# Abstraction-based Control Synthesis of Unknown Systems Using Neural Network Approximations

by

Shengbo Zou

A research paper
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Waterloo, Ontario, Canada, 2023

## Abstract

We consider the control synthesis problem of unknown nonlinear dynamical systems with linear temporal logic (LTL) formulas as specifications. First, we use a neural network to learn the flow map of the unknown system by performing a regression task with a sampled dataset of input-output data pairs. Then, the generalization error of the approximation is derived by estimating the Lipschitz constants of the flow map, and the approximated neural networks. By treating the generalization error as the disturbance in the given unknown system, the abstraction can be obtained for specifications in the form of LTL. It follows that standard tools such as ROCS can be applied to solve the control synthesis problem. The effectiveness of the proposed algorithm is illustrated by a reach-avoid task for a car model.

## Acknowledgements

I would like to thank my supervisor, Prof. Jun Liu, for giving me guidance and support throughout this research project. I would also like to thank Zhibing Sun and Ruikun Zhou for spending their precious time helping me understand the subject. Without them, this report would not have been possible. I would also like to thank Prof. Xinzhi Liu for agreeing to be the second reader for this report.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

While designing safety-critical systems, safety guarantees while complying with complex objectives during operation are a major concern. Systems like power grids, autonomous vehicles, life support systems, and emergency service dispatch systems all fall under the category of safety-critical systems. Control synthesis for these systems had become a popular research topic in recent years. These systems can be formulated as a control system with continuous state spaces with arbitrary dimensions. There are existing control synthesis methods that provide a satisfaction guarantee for complex specifications in the form of Linear Temporal Logic (LTL) [3] [14]. However, integrating these methods with continuous state spaces can be a challenge.

One of the common methods to resolve this issue is to utilize abstraction-based controller design schemes [3] [14]. The first step of the abstraction process is to convert the continuous system into a finite abstraction, which is usually achieved by discretizing the state space and action space. Depending on the system, the finite abstraction can be linked to the original system through a behavioral relation such as feedback refinement relations or alternating bisimulation relations [14]. By leveraging this relation, trajectories found in the finite abstraction can be mapped to the original system. Because of this, a controller designed for finite abstractions can be converted to a controller for the original continuous system. Controllers synthesized from this approach are also considered formal because of the satisfaction guarantees on the specifications.

However, for this type of abstraction-based approach to realize reach-avoid tasks, it is essential to have a precise mathematical model of the continuous system due to the fact that the abstraction process performs reachability analysis over the dynamics of the given continuous system, which requires knowledge of the exact dynamical equations. In theory, the exact dynamical equations can be derived from sources such as physics laws and differential equations, but given the multi-paradigm nature of many modern real-world control systems, finding the exact dynamical equations can be a complex problem. In this case, it is infeasible to derive the dynamical equations of the original continuous system. Over the past decades, various system identification techniques were developed to derive the (approximate) expressions of the dynamics. Among them, neural networks have shown their effectiveness in performing function regression tasks with big data/measurements. Therefore, a promising idea is to leverage neural networks to approximate underlying differential equations for unknown dynamical equations by taking the states and actions as the inputs while the future states as the outputs.

In this research paper, we propose using neural networks to approximate the unknown dynamics equations to address this issue, then apply the trained model in existing control synthesis tools such as Robustly Complete control Synthesis (ROCS) to provide a satisfaction guarantee for temporal specifications. One of the advantages of this approach is that it does not require an accurate mathematical model of the continuous system. Another advantage of this approach is that it provides formal guarantees to satisfy temporal specifications since we derive rigorous bounds from the neural network and use them as disturbances during the control synthesis process.

The rest of this research paper is organized as follows. Chapter 2 covers the preliminary background information, which mainly focuses on introducing core concepts of neural networks, abstraction-based methods, and the syntax of LTL. Chapter 3 provides a more detailed overview of the abstraction-based control synthesis method. In chapter 4, we present how the error bound is derived from the neural network approximation and how it is applied in the abstraction process. Chapter 5 is a case study with the proposed approach. Finally, chapter 6 contains the conclusion and related future research topics.

# Chapter 2

# Background

## 2.1 Neural Networks

Neural network (NN) is one of the most studied research areas in recent years due to its applicability in various fields, including natural language processing, image recognition, stock market prediction, etc. The original idea of neural network algorithms was derived from the biological structure of the human brain, and neurons are the fundamental building blocks of an artificial neural network (ANN) [1].

