Abstraction of ML models into such IMDPs, allows the deriving of formal guarantees on the IMDP that carryover to the underlying ML model.

This thesis focuses on designing a switched controller for a cart-pendulum system using neural network dynamic models (NNDM) by formal control synthesis, validating it through formal verification methods. The methodology includes modeling the system behavior under different controllers, abstracting these models into IMDPs, applying the respective formal methods, and validating the approach through experiments. The aim is to demonstrate the framework's utility in a practical context, comparing different neural network architectures and researching the applicability of formal guarantees to both the models and the actual system.

The main contributions are a practical application of the framework to a specific system, a comparison of neural network architectures for dynamic modeling, and an experiment-based validation of the framework's effectiveness. It confirms that the formal guarantees for abstracted models are relevant to the actual system, providing insights into the framework's potential for real-world applications. The findings suggest areas for further research, particularly in making such frameworks more accessible for practical deployment in safety-critical systems.

Contents

Front Matter i

Preface i

Abstract ii

Main Matter 1

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Objectives and Approach . . . 2
- 1.3 Contribution . . . 2
- 1.4 Outline . . . 4

2 Background 5
- 2.1 Formal Abstraction of Dynamic Stochastic Systems . . . 5
  - 2.1.1 Interval Markov Decision Processes . . . 5
  - 2.1.2 Calculating IMDP Transition Probability Bounds . . . 6
- 2.2 Formal Control Synthesis Over LTLf Specifications . . . 8
  - 2.2.1 Complex Specifications Using Ltlf . . . 8
  - 2.2.2 Control Synthesis . . . 9

3 Method 10
- 3.1 Problem Statement . . . 10
- 3.2 System: Inverted Pendulum on a Cart . . . 11
  - 3.2.1 Dynamics . . . 12
  - 3.2.2 Simulation & Data Acquisition . . . 13
- 3.3 Neural Network Dynamic Models . . . 14
  - 3.3.1 Modes . . . 14
  - 3.3.2 Network Architecture . . . 15
- 3.4 Formal Verification and Control Synthesis . . . 17
  - 3.4.1 Abstraction and Synthesis . . . 17
  - 3.4.2 Approximating the Probability of Satisfying Specifications . . . 18
  - 3.4.3 Determine Standard Deviation for Zero-Mean Gaussian Model Noise . . . 18

4 Experimental Results 19
- 4.1 Neural Network Dynamic Modeling Results . . . 19
  - 4.1.1 Post-Training Analysis . . . 19
  - 4.1.2 Discussion and Conclusion . . . 20
- 4.2 Experiment: Formal Verification . . . 21
  - 4.2.1 Experiment conditions . . . 21
  - 4.2.2 Evaluating the Abstraction . . . 23
  - 4.2.3 Evaluating the Classification . . . 23
- 4.3 Experiment: Formal Control Synthesis . . . 25
  - 4.3.1 Experiment conditions . . . 25
  - 4.3.2 Evaluating the Classification . . . 26
  - 4.3.3 Evaluating the Synthesized Policy . . . 27

5 Conclusion 29

End Matter **31**

Bibliography **31**

List of Figures

2.1 Grid based discretization of an arbitrary 2D state space, with destination states d and obstacles states o 8

3.1 Diagram of the inverted pendulum on a cart, with states in blue and forces in red. 12
3.2 Diagram illustrating the Direct Neural Network Architecture. Including the data streams, standardization steps, Neural Network location and role. 15
3.3 Diagram illustrating the Residual Neural Network Architecture. Including the data streams, standardization steps, Neural Network location and role, and the time-step h 16
3.4 Diagram illustrating the Physics Informed Neural Network Architecture. Including the data streams, standardization steps, Neural Network location and role, and the time-step h . 16

4.1 Comparison between different neural network architectures in terms of recursive simulation starting from the same initial state $x_0 = [0.5, 0.5, 0, 0]$ under the influence of the Modes 0, 1, 2 respectively for $k = 10$ each. 22
4.2 Visualization of State-Space, LTLf Specification's desired region indicated in green, and a model trajectory under the Zero Force Mode 23
4.3 Comparative visualization of neural network single-step predictions (red dots) and its under- and overapproximation determined by the CROWN Algorithm (red rectangle), from a uniformly sampled input region (turquoise rectangle) 24
4.4 Approximation of satisfying the specification under the Zero Force mode for a 2D slice of the states-space, characterizing regions with uniform velocities in the range of $[-0.45, 0]$ m/s for cart velocity and $[0, 0.6]$ rad/s for pendulum angular velocity. 24
4.5 Visualization of State-Space, LTLf Specification's desired region indicated in green, and a model trajectory satisfying the specification for $k \leq 20$ time-steps 26
4.6 Approximation of satisfying the specification under the avaiable mode for a 2D slice of the states-space, characterizing regions with uniform velocities in the range of $[-0.7, 0]$ m/s for cart velocity and $[0, 0.96]$ rad/s for pendulum angular velocity. 26
4.7 Complementary view of the synthesized policy, showcasing the lookup table in (a) and the action to take at $k = 0$ for the classification slice in (b) 27
4.8 Visualization of three sets of NN and System trajectories under the influence of the synthesized policy. Where the dashed lines represents the system trajecotries and the full lines the NN trajecotries 28

List of Tables

3.1 ODE parameters 13

4.1 Performance Comparison of Neural Network Architectures for Mode 0: Stabilizing the Pendulum with Cart Movement in the Positive Direction. 20
4.2 Performance Comparison of Neural Network Architectures for Mode 1: Stabilizing the Pendulum with the Cart Stationary. 20
4.3 Performance Comparison of Neural Network Architectures for Mode 2: Stabilizing the Pendulum with Cart Movement in the Negative Direction 21

1 Introduction

In this chapter, the foundation for the research approach is laid out in Section 1.1, where the motivation and reasoning behind the chosen methods are presented. Section 1.2 then outlines the objectives of this study and provides a brief overview of the approach taken to meet these goals. Following this, the contributions made by this thesis are listed in Section 1.3, aiming to offer clarity on what this work adds to the existing body of knowledge. The chapter concludes with Section 1.4, which organizes the thesis structure, offering readers a clear path through the content that follows.

1.1. Motivation

Machine learning (ML) methods, notably neural networks (NN), have become indispensable in modeling the dynamics of control systems, particularly those with complex or unknown physics. Their adoption in safety-critical domains—where failures could lead to severe consequences for individuals, the environment, or assets—underscores the urgent need for methods to verify these systems' performance reliably. However, the inherent "black-box" nature and complexity of ML models introduce significant challenges. Classical performance specifications such as stability, invariance, and fundamental properties like controllability and observability are difficult to assert with strong guarantees in the context of ML models.

Formal methods, deeply rooted in computer science, utilize (stochastic) finite state transition models, such as Markov Decision Processes (MDPs), to facilitate analysis, verification, and control synthesis. These methods are adept at evaluating systems against "complex" or "rich" specifications, articulated through logic languages that encompass notions of safety, liveness, and fairness. Ones such a temporal logic language that is capable of this is Linear Temporal Logic over finite traces (LTLf) (Giacomo & Vardi, n.d.). Which allows reasoning over specifications/events with finite durations, common in real-world situations. The application of formal methods to ML models, especially neural networks, unlocks potential for establishing robust performance guarantees. This necessitates transforming ML models into a form compatible with formal methods, typically through abstraction to a (stochastic) finite state transition model.

Despite neural networks' capability to model and control systems with intricate physics, their deployment in real-world, safety-critical applications like UAVs and autonomous vehicles remains limited. This hesitancy is attributed to the paramount importance of safety guarantees in such systems, where failures can lead to catastrophic outcomes. Recent advancements in convex relaxation techniques for neural networks (Anderson et al., 2020) have spurred abstraction methods that facilitate the translation of Neural Network Dynamical Systems (NNDMs) into (stochastic) finite state transition models (Adams et al., 2022). This transformation is critical for applying formal methods to verify and synthesize control systems, ensuring they meet comprehensive (performance) specifications of safety, liveness, and fairness.

The current body of literature, however, remains overly focused on theoretical and mathematical validations, applying primarily to the neural network model without considering its implications on the actual system it controls. This gap highlights a disconnect between theory and practical application, with little research addressing the translation of formal safety guarantees from the neural network model to the latent system. This thesis aims to bridge this divide by not only developing functional neural network dynamic systems but also by experimentally validating the effectiveness of formal methods frameworks. This work focuses on designing a switched controller. The controller is engineered to execute complex behaviors expressed through linear temporal logic. An example of such behavior includes guiding a cart-pole setup to reach specific positions and angles. Through this approach, the aim is to demonstrate the effectiveness of formal abstraction methods. Specifically, it seeks to show how these methods can offer guarantees for the designed switched controller, ensuring its performance aligns with our model expectations. Furthermore, experimental validation will explore the extent to which these guarantees are applicable to the true underlying system, aiming to transition from theoretical frameworks to tangible, real-world applications.

1.2. Objectives and Approach

Standard tools from control theory, such as Linear Quadratic Regulators (LQR) with linearization of the dynamics, often offer (optimal) control for linear or linearized systems near an equilibrium point. It does however not account for non-linear dynamics and the inherent uncertainties present in more complex behaviors or scenarios outside of these near-equilibrium conditions. The application of Stochastic Neural network dynamic models (NNDM) enables the design of controllers that can handle non-linearities and uncertainties more effectively as well as conform to novel objectives other than tracking or stability. The switching between these NNDM controllers holds a lot of potential for expressing complex behavior. Furthermore, formal methods can provide formal safety and performance guarantees across an entire finite state-space. This contrasts with standard control theory tools like LQR, which are typically constrained to offering guarantees within a small area around equilibrium points.

Hence, we aim to design a switched controller for a modeled non-linear system performing complex behaviors expressed using linear temporal logic on finite traces (LTLf). To achieve this, we apply machine learning techniques, in the form of neural network dynamic models (NNDM), to design a set of models that describe the closed-loop behavior of an inverted pendulum on a cart under a set of different stabilizing controllers that perform additional tasks, like guiding the cart along the rail. We then use abstraction methods to combine these controllers and design a switched controller that performs complex behaviors. By the use of formal abstraction methods, we obtain guarantees of the designed switched controller w.r.t. our model. We first focus on Neural Networks to learn the system and design controllers.

In summary, we define the following objectives:

- Use Neural Network Dynamical models to learn the behavior of a closed-loop system under a set of different stabilizing controllers performing simple behavior.
- Apply abstraction methods to switched Neural Network Dynamical Systems for LTLf specifications.
- Validate the formal framework through experiments, with an emphasis on showcasing the performance, formal guarantees and intricacies with respect to the latent system.

Throughout this Thesis we refer to the steps taken in this approach, to go from a switched stochastic controller to applying formal methods, as the framework.

1.3. Contribution

This thesis presents the following contributions:

- We demonstrate the full process of modelling and applying formal methods to an inverted pendulum on a cart system.

- We provide a detailed comparison about different neural network architectures for the modeling of dynamic systems.
- We demonstrate the efficacy of the framework across several experiments featuring complex specifications articulated using LTLf semantics.
- We prove that the satisfaction guarantees obtained for the abstractions hold for the neural network dynamic models.
- We demonstrate in what way the formal guarantees on abstraction and the neural network dynamic models apply to the latent system.
- We analyze the advantages and limitations of applying this framework to real dynamic systems, discussing its readiness for practical deployment and identifying areas that require further development.

1.4. Outline

This thesis document is organized to methodically explore the synthesis of control strategies for dynamic stochastic systems and the application of neural network models for system dynamics and control synthesis. Initially, in Chapter 2, we delve into the formal abstraction of dynamic stochastic systems. We introduce foundational concepts such as Interval Markov Decision Processes (IMDPs) in Section 2.1.1 and elaborate on calculating IMDP transition probability bounds in Section 2.1.2. Further, in Section 2.2, we discuss formal control synthesis over LTLf specifications, detailing complex specifications using LTLf in Section 2.2.1 and the control synthesis process in Section 2.2.2.

In Chapter 3, we outline our methodological framework, starting with the problem statement in Section 3.1. We describe the system model, an inverted pendulum on a cart, in Section 3.2, focusing on its dynamics and simulation for data acquisition in Sections 3.2.1 and 3.2.2, respectively. Section 3.3 discusses neural network dynamic models, detailing modes and network architecture in Sections 3.3.1 and 3.3.2. Furthermore, we address formal verification and control synthesis in Section 3.4, including abstraction and synthesis techniques, approximating probabilities of satisfying specifications, and determining standard deviations for model noise in Sections 3.4.2 to 3.4.3.

Chapter 4 presents our experimental results, where we first evaluate neural network dynamic modeling outcomes, including a post-training analysis and discussions on implications in Sections 4.1.1 and 4.1.2. We then assess our experiments on formal verification and control synthesis in Sections 4.2 and 4.3, detailing experiment conditions, evaluating the abstraction and classification, and analyzing the synthesized policy. This chapter emphasizes our analytical process and findings, drawing conclusions from the results and its impact on our research objectives.