Neurons in ANN are grouped into layers with a specific ordering among layers, and the connection between layers is typically one-way. Each layer has its own weight matrix and bias vector, both of which are updated during the training process. Each layer also has its own activation function, but depending on the application, there may not be one for the output layer. The weight matrix connects each individual neuron, and the bias vector helps fine-tune the output so that the network can learn the pattern better.

There exist many different architectures for ANN, including convolutional neural networks that perform well for image-related tasks, recurrent neural networks that perform well with prediction problems, etc. In this work, we mainly focus on its basic form, which is feed-forward ANN, since we are mainly performing a function regression task. This architecture choice is also supported by the Universal Approximation Theorem:

**Universal Approximation Theorem** [5]**.** *Let $C(X, \mathbb{R}^m)$ denote the set of continuous functions from a subset $X$ of a $\mathbb{R}^n$ space to $\mathbb{R}^m$ space. Let $\sigma \in C(\mathbb{R}, \mathbb{R})$. $\sigma \circ x$ denotes $\sigma$ applied to each elements in $x$.*

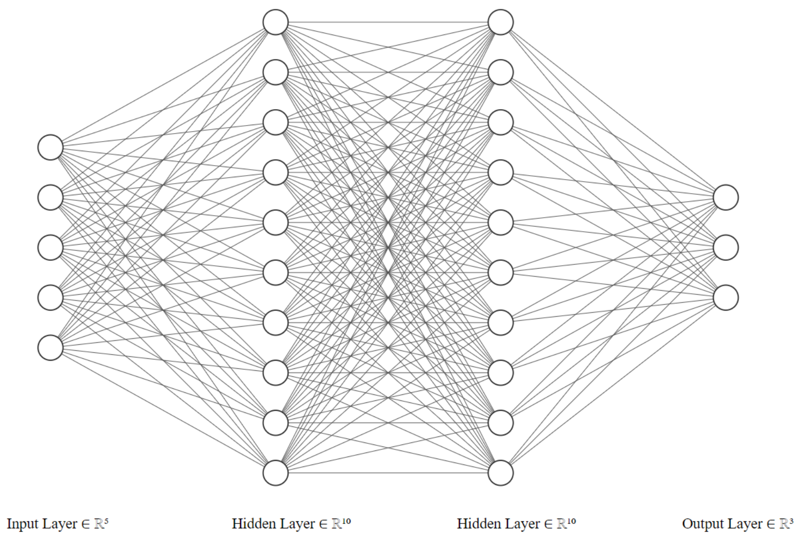

Figure 2.1: Typical structure of a feedforward ANN

*Then* $\sigma$ *is not polynomial if and only if for every* $n \in \mathbb{N}, m \in \mathbb{N}$, *compact set* $K \subseteq \mathbb{R}^n, f \in C(K, \mathbb{R}^m), \varepsilon > 0$ *there exist* $k \in \mathbb{N}, A \in \mathbb{R}^{k\times n}, b \in \mathbb{R}^k, C \in \mathbb{R}^{m\times k}$ *such that*

$$\sup_{x\in K}\|f(x) - g(x)\| < \varepsilon$$

*where* $g(x) = C \cdot (\sigma \circ (A \cdot x + b))$

Therefore, by choosing appropriate configurations such as layer size and activation functions, a feed-forward ANN is able to capture complex data patterns in function regression tasks. Common activation functions for regression tasks include hyperbolic tangent, sigmoid function, rectified linear unit function, etc. If the original function is composed of sine and cosine functions, the sine function can be used as an activation function as well [10].

## 2.2 Abstraction-based Method

For solving control synthesis problems directly in continuous spaces, there are two major problems. The first problem is the fact that there are infinite states in a continuous space.

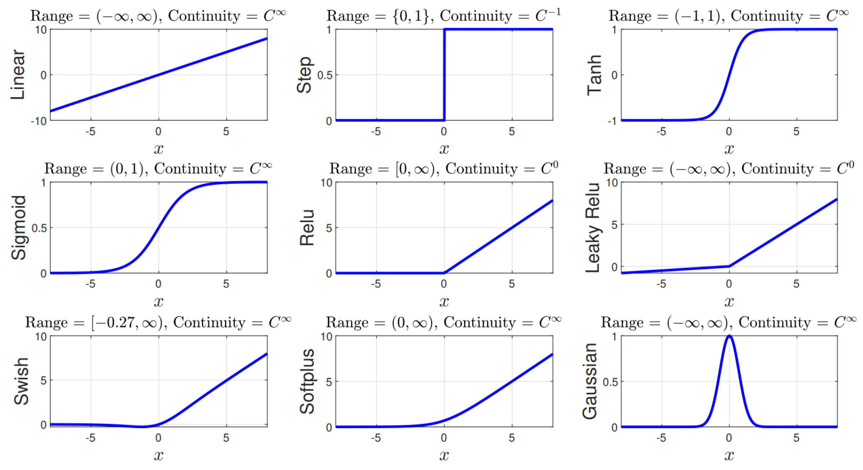

Figure 2.2: Common activation functions [6]