Finally, in Chapter 5, we consolidate our findings, summarizing the significant contributions of our research to the field of dynamic stochastic systems and control synthesis. We reflect on the implications of our work, its limitations, and propose directions for future research that could build upon our findings.

Throughout this document, we aim to present a coherent and comprehensive exploration of our research topic, supported by rigorous methodological approaches, experimental validation, and a critical analysis of our results.

2 Background

This chapter briefly discusses relevant background topics, divided into two main sections. The first section focuses on the formal abstraction of dynamic stochastic systems, as detailed in Section 2.1. The second section addresses formal control synthesis over LTLf specifications, presented in Section 2.2. Within these sections, we will cover some important concepts necessary for abstraction, such as Interval Markov Decision Processes (IMDPs) in Section 2.1.1 and the calculation of their transition probability bounds in Section 2.1.2. Additionally, we provide an introduction to temporal logic, with a focus on LTLf in Section 2.2.1, and to control synthesis in Section 2.2.2.

2.1. Formal Abstraction of Dynamic Stochastic Systems

In this section, we will discuss the methods required to abstract continuous-domain, discrete-time stochastic systems into switched (stochastic) models suitable for transition-based formal control synthesis.

2.1.1. Interval Markov Decision Processes

Markov Decision Processes (MDPs) (Bellman, 1957) are widely used in the field of artificial intelligence and computer science for modeling complex decision-making problems due to their simplicity, scalability and model flexibility. MDPs provide a mathematical framework for modeling sequential decision-making under uncertainty. MDPs are formally defined as in Definition 1.

Definition 1 (MDP)**.** A Markov Decision Process (MDP) is a tuple $\mathcal{I} = (Q, A, P, \Pi, L)$, where

- Q is a finite set of states;
- A is a finite set of actions available in each state $q \in Q$;
- $P : Q \times A \times Q \to [0, 1]$ is a function, where $P(q, a, q')$ defines the transition probability from state $q \in Q$ to state $q' \in Q$ under action $a \in A$;
- Π is a finite set of atomic propositions;
- $L : Q \to 2^{\Pi}$ is a labeling function assigning to each state $q \in Q$ a subset of Π;

For all $q, q' \in Q$ and $a \in A$, it holds that $\sum_{q' \in Q} P(q, a, q') = 1$.

However, the applicability of Markov Decision Processes (MDPs) as an abstraction model for dynamic stochastic systems is limited due to several factors. These factors include the challenge of accurately representing the system dynamics using exact transition probabilities. Additionally, the model size often grows exponentially compared to the underlying model, leading to a phenomenon known as state explosion. Moreover, MDPs are unable to effectively model systems that do not adhere to the Markov property (as defined in Definition 2).

Definition 2 (Markov property)**.** A state q_t is Markov if and only if $P(q_{t+1} \mid q_t) = P(q_{t+1} \mid q_1, \cdots, q_t)$, meaning that the next state q_{t+1} is only dependent on the current state q_t and not on previous states.

To overcome the shortcomings of having exact transition probability values, derivatives of the MDP that include uncertainty in the parameters have been developed called uncertain MDPs which allows more general sets of distributions to be associated with each transition. This thesis will focus on the uncertain MDP called Interval Markov Decision Process (IMDP) (Hahn et al., 2017), first introduced as Bounded Markov Decision Processes (Givan et al., 2000). It introduces upper and lower bounds on the transition probabilities, which makes it a set containing an infinite number of MDPs with the same state and action space. IMDPs are formally defined as in Definition 3.

Definition 3 (IMDP)**.** An interval Markov decision process (IMDP) is a tuple $\mathcal{I} = (Q, A, \check{P}, \hat{P}, \Pi, L)$, where

- Q is a finite set of states;
- A is a finite set of actions available in each state $q \in Q$;
- $\check{P} : Q \times A \times Q \to [0, 1]$ is a function, where $\check{P}(q, a, q')$ defines the lower bound of the transition probability from state $q \in Q$ to state $q' \in Q$ under action $a \in A$;
- $\hat{P} : Q \times A \times Q \to [0, 1]$ is a function, where $\hat{P}(q, a, q')$ defines the upper bound of the transition probability from state $q \in Q$ to state $q' \in Q$ under action $a \in A$;
- Π is a finite set of atomic propositions;
- $L : Q \to 2^{\Pi}$ is a labeling function assigning to each state $q \in Q$ a subset of Π;

For all $q, q' \in Q$ and $a \in A$, it holds that $\check{P}(q, a, q') \leq \hat{P}(q, a, q')$, $\sum_{q' \in Q} \check{P}(q, a, q') \leq 1$ and $\sum_{q' \in Q} \hat{P}(q, a, q') \geq 1$.

IMDPs have the ability to reduces the state explosion problem, as the states of a IMDP represent sets (aggregates) of more primitive states that are grouped together. These primitive states represent, in the case of dynamic systems, a set of continuous-domain states. The interval transition probabilities represent the ranges of the parameters over the primitive states belonging to the aggregates. IMDPs are viewed as the appropriate abstraction model for uncertain systems with large state spaces as the state-space aggregation reduces the amount of states and consequently computation complexity. Additionally, the true distribution of the continuous-domain stochastic system from any initial state is contained within the corresponding IMDP transition probability intervals. Meaning that the results of the formal methods also apply to the underlying stochastic system, as proven by (Adams et al., 2022).

2.1.2. Calculating IMDP Transition Probability Bounds

Abstracting a continuous-domain, discrete-time, switched stochastic model to an IMDP requires to define the states of the abstraction model and determining the transition probability bounds, as given in definition 3 between these states for every available action. The states are created by discretizing the continuous state-space to a set of polytopic, non-overlapping subspaces/regions, with respect to both obstacles and proposition regions, whose union is the continuous-domain state-space.

A desirable discretization depends on a number of factors, including the geometry (obstacles and other regions of interest), dynamics of the system and computational simplicity. The coarseness of the discretization has a direct impact on the computation time of the subsequent formal verification and synthesis. The discretization can be defined by numerous methods. Where the uniform grid is most often used due to its simplicity (also in higher dimensions) and efficient computation of the transition probabilities (Laurenti et al., 2021). Non-uniform grids can be used to allow for a finer discretization around borders of safe sets and obstacles improving the bounds in safety-critical states of the model, while still having the same computational advantage as the uniform grid.

For a certain control policy, a stochastic dynamic model defines a probability measure P which is uniquely determined by the transition kernel and the initial conditions (Bertsekas & Shreve, 2004). In the context of this discussion, it's pivotal to first introduce the concept of a non-linear stochastic system characterized by additive noise. Such a system can be represented as:

$$x_{k+1} = f_a(x_k) + vw_k, \tag{2.1}$$

where $x_k \in \mathbb{R}^n$ denotes the state at time k, $f_a(\cdot)$ is a non-linear function representing the system's dynamics under control action a, and v_k represents the process noise at time k, which is often assumed to be Gaussian with zero mean and covariance matrix Cov_w.

Given $a \in A$, where A is the set of all possible actions, and $X \subseteq \mathbb{R}^n$, the transition kernel $T^a(X|x)$ for a state $x \in \mathbb{R}^n$ and a subset of states X can be defined as follows:

$$T^a(X|x) = \int_X \mathcal{G}(\bar{x}|f_a^w(x), Cov_v)d\bar{x}, \tag{2.2}$$

where $\mathcal{G}(\bar{x}|f_a^w(x), Cov_v)$ denotes the probability density function of the Gaussian distribution centered at $f_a^w(x)$ with covariance Cov_v, representing the uncertainty in the transition from state x to state $\bar{x}$ under action a. This definition encapsulates how the system evolves over time, taking into account the non-linearity of the system's dynamics and the stochastic nature of its evolution.

The transition probability bounds of the IMDP are bounded by T^a as follows

$$\check{P}(q, a, q') \leq \min_{x \in q} T^a(q'|x), \quad \hat{P}(q, a, q') \geq \max_{x \in q} T^a(q'|x). \tag{2.3}$$

Where the probability interval for transitioning to a region $q_u \in Q$ outside the safe set of underlying system states X_{safe} is given by

$$\check{P}(q, a, q_u) \leq 1 - \max_{x \in q} T^a(q_u|x) \tag{2.4}$$

$$\hat{P}(q, a, q_u) \geq 1 - \min_{x \in q} T^a(q_u|x) \tag{2.5}$$

for all $a \in A$ and $q \in Q$. Transitioning between regions outside X_{safe} are made to be absorbing as follows

$$\check{P}(q_u, a, q_u) = \hat{P}(q_u, a, q_u) = 1 \tag{2.6}$$

for all $a \in A$, as there is no interest in the behavior of the system in that area.

To compute the transition probability bounds of the IMDP abstraction model for a Neural Network Dynamic model as the underlying model, bounds need to be identified on the output of the NNDM for every state of the abstraction model.

A recent convex relaxation algorithm called CROWN (Zhang et al., 2018), can be used to build piece-wise linear functions that under- and overapproximate the NNDM's post-image for all its states contained in a single IMDP state. Finding these linear functions for all IMDP states allows to compute the transition probabilities by solving a set of convex optimization problems, as shown in (Adams et al., 2022). By overapproximating this convex bounded output by an axis-aligned hyper-rectangle, the problem can be reduced to an evaluation of an analytical function on a finite set of points, resulting in a more efficient abstraction procedure at the cost of more conservative bounds.

2.2. Formal Control Synthesis Over LTLf Specifications

In this chapter, the final step to formal verification and synthesis on the machine-learned model of chapter **??** is to apply the IMDP analogs of the MDP methods designed for abstraction obtained in chapter **??**. Starting with discussing the literature of temporal logic (TL) languages and showing how the two most used variants of TL are used for defining complex specifications to reason over the system. And finally verifications and synthesis of these specifications will be discussed with respect to IMDP. Reviewing IMDP changes how the standard formal methods have to approach this abstraction model.

2.2.1. Complex Specifications Using Ltlf

Temporal logic (TL) is a formal system for specifying and reasoning about a system's dynamic properties. It consists of rules and symbolism for representing and reasoning about propositions qualified in terms of time. It is used prominently in computer science and artificial intelligence for the analysis, verification, and policy synthesis of deterministic and stochastic transition systems that perform non-terminating (infinite) computations. These systems are ubiquitous in modern applications like cyber-physical systems and autonomous robots. TL is able to capture three key properties of infinite computations, namely liveness properties or eventualities, safety or invariance properties and fairness properties.

Liveness properties or eventualities given that a specific precondition is initially satisfied, then a desirable state satisfying the specification will eventually be reached in the course of the computation.

safety or invariance properties given that a specific precondition is initially satisfied, then undesirable states violating the safety specification will never occur.

Fairness properties concurrently run processes or objectives must be treated equally

The TL subclasses linear temporal logic (LTL) (Baier & Katoen, 2008) is one of the most widely used logic languages for formally representing these infinite computation systems. The syntax and semantics of LTL is defined in its cited source.

Complex temporal logic specifications for dynamic (stochastic) systems are often reasoned over a discretized state-space labeled with regions of interest, e.g., goals and obstacles. An example is given in figure 2.1, which will be used for upcoming example specifications used in this section.

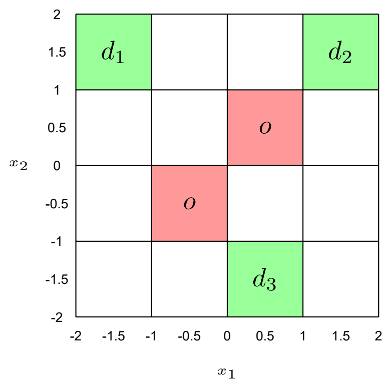


Figure 2.1: Grid based discretization of an arbitrary 2D state space, with destination states d and obstacles states o

Linear temporal logic (LTL) is a popular variant of temporal logic given its rich expressivity and intuitive formalism. It models time as a sequence of time points that extends infinitely into the future. LTL is often not suitable for reasoning about events with durations, they are better modeled if the underlying temporal ontology uses time intervals, i.e., periods rather than instants, as the primitive entities. LTL interpreted over finite traces (LTLf) (Giacomo & Vardi, n.d.) has the same syntax as LTL, but its semantics are defined over finite traces, making them suitable for specifications with finite durations. These finite durations are expressed in k as the number of transition steps that can be taken in the transition model.

Two examples of common LTLf specifications, in the context of the discretization shown in Figure 2.1, are provided below:

$$\mathcal{G}(\neg o) \wedge \mathcal{F}_{\leq k}(d_1) \tag{2.7}$$

which reads as “Globally avoid obstacles o and eventually reach desired/goal region d_1” and

$$\mathcal{G}(\neg o) \wedge (\mathcal{F}_{\leq k}(d_1) \vee \mathcal{F}_{\leq k}(d_2)) \tag{2.8}$$

reading as “Globally avoid obstacles o and eventually reach desired region d_1 or d_2”. Both are a type of specification known as reach-avoid.

The addition of finite traces in LTLf is practical, reflecting the time-limited nature of real-world operations due to task requirements or computational constraints. This makes LTLf appropriate for scenarios where outcomes need to be assured within a specific timeframe, such as automated control systems, by focusing on specifications that must be met within a finite number of steps.

2.2.2. Control Synthesis

Control synthesis is to determine, given an IMDP and a TL specification, a control policy for every IMDP state that maximizes the most pessimistic probability of satisfying that TL specification. IMDP introduces analogs of the standard MDP algorithms for computing the value function for a fixed policy and for computing optimal value functions over all policies, called interval policy evaluation and interval value iteration respectively (Givan et al., 2000). Control synthesis of an IMDP is generally a 2-player stochastic game. Where Player 1 chooses an action $a \in A$ at state $q \in Q$, and Player 2 chooses a feasible transition probability distribution. This game is adversarial, where the objectives of Players 1 and 2 are to maximize and minimize the probability of remaining in the safe set, respectively. Hence, the goal becomes to synthesize a strategy for Player 1 that is robust against all adversaries, as shown in (Laurenti et al., 2021).

After applying a policy to the IMDP, it becomes an IMC on which we can perform Formal verification. Formal verification is the process of checking whether a system satisfies a certain formal specification or property. The formal guarantees on the abstraction model that are the result of the verification process should, ideally, apply to the underlying system. Interval Markov models contain the true distribution of the underlying stochastic system from any initial state within its corresponding interval. If a specification is then satisfied by the distribution bounds of the interval Markov model, it is also satisfied by the distribution of the underlying system. Verification of an IMDP often requires it to be reduced to an IMC by calculating and applying the optimal control policy.

Verification on IMCs is mainly done by performing model checking, which can be defined as given an IMC and a TL specification (ϕ), find the sets of states that definitely (denoted as Q_{yes}), possibly ($Q_?$) and never (Q_{no}) satisfy ϕ. Where a region is labeled as Q_{yes} when the lower bound of the probability of satisfaction (P_L) is greater or equal to 0.95, Q_{no} when the upper bound of the probability of satisfaction (P_U) is less than 0.95 and $Q_?$ when $P_L < 0.95$ and $P_U \geq 0.95$. The model checking problem can be solved by finding the minimizing and maximizing adversaries (Lahijanian et al., 2015), which can be obtained iteratively through an ordering of the states called O-maximizing (Givan et al., 2000).

However, Model checking doesn’t scale well to large systems, a suitable alternative to model checking is the so-called product IMDP (PIMDP), which is the Cartesian product of an IMDP ($\mathcal{I}$) and a TL specification (ϕ) that is translated into an equivalent DFA ($\mathcal{A}_\phi$), and is defined as $\mathcal{P} = \mathcal{I} \times \mathcal{A}_\phi$. Conceptually, $\mathcal{P}$ is also an IMDP where each state is a unique tuple (q, z), where q is a discrete region of the state space and z is a state in $\mathcal{A}_\phi$. Interval Value Iteration (Givan et al., 2000) can then be used to perform the verification of $\mathcal{P}$. Reaching an accepting state in the product IMDP means the system has satisfied the specification and like wise reaching the sink state means that our system is violating the mission specification.

3 Method

This chapter outlines our methodology for formally controlling an inverted pendulum on a cart system. We begin with our problem statement in Section 3.1, defining the scope of our investigation. In Section 3.2, we discuss the system's dynamics, our approach to simulation and data acquisition, and offer a critical discussion of these elements. Section 3.3 focuses on the creation and analysis of Neural Network Dynamic Models, detailing the operational modes, network architecture, and concludes with a discussion. Finally, Section 3.4 delves into formal verification and control synthesis, including methods for approximating the probability of specification satisfaction, determining standard deviation for model noise, and abstracting to a stochastic model, followed by a summarizing discussion.

3.1. Problem Statement

We consider a Continuous-domain, discrete-time, switched stochastic system framework, as described in Equation 3.1. This approach falls within the realm of stochastic hybrid systems (SHSs), which are adept at capturing the complexity of physical systems through the interplay of multiple stochastic differential equations (SDEs). This setup not only accommodates the inherent uncertainties in system dynamics but also allows for the application of control strategies (Yin & Zhu, 2010)..

$$x_{k+1} = f_a^w(x_k) + v_k, \quad v_k \sim \mathcal{G}(0, Cov_v), \tag{3.1}$$

In the equation above, k represents the discrete time index, x_k and v_k are vectors in $\mathbb{R}^n$ representing the system state and noise at time k, respectively, and $a \in A = a_1, \cdots, a_m$ denotes a finite set of possible actions. Each action corresponds to a closed-loop mode defined by a neural network dynamic model, $f_a^w(x_k) : \mathbb{R}^n \to \mathbb{R}^n$ where w denotes the maximum likelihood weights, enabling the system to switch between different modes of operation. The noise term v_k follows a stationary Gaussian distribution with zero mean and a covariance matrix $Cov_v \in \mathbb{R}^{n \times n}$, introducing stochasticity into the system's evolution. This framework effectively models the system's discrete-time stochastic process, where the state transitions are guided by the predictions of various neural networks.

We recognize the challenges of defining an appropriate system and deriving neural network modes 3.1.0.1, creating an IMDP abstraction from the switched stochastic system 3.1 suitable for Markov-based formal methods 3.1.0.2, and applying formal methods to temporal logic specifications 3.1.0.3.

System and neural network modes

The first step in applying our framework is choosing a system that is both relevant to real-world, safety-critical applications and complex enough to demonstrate the framework's capabilities. This means selecting a system where combining simple behaviors can result in interesting and complex behaviors, and where these outcomes are not easily matched by simple conventional methods/controllers. We've chosen the inverted pendulum on a cart system for this purpose. The reasons for this choice, as well as details on its dynamics, simulation implementation, and how we gathered data, will be discussed in Section 3.2

With the system established, the next step is to define the simple operational behaviors, or modes, which will compose the complex behaviors. We aim to capture/learn these modes' closed-loop behavior through neural network dynamic models. with the objective to minimize the gap between the model's predictions and the true state values on an independent test dataset. Ensuring the model can generalize well on out-of-distribution data. In addition to exhibiting good performance in recursive simulation, with a low RMSE over time mean and variance.

We define these modes and elaborate on the methodology for obtaining the neural network dynamic models in Section 3.3.

Abstraction to IMDP

Building upon the stochastic switching model, our next objective is to abstract this model into a stochastic transition-based model suitable for transition-based formal methods. The abstraction model this methodology uses is an Interval MDP (IMDP), the background of which can be found in Subsection 2.1.1. This abstraction aims to accurately capture and reflect the dynamics of the original stochastic switching system. Ensuring that the model is suitable for applying formal verification and synthesis techniques, with guarantees that are applicable to the original stochastic switched system. This methodology of this process is discussed in Section 3.4.

Formal methods over LTLf

To define specifications on the system's behavior, we use LTLf, as detailed in Subsection 2.2.1. Then given a specification defined with LTLf and an IMDP, the next step involves applying formal methods designed for IMDPs, such as interval-iteration algorithms. the background on these algorithms, suited for our model's analysis and control policy synthesis, will be detailed in Subsection 2.2.

Next, it is crucial to validate and analyze the intermediate and final results of these formal methods, the methodology of which will also be discussed in Subsection 3.4.

3.2. System: Inverted Pendulum on a Cart

The methods and findings in this Thesis are with respect to a simulation based on a real-world Inverted Pendulum on a Cart or cart-pole system. As this system is a common benchmark for testing control strategies in the field of systems & control due to its nonlinearity, underactuated dynamics and inherently open-loop unstable equilibrium in the upwards direction of the pendulum, requiring constant feedback for stabilization.

The cart-pole system consists of a pole mounted on a cart in such a way that the pole can swing freely around its pivot point constrained to the vertical plane. To swing and to balance the pole, the cart is being moved on the horizontal plane on a rail of limited length. The objective is to steer the pole toward and maintain it in one of the equilibrium points despite random disturbances by carefully controlling the cart's movement. The focus will be on the unstable vertically upright pole position with arbitrary cart position equilibrium states. However, all findings will also apply to the same equilibrium points when the pendulum is in the stable downward position.

Real-world applications/similarities of this system can be found in the control of an aerodynamically unstable rocket, segway, self-balancing robots, a crane moving a load and in general underactuated systems.

This section will describe the system in detail, starting with presenting the dynamics of the system in Subsection 3.2.1. Followed by outlining and discussing the simulation of the ODE's and the data acquisition method that will be used in Subsection 3.2.2. At last, a discussion about the specific usage of the system and its implementation in this thesis will be held.

3.2.1. Dynamics

The dynamics of the inverted pendulum on a cart system are governed by Newton's laws of motion and rotational dynamics. The system's behavior is influenced by the gravitational force acting on the pendulum's weight and the inertial forces experienced by the cart and pendulum during motion. The system can be described by a forth order differential equation with a single input.

By basing the dynamics on a real-world counterpart system in the lab opens up the way for future work extending the results to the real-world system.

State Space and Action Space

The state of the system is a vector $x = [x_1, x_2, x_3, x_4]^T$ where x_1 is the position of the cart on the rail (m), x_2 is the angle between the upward direction and the pendulum (rad), measured counterclockwise ($x_2 = 0$ for the upright position of the pendulum), x_3 is the linear cart velocity (m/s), and x_4 is the pendulum angular velocity (rad/s). State x_1 is limited by the rail length, state x_2 takes values in the interval, $[-\pi, \pi]$ and their derivatives are only limited by their dynamics. To exert control on the system, a force F, parallel to the rail, is applied to the cart. In the real-world setup, this force is produced by a DC flat motor. This motor is controlled by a pulse width modulation (PWM) voltage signal u. The system control value u takes values in the interval $[-1, 1]$. It is linearly translated to a force by multiplication with the constant M.

A Diagram illustrating the system states and input force is shown in Figure 3.1.

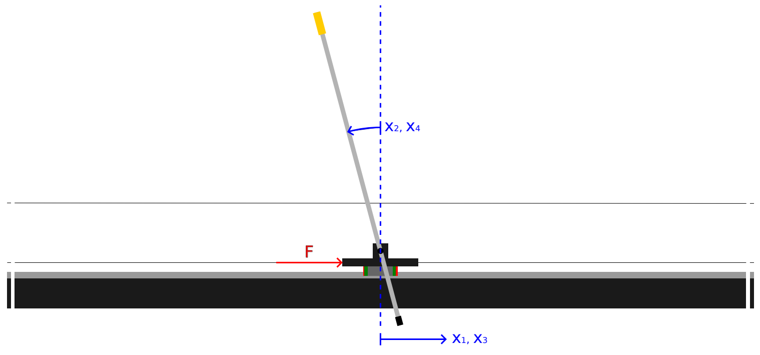


Figure 3.1: Diagram of the inverted pendulum on a cart, with states in blue and forces in red.

Equations of Motion (EoM)

The equation of motions is derived with respect to the pendulum being a compound pendulum, where the cart and pendulum both exert forces on each other. Furthermore, it features, viscous and coulomb friction on the cart and a viscous friction torque acting on the angular motion of the pendulum.

The equations of motion of the system are represented by

$$\dot{x} = \begin{bmatrix} x_3 \\ x_4 \\ \frac{l\,\mathsf{cos}(x_2)(m_i g\,\mathsf{sin}(x_2) - f_p x_4) + a(F_t(u,x_3) - m_i\,\mathsf{sin}(x_2) x_4^2}{J + m_i l\,\mathsf{sin}(x_2)^2} \\ \frac{m_i g\,\mathsf{sin}(x_2) - \frac{1}{2} m_i l sin(2x_2) x_4^2 - f_p x_4 + l\,\mathsf{cos}(x_2) F_t(u,x_3))}{J + m_i l\,\mathsf{sin}(x_2)^2} \end{bmatrix} \tag{3.2}$$

with the system's physical properties and the sub-equations used for simplifying the equations of motion are defined by:

$$J_p = \frac{1}{12}(m_{pw}l_c^2 + m_{ps}l_p^2) + \frac{1}{4}(m_{pw}r_c^2 + m_{pw}r_p^2) + (m_{pw}l_{co}^2 + m_{ps}l_{po}^2) \tag{3.3}$$

$$J = J_p - l^2(m_c + m_p), \text{ with} \tag{3.4}$$

$$l = \frac{l_{po}m_{ps} + l_{pwo}m_{pw}}{m_c + m_{ps} + m_{pw}} \tag{3.5}$$

$$m_i = l(m_c + m_p) \tag{3.6}$$

$$a = l^2 + \frac{J}{m_c + m_p} \tag{3.7}$$

$$F_t = Mu - sat_{-1}^{+1}(F_c sign(x_3) + F_s x_3). \tag{3.8}$$

Where, sat_{-1}^{+1} is the saturation function clamping its input between -1 and +1, and $sign$ is the sign function returning the sign of its argument. The used variables and their values are elaborated in Table 3.1

Table 3.1: ODE parameters

Parameter	Value	Unit	Description
m_c	0.5723	kg	Mass cart
m_p	0.12	kg	Mass pendulum
l	0.019557	m	Length
g	9.81	$/s^2$	Gravitational constant
J	0.0038583	kgm^2	Moment of inertia
f_p	0.00027344	Nms/rad	Pendulum friction
F_s	1.1975875	N	Static cart friction
F_c	0.5	N	Coulomb cart friction
M	12.86	N/pwm	Control magnitude

We have access to a perfect mathematical model of our system. However, for the following reason, it has been decided to model the full mode (model + controller) using a neural network instead of combining the mathematical model with a neural network controller. This approach facilitates extending the research to the physical world, where it is likely that the system will be modeled fully or partly by a neural network. This could involve using model-based reinforcement learning to simultaneously learn and utilize a model of the system for training a reinforcement controller, as suggested by (Adams et al., 2022). Another option is to use the mathematical model for known dynamics and a neural network for unknown dynamics from data, a method suitable for physical systems, explored by (Lahijanian et al., 2015) with Gaussian process regression.