In terms of computation, infinitely many states may not be able to be represented or computed feasibly. The other problem originates from the fact that real number computations are more expensive than integer computations. Therefore, the abstraction process can reduce the infinite space into finite space, which helps computers to solve control synthesis problems in finite time. Since we would like to use computers to solve control synthesis problems, abstraction is necessary to ensure the practicality of the method.

In recent years, researchers have proposed the abstraction-based controller design paradigm. This procedure first constructs a finite-state abstraction of the original continuous dynamical system and then defines a relation between the abstraction and the original system. This abstraction can then be used to design output-feedback controllers [9]. There are other works that incorporate perturbation during the abstraction process to compute controllers that are resilient to high disturbance spikes [12]. In this work, the abstraction is mainly done by discretizing the continuous spaces into uniform grids and then constructing the relation in the form of a non-deterministic transition system. The details are included in the following chapter.

## 2.3 Linear Temporal Logic

Since this work uses tools like Spot [4] to convert LTL formulas to automatons, we will also introduce the syntax of LTL here. The main components of LTL include a finite set of propositional variables $AP$, logical operators $\neg$ (not) and $\vee$ (or), and temporal modal operators $\mathbf{X}$ (next) and $\mathbf{U}$ (until). Propositional variables are conditions or variables that can either be true or false, and logical operators can group propositional variables together to express complex boolean expressions. The temporal modal operators are used to specify the future conditions, e.g., a condition will eventually be true. The set of LTL formulas w.r.t $AP$ is inductively defined with the following rules:

- If $p \in AP$ then $p$ is an LTL formula.
- If $\psi, \varphi$ are LTL formulas, then $\neg\psi, \varphi \vee \psi, \mathbf{X}\psi, \varphi\mathbf{U}\psi$ are LTL formulas.

There are other temporal modal operators that are widely used, such as **G** (globally), **F** (finally), **R** (release), etc.

# Chapter 3

# Robustly Complete Control Synthesis (ROCS)

Since this work is built on an existing abstraction-based method from ROCS, this section would provide high-level steps explaining how the existing tool operates. There are two major steps. The first step is to calculate the abstraction of the system, and the second step is to use the LTL formula along with the abstraction to synthesize a feedback controller, which includes both the winning set and the winning strategy of the control synthesis problem.

## 3.1 Abstraction Calculation

Here are the environment specifications required to calculate the abstraction of the system:

$$\Sigma :< \mathcal{T}, \mathcal{X}, \mathcal{U}, \mathcal{D}, f, AP, L >$$

- $\mathcal{T} = \mathbb{Z}_{\geq 0}$ is a set of time instances.
- $\mathcal{X} \subseteq \mathbb{R}^n$ is a non-empty set of states, also known as state space.
- $\mathcal{U} \subseteq \mathbb{R}^m$ is a non-empty set of control inputs, also known as control space.
- $\mathcal{D} \subseteq \mathbb{R}^n$ is a set of bounded perturbations.