3.2.2. Simulation & Data Acquisition

This subsection details the method of obtaining qualitative training, validation, and test datasets utilizing the ODEs introduced in section 3.2.1 as our data generating function. It details the data acquisition philosophy and pre-processing steps taken to ensure that the data is representative of data that can be encountered in the intended application. This aids the neural network model in learning the latent system dynamics efficiently.

Simulation

The system is simulated by solving the ODEs defined in Subsection 3.2.1 using the Explicit Runge-Kutta method of order 5(4).

The use of saturation and sign functions in the ODEs can result in the time-step, of variable time-step ODE solvers, becoming substantially small around the location of the discontinuities in these functions, which increases the computation time significantly. Therefore, continuous smooth approximations of these discontinuous functions have been used (Mahdi Ghazaei Ardakani & Magnusson, 2018; Shokouhi & Davaie Markazi, 2018).

Data Acquisition & Pre-Processing

To construct a dataset that is both sample efficient and representative, importance sampling is employed. This involves the simulation of trajectories of a predetermined size, initiating from states within a defined area of the operational state-space. Each trajectory, denoted as x, is decomposed into discrete pairs of inputs X and outputs Y. These pairs correspond to individual steps along the trajectory, formatted as:

$$[X : x(k), Y : x(k+1)]. \tag{3.9}$$

The trajectory's length is crucial as it influences the richness of the dataset in capturing the system's dynamics. The model is intended for use within a specific operational state-space, which has defined bounds. During simulations, many trajectories will extend beyond these bounds. However, data points falling outside the bounds of the operational state-space are not relevant to the model's application and can be discarded, allowing the dataset to be concentrated with pertinent information only.

It is essential to ensure that the process of discarding data does not eliminate valuable information. Thus, any single-step trajectory that begins within the bounds of the operational state-space is considered valid, even if it terminates outside these limits. These points are informative as they describe the system's behavior when it exits the operational state-space. The operational state-space is constrained by cart position and the pendulum angle, while their respective velocities are considered unbounded.

The data is then split into training (70%), validation (15%) and test (15%) subsets using random sampling to ensure equal data distribution between the subsets.

The data will be standardized to have zero mean and unit variance, which ensures balanced feature importance, faster convergence of optimization methods and numerical stability.

The mean and variance for the input and output data in the training set will be computed independently. These calculations will then be used to standardize the input and output data of the training set as well as both the validation and test sets.

3.3. Neural Network Dynamic Models

This section explores the process of acquiring models using neural networks through systematic examination of various neural network architectures in Subsection 3.3.2. Subsection 3.3.1, introduces the modes that the neural networks will train to mimic. A step-by-step approach is followed to evaluate the strengths and limitations of each architecture, the results of which are presented in Section 4.1. This iterative process refines the choices, leading to a better understanding of which configurations align most effectively with the objectives.

By the end of this chapter, readers will gain insights into the intricacies of learning models of closed-loop dynamical systems, the pros and cons of various neural network architectures, and the possibilities offered by incorporating external inputs.

3.3.1. Modes

In this subsection, we define three distinct operational states, referred to as modes, for the inverted pendulum on a cart system. Each mode is aimed at stabilizing the pendulum at its unstable equilibrium point, with a unique specific objective also pursued within each mode. These modes are detailed in Modes 0, 1, and 2.

For simplicity, the modes combine a mathematical model and a classic controller, with a neural network trained to replicate the closed-loop behavior of these modes. Future research could explore methods that learn to achieve specific objectives using neural networks directly, bypassing the need for pre-existing controllers, potentially allowing for more complex behavior expressed by each mode.

Mode 0 (Positive Force)**.** A mode that utilizes a stabilizing LQR Controller for pendulum stabilization, excluding cart position, with a constant additive PWM disturbance of 5.0. The objective is to drive the cart positively while ensuring pendulum stability.

Mode 1 (Zero Force)**.** A mode that utilizes a stabilizing LQR Controller for pendulum stabilization, excluding cart position, with zero input disturbance. The objective is to achieve stabilization of the pendulum and cart velocity at zero, aiming for system equilibrium independent of cart position across infinite potential equilibrium points.

Mode 2 (Negative Force)**.** A mode that utilizes a stabilizing LQR Controller for pendulum stabilization, excluding cart position, with a constant additive PWM disturbance of -5.0. The objective is to drive the cart negatively while ensuring pendulum stability.

All modes leverage a consistent LQR Controller designed from the system's linearized dynamics (see Subsection 3.2.1), with weight matrices $Q = diag(0, 1, 10, 1)$ and $R = 1$ respectively. By strategically combining these modes, we can freely maneuver or halt the cart along the rail in either direction, all while maintaining the pendulum's stability.

3.3.2. Network Architecture

In the Network Architecture section, we introduce three neural network models: Direct Time-Stepper, Residual Time-Stepper, and Physics-Informed (Residual) Time-Stepper. Each offers a different method for simulating dynamic systems using neural networks. We will evaluate these models against each other in the post-training analysis results, in Subsection 4.1.1, to see how they perform. Based on this comparison, we'll decide which architecture is best suited for each mode of the system we're studying. This approach allows us to methodically examine the advantages and limitations of each model, ensuring we choose the most effective one for our specific needs without overstating their capabilities.

Direct Time-Stepper

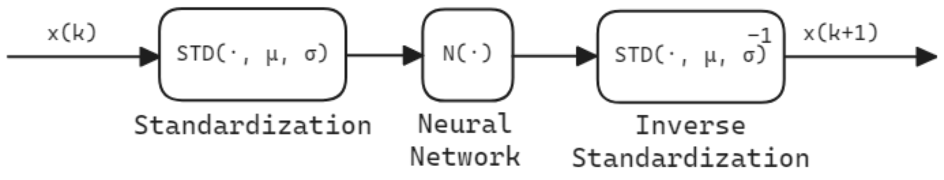


Figure 3.2: Diagram illustrating the Direct Neural Network Architecture. Including the data streams, standardization steps, Neural Network location and role.

$$x_{k+1} = \mathcal{N}_4^4(x_k) \tag{3.10}$$

Where $\mathcal{N}_4^4(x_k)$ is a neural network with input x_k and output x_{k+1} being the current state and next state respectively. Both have an equivalent size of 4 state values in each vector. During training and inference, both input and output data will be normalized.

The Direct Time-Stepper architecture represents the foundational approach in neural-network-based dynamic system simulation. It directly predicts the next state from the current state using the network's output. This method is initially chosen for its simplicity and the straightforward mapping it provides between consecutive states. From an initial state x_0, the model simulates the system's trajectory by successively using the network's output as the input for the next time step. While this process is trivial to implement, it inherently limits the model to accurately predicting only over short durations. As the simulation extends, the model's predictions tend to diverge from the actual system dynamics, underscoring its limited utility for long-term forecasting.

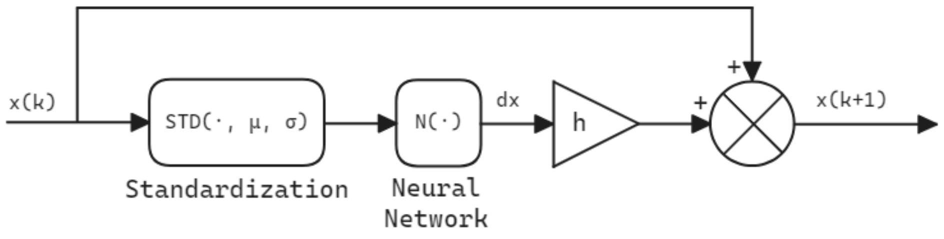


Figure 3.3: Diagram illustrating the Residual Neural Network Architecture. Including the data streams, standardization steps, Neural Network location and role, and the time-step h

Residual Time-Stepper

$$\dot{x} = \mathcal{N}(x) \tag{3.11}$$

$$x_{k+1} = x_k + h * \mathcal{N}(x_k) \tag{3.12}$$

Where h is the time-step encoded in the dataset, $\mathcal{N}(x_k)$ a neural network with the current state x_k as the input and the acceleration of the states as the output. During training and inference, normalization will be applied exclusively to the input data, as the neural network's output no longer directly corresponds to the dataset's output values.

Building upon the Direct Time-Stepper, the Residual Time-Stepper offers a refined approach by focusing on learning incremental changes in the system, akin to a discrete representation of the system's differential equations. By encoding the timestep h into the model, this architecture facilitates learning the derivative of the system's state over time. This approach is expected to enhance simulation performance, especially in terms of stability and the accuracy of predictions over longer periods. It addresses some of the direct method's limitations by potentially simplifying the learning process and achieving a closer approximation to the system's continuous dynamics. However, it requires re-initialization following discontinuities in the system, presenting a challenge for modeling systems with sudden changes.

Physics-Informed (Residual) Time-Stepper

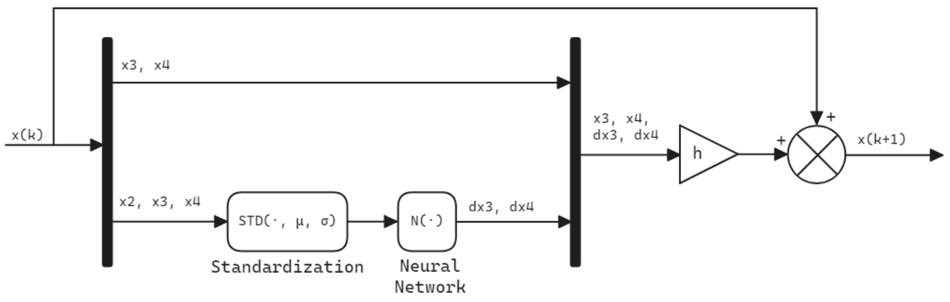


Figure 3.4: Diagram illustrating the Physics Informed Neural Network Architecture. Including the data streams, standardization steps, Neural Network location and role, and the time-step h

without prior knowledge of the ODE governing the inverted pendulum on a cart system, as described in Section 3.2, the following statements can be made.

1. The linear velocity of the cart is the direct derivative of its linear position.
2. The angular velocity of the pendulum is the direct derivative of its angular position.
3. The linear position of the cart has no influence on the linear and angular acceleration of the cart and pendulum respectively. This is due to the cart friction being constant across the entire length of the rail.

These physics-informed insights into the system's dynamics can be encoded into the Residual architecture, with the subsequent equation representing their implementation.

$$\dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_3 \\ x_4 \\ \mathcal{N}_2^3(x_2, x_3, x_4) \end{bmatrix} \tag{3.13}$$

$$x_{k+1} = x_k + h \begin{bmatrix} x_3 \\ x_4 \\ \mathcal{N}_2^3(x_2, x_3, x_4) \end{bmatrix} \tag{3.14}$$

Where h is the time-step encoded in the dataset, $\mathcal{N}(x_k)$ a neural network with the current state x_k as the input and the acceleration of the states as the output. During training and inference, normalization will be applied exclusively to the input data, as the neural network's output no longer directly corresponds to the dataset's output values.

The Physics-Informed (Residual) Time-Stepper architecture advances the concept further by incorporating specific domain knowledge into the modeling process. It leverages known physical relationships, for instance, the integrative nature of certain state variables, to ensure the model's predictions adhere to physical laws. This integration aims to overcome the inherent limitations of the previous models by ensuring accuracy and consistency across a wider range of inputs. It is expected to enhance the model's ability to generalize and ensure its predictions are physically plausible. Embedding domain knowledge directly into the model not only aims to improve accuracy but also narrows the search space for the optimization process to physically realistic solutions. Implementing this architecture requires a thorough understanding of the system's underlying physical principles, presenting both a challenge and an opportunity for enhancing model fidelity.

3.4. Formal Verification and Control Synthesis

In this section, we explore the application of formal methods to the neural network modes identified in the system. The process involves abstracting the neural network dynamics into Interval Markov Decision Processes (IMDPs), leveraging the CROWN algorithm for neural network relaxation, and conducting model checking for LTLf specifications. This methodology aims to establish formal guarantees on the neural network's behavior, ensuring its alignment with safety-critical requirements.

3.4.1. Abstraction and Synthesis

Abstraction of the stochastic switched NNDM 3.1 involves discretizing a subset of the state-space $\mathcal{X} \subset \mathbb{R}^n$ such that the transition kernel, shown in 2.2 can be computed analytically, allowing for the transition probability bounds to be calculated. Having obtained the abstraction states Q and the transition probability bounds $\check{P}$ and $\hat{P}$, Defining the remaining entries of the IMDP tuple (atomic propositions AP, labeling L) is trivial and completes the abstraction to the IMDP.

The abstraction methodology is based on (Adams et al., 2022), where it is concluded that an axis-aligned grid discretization allows for the efficient and scalable computation of the transition probability bounds. It allows the rewriting of the transition kernel as the product of Gause error functions. The probability transition bounds can then be calculated by optimizing this rewritten function over all points in the post-image, of each region corresponding to the individual states of the IMDP, under each action. where the post-image refers to the set of all points that can be reached from every point in a region under a certain action.

However, the exact computation of the post-image is intractable since neural-network dynamics are inherently nonconvex. This is solved by bounding the post-image by a convex hull through overapproximating the post-image. Achieved by recursively finding linear functions on the NN-structure that under- and overapproximate the neural network dynamics model's output for all initial states in a region corresponding to that IMDP state. It is shown that to construct the post-image overapproximation of the NN, only the vertices of each region have to be checked.

Additional optimization includes the further overapproximation of the convex hull by an axis-aligned hyperrectangle, reducing the number of function evaluations even further at the cost of more conservative bounds.

For synthesis the policy can be computed using known algorithms with a computational complexity polynomial in the number of states in the IMDP. The used algorithms can be found in (Lahijanian et al., 2015). where the objective is to synthesize a strategy that maximizes the probability of satisfying the LTLf specification. Applying to policy to the system and neural network dynamic model, requires a mapping between the current continuous-time state to the corresponding discrete regions. Then at each time-step the program needs to apply the policy assigned to the current discrete region for the current time-step.

In this Thesis we perform the under- and overapproximation of the NNDM's output by utilizing the CROWN-alpha algorithm implemented by the bound-propagation (Mathiesen, 2022) python package. Furthermore, the abstraction and synthesis are implemented by the formal_synthesis_NNDMs package (Adams, 2022)

3.4.2. Approximating the Probability of Satisfying Specifications

To estimate the likelihood that the system or neural network dynamic model satisfies a given LTLf specification, we employ a sampling-based simulation approach. This involves uniformly sampling a substantial number of initial states and observing how many trajectories, under the synthesized control policy, reach a desired region within k steps, as mandated by the LTLf specification.

Empirical evaluation suggests that sampling 2048 initial states yields a probability value converging closely to the actual probability, with additional simulations making negligible adjustments to this estimate.

Given the high computational demands of this process for each state, our analysis will primarily focus on a 2D slice of the abstraction that represents specific properties of interest in other dimensions— for example, states with identical velocity discretization ranges.

The objective is to verify the state classifications as determined by the framework, with a particular focus on comparing the neural network and latent system's classifications and probabilities. We explore the impact of the tunable noise term in the neural network dynamic model on the framework's classifications, seeking ways to align these closer to the latent system's approximated classifications.

States are categorized as Q^{yes}, Q^{no}, and $Q^{?}$, based on threshold probabilities: $p \geq 0.95$ for Q^{yes}, $p \leq 0.05$ for Q^{no}, and $0.05 < p < 0.95$ for $Q^{?}$.

3.4.3. Determine Standard Deviation for Zero-Mean Gaussian Model Noise

This subsection outlines the approach to determining an appropriate standard deviation for the zero-mean Gaussian noise term v_k in our stochastic switched model, as presented in Equation 3.1. The aim is to identify a minimal standard deviation that ensures the actual system states predominantly reside within the bounds set by the neural network dynamic model's predictions, adjusted for noise.

To achieve this, we initially calculate the standard deviation of the prediction errors from the neural network dynamic model (NNDM) across a test dataset. By adopting a threshold of three standard deviations, we align with the empirical rule, asserting a 99.7% confidence level that the actual system outputs will fall within this range from the NNDM predictions. Consequently, this method establishes a conservative estimate for the noise's standard deviation, enhancing the robustness of the model's predictions against the variability inherent in the system's behavior.

It's important to note the current framework utilizes a single, unified standard deviation for noise across all operational modes, due to limitations in the code implementation rather than the framework itself. Due to this limitation, a uniform standard deviation that uses, for each state, the maximum error standard deviation across all modes is selected. This approach increases the probability that formal guarantees on the neural network dynamic model are applicable to the actual system.

4 Experimental Results

In this chapter, the results of the neural network architectures are presented, discussed, and concluded in Section 4.1. Next, two experiments using formal verification on a single mode and control synthesis on multiple modes are conducted in Sections 4.2 and 4.3, respectively.

All the computations were performed on an AMD EPYC 7252 (8-core 16-threads) CPU @ 3.1GHz machine with 251.64 GiB of SSD RAM running Ubuntu 20.04.6 LTS (Focal Fossa). The abstraction and synthesis algorithms are implemented in Python 3.8.10.

4.1. Neural Network Dynamic Modeling Results

This section presents in Subsection 4.1.1 the post-training analysis results for the neural network architectures detailed in Subsection 3.3.2. Results are organized in tables for clarity. Discussion and conclusions about which neural network architecture suits each mode, as outlined in Subsection 3.3.1, will follow in Subsection 4.1.2.

4.1.1. Post-Training Analysis

For training, a single-step criterion is used where we minimize the MSE loss using the ADAM Optimizer with a learning rate of 1e-4 and a batch size of 128 data points. Early Stopping is used as a regularization technique which stops the training after 7 epochs without an improvement to the validation loss. Each model is trained, validated and tested on subsets of a dataset consisting of 1024 trajectories each with a maximum length of 128 time-steps, generated as described in Subsection 3.2.2. The models use a fully connected network consisting of 2 hidden layers with 128 neurons each. Each layer of the network applies a ReLU activation function. The number of inputs and outputs are determined by a combination of the number of states characterizing the system and the chosen architecture.

Performance evaluation for each neural network architecture utilizes two key metrics to ensure a comprehensive assessment. The Root Mean Square Error (RMSE) across all subsets of the dataset serves two primary functions. Firstly, it reveals the model's ability to accurately predict one time-step ahead. Secondly, it acts as an indicator of potential overfitting or underfitting, as well as the model's capacity for generalizing to unseen data. This metric essentially measures the model's accuracy in capturing the system's dynamics, providing insight into its predictive precision.

Furthermore, the RMSE over time across trajectories within the testing dataset, characterized by its mean and standard deviation, delves into the model's overall simulation performance. A lower mean RMSE over time indicates a closer alignment of the predicted trajectory with the true trajectory, highlighting the model's effectiveness in simulating system behavior over time. The standard deviation of the RMSE over time further illuminates the consistency of this performance across different trajectories. A lower standard deviation suggests a more reliable model, as it indicates less variation in simulation accuracy, thus providing a higher degree of confidence in the model's predictions across varying initial states.

Performance metrics for each architecture and system state across modes 0, 1, and 2 are detailed in Tables 4.1, 4.2, and 4.3, respectively. The optimal results in each metric are highlighted in bold, offering a direct comparison of architecture performance by mode and individual states.

Table 4.1: Performance Comparison of Neural Network Architectures for Mode 0: Stabilizing the Pendulum with Cart Movement in the Positive Direction.

		Quality of neural network measured in RMSE with respect to:			Mean (μ) and std. deviation (σ) of RMSE over time on trajectories with respect to:	
Neural network architecture	State	Training dataset $t = 0.1s$	Validation dataset $t = 0.1s$	Testing dataset $t = 0.1s$	Testing subset 1000 initial states $t = 1.0s$ (10 steps)	$t = 2.0s$ (20 steps)
Direct (3.3.2.1) ($\Delta t = 0.1$)	x	3.986099e-3	4.019103e-3	3.934208e-3	2.876397e-2 +- 5.858026e-2	1.214829e-1 +- 1.476545e-1
	θ	4.072951e-3	4.031150e-3	4.023779e-3	8.665715e-3 +- 1.619282e-2	**9.397477e-3 +- 1.226670e-2**
	$\dot{x}$	1.093870e-2	1.095420e-2	1.086443e-2	4.922664e-2 +- 8.929576e-2	6.288309e-2 +- 6.172371e-2
	$\dot{\theta}$	**2.082044e-2**	**2.102464e-2**	**2.098981e-2**	5.298706e-2 +- 9.498770e-2	**7.527850e-2 +- 6.982767e-2**
Residual (3.3.2.2) ($\Delta t = 0.1$)	x	**1.908012e-3**	**1.917313e-3**	**1.903257e-3**	**9.826561e-3 +- 1.499090e-2**	**1.651457e-2 +- 2.923048e-2**
	θ	**3.333516e-3**	**3.339943e-3**	**3.360645e-3**	**5.583763e-3 +- 9.455054e-3**	1.153697e-2 +- 1.275278e-2
	$\dot{x}$	**9.771481e-3**	**9.781092e-3**	**9.664158e-3**	**2.793586e-2 +- 6.401282e-2**	**2.289666e-2 +- 4.765756e-2**
	$\dot{\theta}$	2.125058e-2	2.121099e-2	2.104416e-2	**4.529470e-2 +- 1.048722e-1**	7.563911e-2 +- 9.221555e-2
Physics-Informed (3.3.2.3) ($\Delta t = 0.1$)	x	7.979177e-2	7.988800e-2	7.975314e-2	1.901380e-1 +- 2.123003e-1	2.455700e+0 +- 9.139306e+0
	θ	2.080068e-1	2.078730e-1	2.083933e-1	3.145409e-1 +- 6.319578e-1	1.748964e+1 +- 7.837483e+1
	$\dot{x}$	1.060047e-2	1.044633e-2	1.036507e-2	7.626831e-1 +- 1.037656e+0	1.453342e+1 +- 6.033907e+1
	$\dot{\theta}$	2.133551e-2	2.120035e-2	2.113070e-2	1.768638e+0 +- 3.953156e+0	1.203529e+2 +- 5.473835e+2

Table 4.2: Performance Comparison of Neural Network Architectures for Mode 1: Stabilizing the Pendulum with the Cart Stationary.

		Quality of neural network measured in RMSE with respect to:			Mean (μ) and std. deviation (σ) of RMSE over time on trajectories with respect to:	
Neural network architecture	State	Training dataset $t = 0.1s$	Validation dataset $t = 0.1s$	Testing dataset $t = 0.1s$	Testing subset 1000 initial states $t = 1.0s$ (10 steps)	$t = 2.0s$ (20 steps)
Direct (3.3.2.1) ($\Delta t = 0.1$)	x	3.397756e-3	3.417063e-3	3.431818e-3	4.248796e-2 +- 9.068714e-2	8.427933e-2 +- 1.619413e-1
	θ	3.753193e-3	3.752171e-3	3.803171e-3	8.556233e-3 +- 1.818780e-2	8.802772e-3 +- 1.501295e-2
	$\dot{x}$	9.882907e-3	1.006878e-2	1.000421e-2	4.515327e-2 +- 1.206631e-1	4.038146e-2 +- 8.645238e-2
	$\dot{\theta}$	1.811757e-2	1.840521e-2	1.826410e-2	4.572835e-2 +- 1.231187e-1	3.588912e-2 +- 8.959243e-2
Residual (3.3.2.2) ($\Delta t = 0.1$)	x	**1.878728e-3**	**1.876222e-3**	**1.887725e-3**	**1.387559e-2 +- 1.349794e-2**	**2.236144e-2 +- 1.789932e-2**
	θ	**2.600980e-3**	**2.605263e-3**	**2.586454e-3**	**5.663732e-3 +- 7.149346e-3**	**4.641365e-3 +- 5.202798e-3**
	$\dot{x}$	**7.955784e-3**	**7.952134e-3**	**7.928675e-3**	**2.657601e-2 +- 3.735466e-2**	**2.369573e-2 +- 2.661489e-2**
	$\dot{\theta}$	**1.628350e-2**	**1.631256e-2**	**1.642616e-2**	**4.129912e-2 +- 5.916665e-2**	**3.121119e-2 +- 4.289140e-2**
Physics-Informed (3.3.2.3) ($\Delta t = 0.1$)	x	7.900462e-2	7.903646e-2	7.897978e-2	1.763930e-1 +- 1.823956e-1	1.730425e+0 +- 6.653227e+0
	θ	2.022699e-1	2.019445e-1	2.024932e-1	2.412917e-1 +- 3.719298e-1	4.568143e+0 +- 1.960642e+1
	$\dot{x}$	8.494009e-3	8.591868e-3	8.508120e-3	6.964760e-1 +- 8.956364e-1	9.007474e+0 +- 3.893109e+1
	$\dot{\theta}$	1.748817e-2	1.760462e-2	1.746311e-2	1.243155e+0 +- 1.841084e+0	2.507996e+1 +- 1.120814e+2

4.1.2. Discussion and Conclusion

The analysis of neural network architectures across different operational modes reveals the Residual architecture uniformly outperforms the others in every scenario with respect to the single-step RMSE, with the exception for the angular velocity of Mode 0. However, in this exception, the difference between the RMSE and the RMSE over time is minimal and has thus little impact on the final decision.