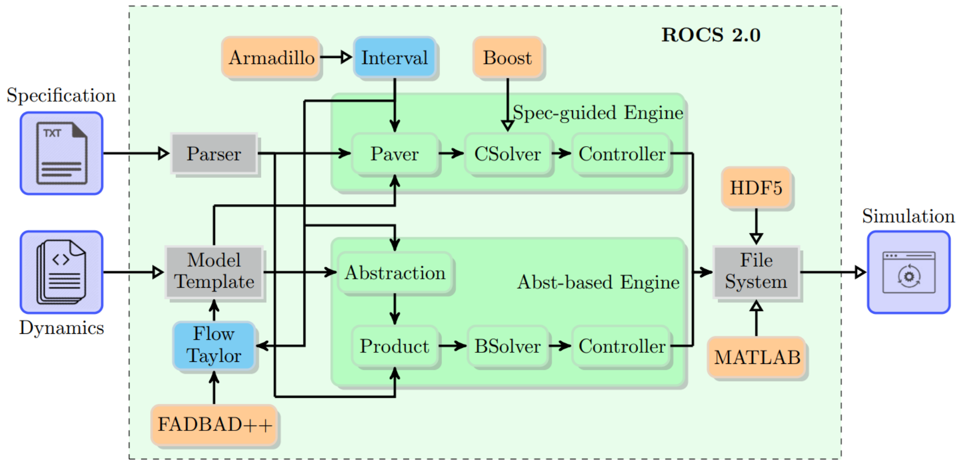

Figure 3.1: ROCS architecture diagram [8]

- $f : \mathbb{R}^n \times \mathbb{R}^m$ is a continuous function describing how the control input signals translated to changes in state space. This is also known as the dynamics function of the system.
- $AP$ is a set of atomic propositions, which are conditions that are true or false. In practice, it is often a function that checks the current state of the agent against some desired properties.
- $L : \mathcal{X} \rightarrow 2^{AP}$ is a labeling function, which takes the current state of the agent as input, encodes each atomic proposition into binary bits, and then combines all generated binary bits into one numerical value.

The first step of the abstraction process is to discretize all the continuous components in the environment specifications, which include the state space, control space, and the dynamics function of the system. For state space and control space, the discretization process is to simply divide each dimension of the space into grids of the same size, then assign an index to each divided hypercube lexicographic order. As for the dynamics function, it usually starts in its continuous form as a continuous time ordinary differential equation in the form of $\dot{x} = f(x, u)$. Then, one can discretize it with some fixed sampling time $h$, yield

$$\text{Discrete} : x_{k+1} = \Phi(x_k, u_k).$$

The discretization process typically uses one-step Euler integration $x_{k+1} = \Phi(x_k, u_k) = x_k + f(x_k, u_k)h$, but it can also be replaced with other numerical integration techniques as required by the specific control synthesis problem.

The second step is to pass all discretized states in the state space through the labeling function to map the atomic propositions onto state space.

The last step, which is also the most computation-intensive step, is to create the non-deterministic transition system (NTS) of the system. NTS is a graph constructed on the state space with each vertex being a discretized state, and each transition being a discretized action. At the beginning of this step, all the partitioned states and actions are stored as continuous intervals on each dimension of the state space and control space, respectively. Next, the program will iterate through all unique combinations of $(x, u)$ through the dynamics function, during which interval analysis will be performed as well. The interval analysis is done by overloading the operators to incorporate interval analysis into each interval arithmetic operation in the dynamics function. The resulting graph can be non-deterministic based on the disturbance applied after the dynamics function. Lastly, these continuous intervals are converted to discretized indices. The final NTS is then passed into the discrete solver for control synthesis.

## 3.2 Discrete Solver

The discretized NTS constructed from the abstraction process is in the form of the following definition:

**Definition** (non-deterministic transition system [13]). A Non-deterministic Transition System (NTS) is a 4 tuple $(X, \Sigma, \xi, L)$, where

- $X_d$ is the set of discrete system states.
- $\Sigma_d$ is the set of discrete control actions.
- $\xi_d : X_d \times \Sigma_d \to 2^{X_d}$ is a non-deterministic transition function.
- $L \subseteq 2^{AP}$ is the set of labels.

The goal of the discrete solver is to find the winning set of initial conditions and winning control strategy for the given NTS$= (X_d, \Sigma_d, \xi_d, L)$ and temporal specification $\varphi$. All trajectories induced by the winning set and winning strategy are guaranteed to satisfy the accepting condition of the temporal specification, regardless of the non-determinism from the NTS.

The first step is to convert $\varphi$ into a deterministic automaton (DA). The DA will have an accepting set (Acc), which contains all the accepting vertices in DA, and an accepting condition, which describes how the accepting vertices should be visited. This is done with existing tools such as Spot [4].

The second step is to take the product of NTS and DA to construct the product space (P$\mathcal{A}$). The label of each vertex in the P$\mathcal{A}$ graph is a unique combination of NTS vertices ($x$) and DA vertices ($q$).

The third step is to define the objective set and winning condition of the game ($\mathcal{G}$) in P$\mathcal{A}$. The objective set contains vertices with labels of $(x, q)$ where $q \in$ Acc. The accepting condition of DA is used to construct the winning condition of $\mathcal{G}$.