For Mode 2, the Direct architecture has a slight advantage over the Residual architecture in the RMSE over time at $t = 2.0s$. Suggesting its steady-state predictions are more accurately aligned with the true system behavior, resulting in a lower cumulative error over time. However, the standard deviation is in favor of the Residual architecture, providing more certainty over its long-term prediction.

For Mode 1, the Residual architecture outperforms the other architectures on all metrics.

Table 4.3: Performance Comparison of Neural Network Architectures for Mode 2: Stabilizing the Pendulum with Cart Movement in the Negative Direction

		Quality of neural network measured in RMSE with respect to:			Mean (μ) and std. deviation (σ) of RMSE over time on trajectories with respect to:	
Neural network architecture	State	Training dataset $t = 0.1s$	Validation dataset $t = 0.1s$	Testing dataset $t = 0.1s$	Testing subset 1000 initial states $t = 1.0s$ (10 steps)	$t = 2.0s$ (20 steps)
Direct (3.3.2.1) ($\Delta t = 0.1$)	x	3.364736e-3	3.364598e-3	3.390027e-3	4.768637e-2 +- 8.259894e-2	2.326112e-1 +- 2.691347e-1
	θ	4.023699e-3	4.055061e-3	4.094667e-3	1.224985e-2 +- 2.027513e-2	1.998634e-2 +- 1.966112e-2
	$\dot{x}$	9.983380e-3	1.001497e-2	1.013435e-2	4.802760e-2 +- 9.476592e-2	**5.210425e-2 +- 6.528978e-2**
	$\dot{\theta}$	1.971493e-2	1.980232e-2	1.987621e-2	6.345422e-2 +- 1.253589e-1	**5.964277e-2 +- 9.243456e-2**
Residual (3.3.2.2) ($\Delta t = 0.1$)	x	**1.949480e-3**	**1.953927e-3**	**1.983953e-3**	**1.569503e-2 +- 1.898898e-2**	**3.794937e-2 +- 2.503526e-2**
	θ	**2.698446e-3**	**2.729052e-3**	**2.718725e-3**	**8.226214e-3 +- 1.027502e-2**	**1.491621e-2 +- 8.390581e-3**
	$\dot{x}$	**8.841006e-3**	**8.891408e-3**	**8.960868e-3**	**4.392524e-2 +- 7.261709e-2**	5.939895e-2 +- 5.983781e-2
	$\dot{\theta}$	**1.855922e-2**	**1.849922e-2**	**1.870165e-2**	**5.947764e-2 +- 1.083669e-1**	7.477336e-2 +- 7.731328e-2
Physics-Informed (3.3.2.3) ($\Delta t = 0.1$)	x	7.982494e-2	7.982365e-2	7.966657e-2	1.987959e-1 +- 2.173938e-1	1.436071e+0 +- 4.731016e+0
	θ	2.081200e-1	2.078931e-1	2.078625e-1	2.589927e-1 +- 3.677476e-1	3.549475e+0 +- 1.572755e+1
	$\dot{x}$	1.012981e-2	1.011387e-2	1.015985e-2	8.186851e-1 +- 1.053637e+0	6.685267e+0 +- 2.550935e+1
	$\dot{\theta}$	2.233850e-2	2.206389e-2	2.211069e-2	1.295498e+0 +- 1.643243e+0	1.917195e+1 +- 8.959472e+1

The Physics-Informed architecture underperforms, which can be linked to the calculation method for position states, introducing a one-time-step lag. This most likely negatively impacts the learning of dynamics, when single time-step optimization is used. Conversely, the residual architecture, despite having a similar structure, successfully navigates around this issue. It compensates for the lag by allowing the velocities that determine the next position states to be learned rather than manually fixed, thereby allowing to compensate for the lag by adjusting these velocity predictions. Reducing the time-step and optimizing for multiple time-steps could lessen this lag's impact on learning the dynamics.

Figure 4.1 compares the performance of different architectures in recursive simulation. The physics-informed architecture results in noticeable lag in both the cart's position and the pendulum's velocity. Specifically, the cart position strays outside the expected range of $[-1, 1]$ meters, unlike the data used to train the model. The residual architecture demonstrates better generalization abilities compared to the others. While the direct architecture performs well within the dataset's limits, it is less effective at generalizing to data outside of those limits, unlike the residual architecture.

Based upon this discussion, the metrics provided in the tables, and the visualization, it is concluded that the Residual architecture will be utilized for all Modes 0,1,2.

4.2. Experiment: Formal Verification

This section outlines the process and findings from conducting formal verification on a single system mode. In Subsection 4.2.1, we define the conditions under which the experiments were conducted. Following this, Subsection 4.2.2 evaluates the abstraction. We then proceed to validate the state classification, as determined through formal verification. This involves empirically estimating the probability of meeting the specification for both the neural network dynamic model and the underlying system, discussed in Subsection 4.2.3.

4.2.1. Experiment conditions

For these experiments we use the Zero Force mode 1, which stabilizes the pendulum, on any part of the rail, without any external force applied to the cart, detailed in Subsection 3.3.1.

We consider the state-space to be the compact set $\mathcal{X} = [-1, 0] \times [-0.3, 0.66] \times [-3.6, 0.9] \times [-4.8, 1.2]$ for the cart position x_1 (m), pendulum angle x_2 (rad), cart velocity x_3 (m/s) and pendulum angular velocity x_4 (rad/s) respectively. A neural network dynamic model has been trained using the Residual architecture, as described in Section 3.3 and analyzed in Section 4.1. The neural network dynamic model is then abstracted into an IMC with a sampling time of $\Delta t = 0.1s$ and discretized $\mathcal{X}$ by a course grid such that each cell in the original space is $\Delta x = [0.0625, 0.06, 0.45, 0.6]$. This results in a discretization with

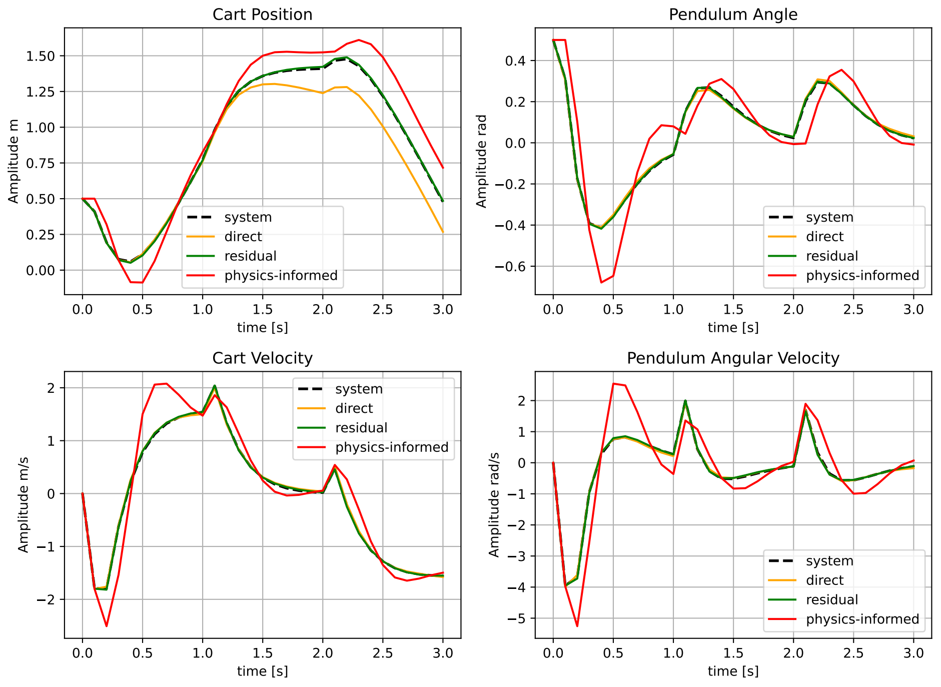


Figure 4.1: Comparison between different neural network architectures in terms of recursive simulation starting from the same initial state $x_0 = [0.5, 0.5, 0, 0]$ under the influence of the Modes 0, 1, 2 respectively for $k = 10$ each.

$N = [16, 16, 10, 10]$ cells in each dimension respectively, where each IMC abstraction Q has a cardinality $|Q|$ of 25600 states. An std of $[1.365532e-3, 2.225981e-3, 6.543285e-3, 1.433179e-2]$ is used to add stochasticity to the abstraction, as described in Subsection 3.4.3.

The mode will be formally validated against the following reach-avoid LTLf specification:

$$\phi = \mathcal{G}(\neg OB) \wedge \mathcal{F}_{\leq k}(D) \tag{4.1}$$

Reading as “Globally avoid the out-of-bounds (OB) area and eventually, within k steps, reach the desired region (D)”. Here, the out-of-bounds (OB) area refers to regions beyond the state-space limits, essentially representing real-world constraints such as the cart’s movement being physically restricted by the rail’s length or areas beyond our scope of interest. This specification allows validating when the pendulum can’t be stabilized without the cart exceeding the rail limit.

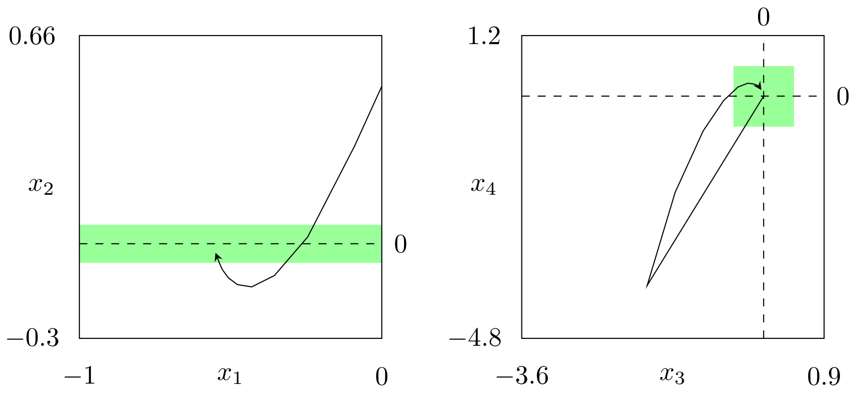
The LTLf specification and the system’s dynamics within the defined state-space are visually represented in Figure 4.2, showcasing the desired region “D” in green and a trajectory that exemplifies the system’s behavior under the Zero Force mode.

The state classification is determined with respect to $k = 20$ time-steps resulting with the abstraction time of $t = 0.1s$ in a simulation time of 2 seconds.

The choice of state-space was mainly motivated by available computational resources/time and computational complexity of the framework. Focusing on a subset of the latent system’s overall state-space enables a more detailed discretization with a manageable number of states than if the entire state-space were used. The subset encompasses the negative side of the rail and a narrow range of pendulum angles. Given the system’s symmetry, these observations are assumed to be applicable to the opposite side as well.

The ranges of the state-space and its discretization coarseness have been guided by observing simulated trajectories of both the neural network dynamic model and the latent system. Making sure that relevant trajectories remain within the state-space boundaries and that the discretization allows for the mode to exit its starting IMC state within a single time-step as much as possible.

Choosing the discretization coarseness is a tradeoff between the resolution of the formal verification and the quickly increasing computation time caused by the number of states $|Q|$ increasing proportionally to the product of the number of cells in each dimension.



(a) Cart Position vs. Pendulum Angle **(b)** Cart Velocity vs. Pendulum Angular Velocity

Figure 4.2: Visualization of State-Space, LTLf Specification's desired region indicated in green, and a model trajectory under the Zero Force Mode

4.2.2. Evaluating the Abstraction

In this subsection we will evaluate parts of the abstraction in order to better understand the subsequent results of formal verification.

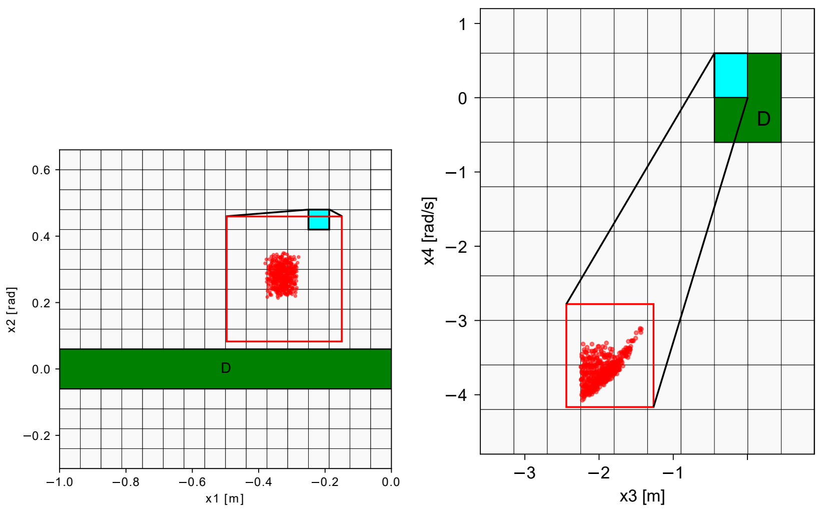
Due to the large number of abstraction states, it becomes computationally intractable to, for example, empirically approximate the transition probabilities for each and every state with a sufficient number of samples per state. For this reason, we opted to instead visualize the linear under- and overapproximation of the neural network mode by the CROWN algorithm given a region of the state-space as an input. Showcasing the tightness of the functions used to bound the post-image, as described in 3.4.1

Figure 4.3, illustrates the single-step prediction of the neural network from 512 uniformly sampled initial states in the input hyper-rectangle / region, as well as the under- and overapproximated output hyper-rectangle by the CROWN algorithm.

The output hyper-rectangle doesn't encapsulate the prediction of cart position and pendulum angle states tightly. This will have a negative impact on the tightness of the transition probability bounds, and down the line will result in a many lower bounds on probability of satisfying the specification with zero or near zero values. Conversely, the output hyper-rectangle on the predictions of cart velocity and pendulum angular velocity encapsulates the predictions more tightly. It is hence peculiar that it's so strikingly different from the cart position and pendulum angle output hyper-rectangle. Seeing these results, it is expected to see a lot of uncertainties in the subsequent state classification after performing formal verification.

4.2.3. Evaluating the Classification

This subsection evaluates the resulting state classification after applying the framework. For intuitive reasons as well to facilitate analysis and visualization, our focus is narrowed to a 2D slice consisting of the regions of the state-space with uniform velocities close to zero. Specifically, regions with lower and upper bounds of $[-0.45, 0]$ and $[0, 0.6]$ for the cart velocity (m/s) and pendulum angular velocity (rad) respectively are chosen. Starting in these regions will result in trajectories similar to the one illustrated in Figure 4.2.

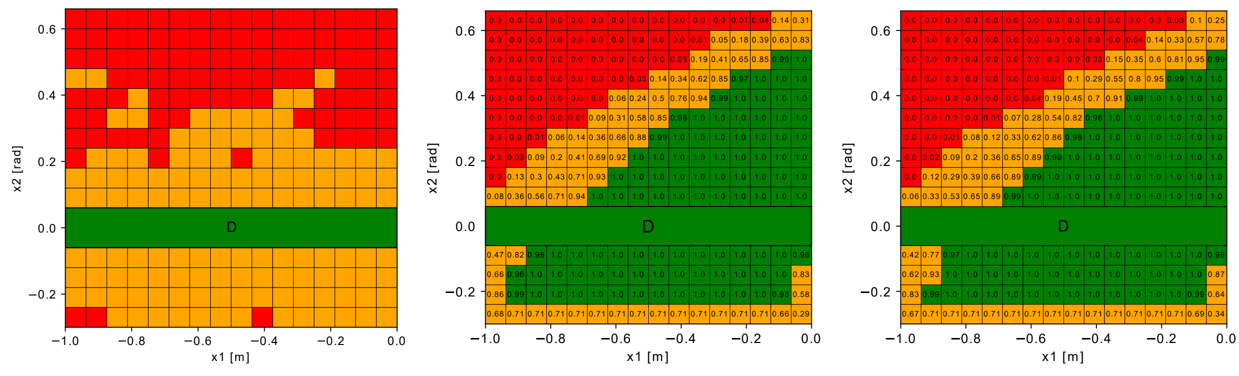


(a) Cart Position vs. Pendulum Angle **(b)** Cart Velocity vs. Pendulum Angular Velocity

Figure 4.3: Comparative visualization of neural network single-step predictions (red dots) and its under- and overapproximation determined by the CROWN Algorithm (red rectangle), from a uniformly sampled input region (turquoise rectangle)

For evaluating the classification result of the framework, a sampling-based approximation of the probability of satisfying the specification will be determined for each state in the slice, allowing to classify the states in an empirical manner. The methodology of this is discussed in Subsection 3.4.2. This approximation will be done for both the neural network model and the system, allowing to draw comparisons between the framework's classification and the neural network's approximated classification as well as comparisons between the neural network and system result. Giving us an insight into how the difference between the NN and system affects the approximated classification and how the classification results can be applied to the system.

Figure 4.4 illustrates, for the states in the 2D slice, the evaluation results starting with the state classification according to the framework, followed by the empirical state classification of the NN and the system respectively. Regions are classified based on their satisfaction probability, with $p \leq 0.05$ indicating Q^{no} (red), $p \geq 0.95$ signifying Q^{yes} (green), and intermediate probabilities marked as $Q^{?}$ (yellow).



(a) Formal state classification **(b)** NN emperical state classification **(c)** System emperical state classification

Figure 4.4: Approximation of satisfying the specification under the Zero Force mode for a 2D slice of the states-space, characterizing regions with uniform velocities in the range of $[-0.45, 0]$ m/s for cart velocity and $[0, 0.6]$ rad/s for pendulum angular velocity.

In Figure 4.4, it can be seen that the framework's classification has a lot of uncertainties as no states are labeled as Q^{yes}. States close to the destination are labeled, $Q^{?}$ although it has some outliers, while states with a higher starting pendulum angle are classified as Q^{no}. The result is expected, albeit with less uncertainty. The approximated probability of satisfaction for the neural network and system both show that the framework is over conservative. They showcase a very clear and intuitive pattern in the state classification where the specification is satisfied for increasingly lower angles, the more the starting position on the rail goes negative.

The state classification for the neural network is, with minor differences, equivalent to the system.

4.3. Experiment: Formal Control Synthesis

This section outlines the process and findings from conducting formal control synthesis using the available system modes. In Subsection 4.3.1, we define the conditions under which the experiments were conducted. Following this, Subsection 4.3.2 presents and interprets the state classification resulting from the synthesized control policy and formal verification. Lastly in Subsection 4.3.3, the synthesized policy is evaluated and trajectories are showcased.

4.3.1. Experiment conditions

For this experiment all three modes are utilized, namely the Positive Force 0, Zero Force 1 Negative Force 2 modes, The objective is to synthesize a policy that exhibits complex behavior while keeping the pendulum stable.

For the experiment, we consider the state-space to be the compact set $\mathcal{X} = [-1, 1] \times [-0.64, 0.64] \times [-3.5, 3.5] \times [-4.8, 4.8]$ for the cart position (m), pendulum angle (rad), cart velocity (m/s) and pendulum angular velocity (rad/s) respectively. Each neural network dynamic model has been trained using the Residual architecture, as described in Section 3.3 and analyzed in Section 4.1. The neural network dynamic models are then abstracted into an IMDP with a sampling time of $\Delta t = 0.1$ and discretized $\mathcal{X}$ by a course grid such that each cell in the original space is $\Delta x = [0.125, 0.08, 0.7, 0.96]$. This results in a discretization with $N = [16, 16, 10, 10]$ cells in each dimension respectively, where each abstraction Q to an IMCs has a cardinality $|Q|$ of 25600 states, combining the IMC into an IMDP results in 76800 states. An std of $[1.365532e^{-3}, 2.225981e^{-3}, 6.543285e^{-3}, 1.433179e^{-2}]$ is used to add stochasticity to the abstraction, as described in Subsection 3.4.3.

We consider applying formal control synthesis on a reach-avoid LTLf specification. Where reach-avoid implies that the goal is to reach one of the singular/multiple objectives while avoiding one or more obstacles. This specification is formally defined as:

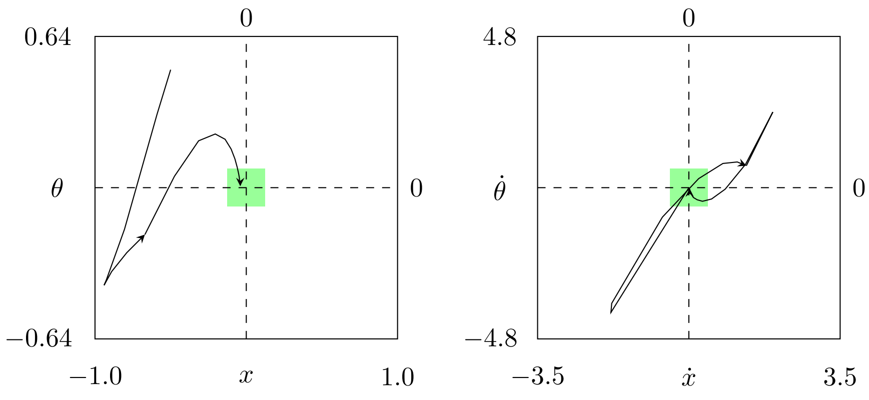
$$\phi = \mathcal{G}(\neg OB) \wedge \mathcal{F}_{\leq k}(D) \tag{4.2}$$

Reading as "Globally avoid the out-of-bounds (OB) area and eventually, within k steps, reach the desired region (D)". Here, the out-of-bounds (OB) area refers to regions beyond the state-space limits, essentially representing real-world constraints such as the cart's movement being physically restricted by the rail's length or areas beyond our scope of interest.

Using the specification and the definition of objectives and obstacles as visualized in Figure 4.5, we aim to synthesize a policy which utilizes a combination of modes to steer the cart and pendulum towards the origin in a stabilizing manner.

For intuitive reasons as well to facilitate analysis and visualization, our focus is again narrowed to a 2D slice consisting of the regions of the state-space with uniform velocities close to zero. Similar to what is done in the formal verification Subsection 4.2.3 Specifically, regions with lower and upper bounds of $[-0.7, 0]$, $[0, 0.96]$ for the cart velocity (m/s) and pendulum angular velocity (rad) respectively are chosen.

For evaluating the classification result of the formal control synthesis a sampling-based method is utilized. Where the approximation of the probability of satisfying the specification will be determined for each state in the slice, allowing to classify the states in an empirical manner. The methodology of this is discussed in Subsection 3.4.2. This approximation will be done for both the neural network model and the system, allowing to draw comparisons between the framework's classification and the neural network's approximated classification as well as comparisons between the neural network and system result. Giving us an insight into how the difference between the NN and system affects the approximated classification and how classification resulting from the control synthesis can be applied to the system.



(a) Cart Position vs. Pendulum Angle **(b)** Cart Velocity vs. Pendulum Angular Velocity

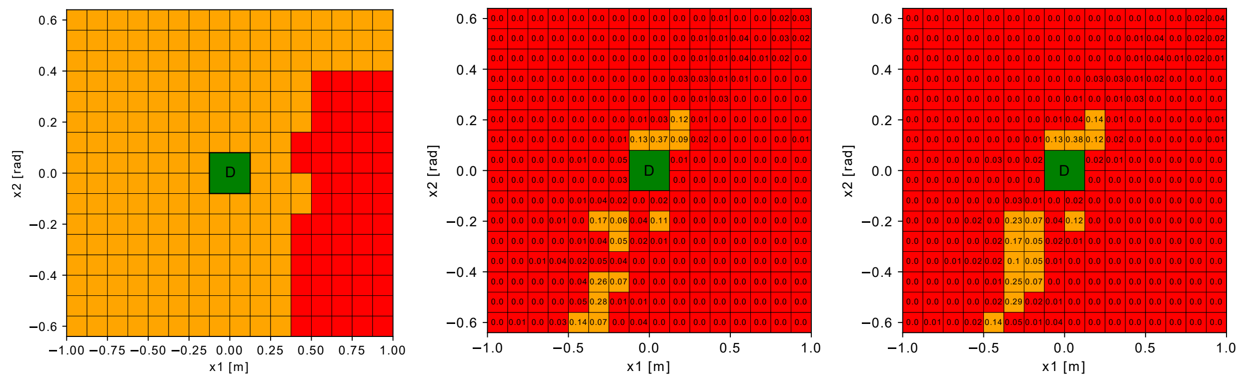
Figure 4.5: Visualization of State-Space, LTLf Specification's desired region indicated in green, and a model trajectory satisfying the specification for $k \leq 20$ time-steps

The abstraction took approximately 382 seconds for calculating the output hyper-rectangles for each region in the state-space and 1489, 1383 and 1348 seconds for computing the transition probability bounds for the positive, zero and negative force modes respectively. The computation time sums up to a total of 4602 seconds corresponding to 76.7 minutes. At last, performing formal control Synthesis on the specification took 748 seconds.

4.3.2. Evaluating the Classification

This subsection evaluates the resulting state classification after performing formal control synthesis on the LTLf specification 4.2

Figure 4.4 illustrates, for the states in the 2D slice, the evaluation results starting with the state classification according to the framework, followed by the empirical state classification of the NN and the system respectively. Regions are classified based on their satisfaction probability, with $p \leq 0.05$ indicating Q^{no}, $p \geq 0.95$ signifying Q^{yes}, and intermediate probabilities marked as $Q^{?}$.



(a) Formal state classification **(b)** NN emperical state classification **(c)** System emperical state classification

Figure 4.6: Approximation of satisfying the specification under the avaiable mode for a 2D slice of the states-space, characterizing regions with uniform velocities in the range of $[-0.7, 0]$ m/s for cart velocity and $[0, 0.96]$ rad/s for pendulum angular velocity.

It can be seen that the formal classification has a lot of uncertainties as again no states are labeled as Q^{yes}. There is a large section of states labeled as Q^{no} on the right end of the state-space, where over it can be reasoned that with the initial velocities of these regions, there is either not enough space in the state-space for any mode to steer it towards the center without getting out of bounds or there is not enough time to reach the origin. Given the speed at which the positive and negative modes move their cart, time should not be the limiting factor. When reviewing the approximated probability of satisfaction for both the neural network and the system, it is observed that many of the uncertainly labeled states collapse. Eventually, only a small group of states remains that are not labeled as Q^{no}, but remain uncertain with a rather low probability of satisfaction.

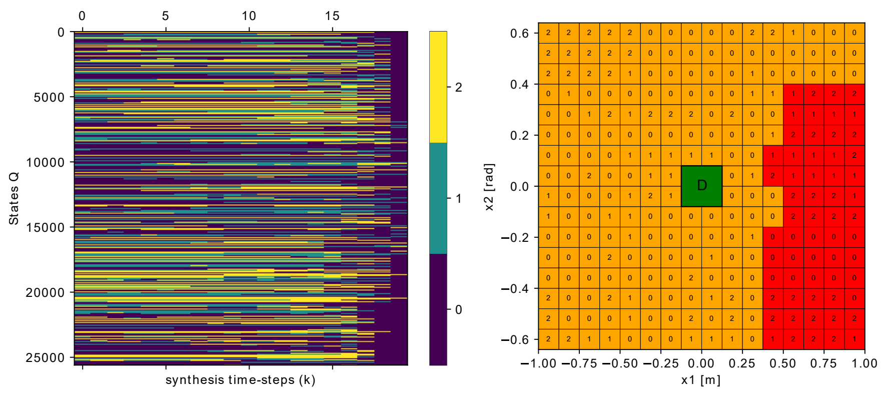
Reducing the additional force on the non-zero modes could reduce the space needed to make a maneuver towards a certain direction. As this first requires steering the pendulum angle towards that direction after which it can be pushed along at the desired speed. This would reduce the Q^{no} states protruding from the edges of the cart position and pendulum angle state-space ranges. It would also allow more time for the non-zero modes to reach their equilibrium velocity, which would be lower. This, in turn, would enable the zero mode to bring the pendulum to a stop using less space.

The state classification for the neural network is, with minor differences, equivalent to the system.

4.3.3. Evaluating the Synthesized Policy

This subsection evaluates the synthesized policy after performing formal control synthesis on the LTLf specification 4.2. We will look at the policy chosen for each state over time and showcase trajectories under the influence of this policy.

Figure 4.7 presents two complementary visualizations of the control policy. In Figure **??** we can see for each state the policy chosen for each time-step. This represents the look-up table to figure out which action to take given the current state and time-step. What is apparent is that as expected, each mode is being utilized, indicated by all colors representing the modes being present. One logical expectation would be that the non-zero modes (1, 2) would be over represented in the first few time-steps. These are required to steer the system towards the origin and are the logical choice for the majority of states that are not near the origin. Looking at Figure **??** we see that this expectation is not what is happening. In the mentioned figure the actions at $k = 0$ have been plotted onto the classification slice. The clear divide between the negative rail side using the positive mode 1 and the opposite site using the negative mode 2 can't be observed, there seems to be no clear pattern present.



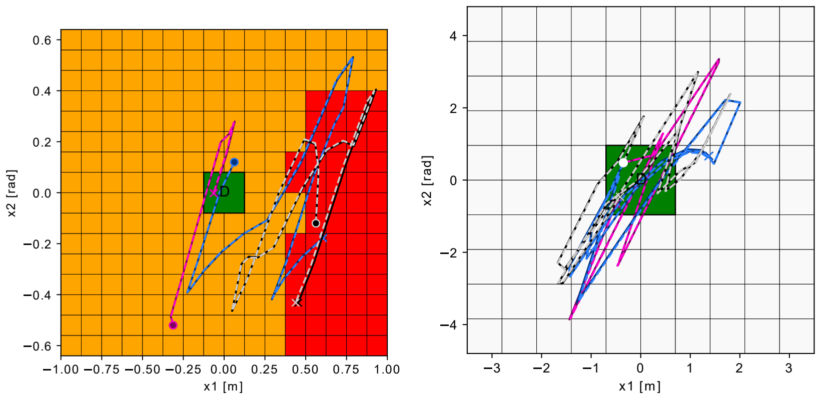
(a) Policy of states in Q with respect to k

(b) Policy at $k = 0$ overlayed on the classification slice

Figure 4.7: Complementary view of the synthesized policy, showcasing the lookup table in (a) and the action to take at $k = 0$ for the classification slice in (b)

This seemingly random assignment of actions in the synthesized policy is evident in Figure 4.8, where trajectories have been plotted under the influence of this policy. The initial points were chosen to start from the center of the states with the highest probability and one state where it starts in a Q^{no} region to observe what happens in those cases. We see that the trajectories are erratic due to the quick and frequent switches between the modes. None of the trajectories satisfied the specification, which was to be expected given the low probabilities of these states.

The failure of the synthesized policies to satisfy the specification with a higher probability stems from the numerous variables that must be carefully selected. The size of the regions and state space depends on many factors, including the dynamics of the modes and the trade-offs required due to available computer



(a) Cart Position vs. Pendulum Angle **(b)** Cart Velocity vs. Pendulum Angular Velocity

Figure 4.8: Visualization of three sets of NN and System trajectories under the influence of the synthesized policy. Where the dashed lines represents the system trajecotries and the full lines the NN trajecotries

.

power. Additionally, the crown algorithm needs to provide a tight enough under- and overapproximation to result in an abstraction with tight transition probability bounds.

5 Conclusion

In this thesis, we delved into the complex field of developing switched controllers for non-linear systems, using Neural Network Dynamical Models (NNDM) and formal methods to handle the inherent complexities of these systems. Our goal was to address the challenges posed by non-linear dynamics and uncertainties, by utilizing the capabilities of stochastic neural network models and the detailed analysis provided by formal abstraction methods.

Throughout our work, we aimed to create a switched controller framework with the expectation of overcoming some limitations found in traditional control theory tools, such as Linear Quadratic Regulators (LQR). Instead of claiming to have fully addressed these limitations, we recognize that our initial findings suggest the potential for such advancements, echoing previous literature. With further refinements, our framework may indeed provide a valuable approach to managing the complexity and uncertainty of non-linear systems, especially those modeled by neural network dynamic models. This endeavor led to a detailed examination using an inverted pendulum on a cart system, which offered a practical context for our experimental work and theoretical discussions.

Contributions and Insights:

We presented comprehensive contributions, including the successful modeling of dynamic systems using neural network architectures and the validation of our formal framework through rigorous experiments. Our contributions extend to providing formal satisfaction guarantees, analyzing the applicability of our framework to real dynamic systems, and discussing its practical deployment challenges.

> This insight hints at the potential for starting with a pre-calculated discretization to streamline the process.We advocate for the development of a more comprehensive guide to facilitate the use of our framework, acknowledging its current complexity and the steep learning curve for new users. Our experiments underscored the effectiveness of residual architectures in generalization and simulation, with a nuanced comparison between direct methods and physics-informed approaches, revealing their respective strengths and areas requiring further exploration. The selection of modes emerged as a critical factor, influencing the system's ability to satisfy specifications. This aspect underscores the delicate balance between control precision and the dynamical range of the system states.

Personal Reflections and Future Directions:

Reflecting on the challenges encountered, the implementation of the framework presented a computationally intensive endeavor, highlighting the need for optimization to enhance efficiency, as working with this formal framework takes up a lot of time and resources.

Learning the neural networks proved to be quite a difficult challenge, as the architecture, datasets and hyperparameters all played a key role and had to be finely tuned to achieve these results.

In light of our findings and the challenges faced, several areas for future work emerge:

- The creation of a more accessible guide to our framework could significantly lower the barrier to entry for future researchers. For example, what works and what does not, what are the pitfalls one should keep in mind while working with this framework.
- Further investigation into the impact of network architecture on the performance and generalization capabilities of NNDM could unveil new directions for model refinement.
- An in-depth exploration of mode selection criteria and its implications on system behavior could lead to more nuanced strategies for satisfying complex specifications.
- Addressing the computational demands of our framework remains a pressing concern, with potential solutions lying in algorithmic optimizations or hardware advancements.

In conclusion, while our approaches have demonstrated promising results in synthesizing strategies for dynamical systems under complex specifications, they also uncover a trade-off between conservatism and efficiency. Our contributions highlight the current state of research in the field and opens up avenues for further research and refinement of the method.

Bibliography

Adams, S. (2022). Bound propagation.

Adams, S., Lahijanian, M., & Laurenti, L. (2022). Formal control synthesis for stochastic neural network dynamic models. *IEEE Control Syst. Lett.*, *6*, 2858–2863. https://doi.org/10.1109/lcsys.2022.3178143

Anderson, B. G., Ma, Z., Li, J., & Sojoudi, S. (2020). Tightened Convex Relaxations for Neural Network Robustness Certification [Issn: 2576-2370]. *2020 59th IEEE Conference on Decision and Control (CDC)*, 2190–2197. https://doi.org/10.1109/cdc42340.2020.9303750

Baier, C., & Katoen, J.-P. (2008, January). *Principles of Model Checking* (Vol. 26202649).

Bellman, R. (1957). A Markovian Decision Process [Publisher: Indiana University Mathematics Department]. *Journal of Mathematics and Mechanics*, *6*(5), 679–684. Retrieved March 9, 2023, from https://www.jstor.org/stable/24900506

Bertsekas, D. P., & Shreve, S. (2004, March). *Stochastic optimal control: The discrete-time case*. Retrieved March 22, 2023, from https://dspace.mit.edu/handle/1721.1/4852

Giacomo, G. D., & Vardi, M. Y. (n.d.). Linear Temporal Logic and Linear Dynamic Logic on Finite Traces.

Givan, R., Leach, S., & Dean, T. (2000). Bounded-parameter markov decision processes. *Artif. Intell.*, *122*(1-2), 71–109. https://doi.org/10.1016/s0004-3702(00)00047-3

Hahn, E. M., Hashemi, V., Hermanns, H., Lahijanian, M., & Turrini, A. (2017, July). Multi-objective Robust Strategy Synthesis for Interval Markov Decision Processes [arXiv:1706.06875 [cs]]. Retrieved March 5, 2023, from http://arxiv.org/abs/1706.06875

Lahijanian, M., Andersson, S. B., & Belta, C. (2015). Formal verification and synthesis for discrete-time stochastic systems. *IEEE Trans. Automat. Contr.*, *60*(8), 2031–2045. https://doi.org/10.1109/tac.2015.2398883

Laurenti, L., Lahijanian, M., Abate, A., Cardelli, L., & Kwiatkowska, M. (2021). Formal and efficient synthesis for continuous-time linear stochastic hybrid processes. *IEEE Trans. Automat. Contr.*, *66*(1), 17–32.

Mahdi Ghazaei Ardakani, M., & Magnusson, F. (2018). Ball-and-finger system: Modeling and optimal trajectories. *Multibody System Dynamics*, *44*(2), 163–189. https://doi.org/10.1007/s11044-018-9617-8

Mathiesen, F. B. (2022). Bound propagation.

Shokouhi, F., & Davaie Markazi, A. H. (2018). A new continuous approximation of sign function for sliding mode control.

Yin, G., & Zhu, C. (2010, January). *Hybrid Switching Diffusions: Properties and Applications* (Vol. 63). https://doi.org/10.1007/978-1-4419-1105-6

Zhang, H., Weng, T.-W., Chen, P.-Y., Hsieh, C.-J., & Daniel, L. (2018, November). Efficient Neural Network Robustness Certification with General Activation Functions [arXiv:1811.00866 [cs, stat]]. https://doi.org/10.48550/arXiv.1811.00866