The fourth step is to solve G in PA. The details of the algorithm for this step are well-documented in [13].

The last step is to map the winning set and winning strategy indices from P$\mathcal{A}$ back to NTS.

# Chapter 4

# Neural Network Approximations

In this work, we consider a nonlinear control system of the form

$$\dot{x} = f(x, u), \quad x(0) = x_0, \tag{4.1}$$

where $x \in \mathcal{X} \subseteq \mathbb{R}^n$ is the state of the system, $u \in \mathcal{U} \subseteq \mathbb{R}^m$ is the piecewise constant control input with respect to a sampling time $\tau > 0$. Then, for every initial condition $x_0 \in \mathcal{X}$ and $t \geq 0$, the flow map $\phi(t, x_0, u) : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ satisfies:

$$\phi(0, x_0, u) = x_0 \tag{4.2}$$

$$\frac{d}{dt}\phi(t, x_0, u) = f(\phi(t, x_0, u), u(t)). \tag{4.3}$$

**Assumption 1** (Lipschitz Continuity)**.** *The right-hand side of the nonlinear system (4.1) is assumed to be Lipschitz continuous, i.e.,*

$$\|f(x, u) - f(y, v)\| \leq L_f \|(x, u) - (y, v)\| \;\; \forall x, y \in \mathcal{X} \quad \textit{and} \quad \forall u, v \in \mathcal{U},$$

*where $L_f$ is called the Lipschitz constant; $(x, u)$ and $(y, v)$ denote the concatenation of the corresponding two vectors.*

## 4.1 Generalization Error

In this work, we assume the dynamics $f$ is unknown, but we have the knowledge of the upper bound of the Lipschitz constant $L_f$. As a result, if $\mathcal{X}$ is a compact set in $\mathbb{R}^n$ with a fixed $t = \tau$ with a known constant input $u_c$, the flow map $\phi$ is also Lipschitz continuous, and the corresponding Lipschitz constant is denoted as $L_\phi$. By applying the well-known Grönwall's inequality, we have the upper bound:

$$L_\phi \leq \|x_0\| e^{L_f \tau} \tag{4.4}$$

For given $x_0$, we denote the states on the flow map at the fixed time $\tau$ as $\phi^\tau(x_0, u)$. Then, we can use a neural network to approximate $\phi^\tau(x_0, u)$. By using smooth activation functions in the deep neural networks, we have the neural approximation $\psi_{NN}$ is also Lipschitz continuous with the Lipschitz constant $L_\psi$.

Assume that $(x_0, u)$ is a pair of unsampled states and inputs in the state space and input state respectively, and $(y_0, v)$ is its nearest known sample used in training or testing the neural network for learning the dynamics. Let $\delta > 0$ be such that $\|(x_0, u) - (y_0, v)\| \leq \delta$ holds for all such $(x_0, u)$ and $(y_0, v)$. Denote $\alpha$ as the maximum of the 2-norm loss among all known samples, which can be from either the training dataset or the test dataset. Then the generalization error of the neural network approximation is bounded as

$$\begin{aligned} \|\phi^\tau(x_0, u) - \psi_{NN}(x_0, u)\| &\leq \|\phi^\tau(x_0, u) - \phi^\tau(y_0, v)\| + \|\phi^\tau(y_0, v) - \psi_{NN}(y_0, v)\| + \\ &\quad \|\psi_{NN}(y_0, v) - \psi_{NN}(x_0, u)\| \\ &\leq L_\phi \delta + \alpha + L_\psi \delta \\ &< \epsilon. \end{aligned} \tag{4.5}$$

By choosing $\epsilon$, we obtain the upper bound of the generalization error.

## 4.2 Accommodating the Generalization Error in Abstraction

From the previous subsection, we know that if we have access to $\psi_{NN}(x_0, u)$, then the flow map with respect to the actual dynamics is bounded by

$$\|\phi^\tau(x_0, u)\| < \|\psi_{NN}(x_0, u)\| + \epsilon. \tag{4.6}$$

In the abstraction, we use the approximated flow map $\psi_{NN}(x_0, u)$ to generate the NTS, as discussed in Section 3.1. For a certain sampling time $\tau$, we have the constant control input, denoted as $u_c$. Consequently, the controllers are synthesized by the generated NTS. By satisfying the LTL specifications, the states in the approximated flow map $\psi_{NN}(x_0, u_c)$ are within some intervals, i.e., $\hat{x}_i = [\inf_i, \sup_i] \in X, i = \{1, \cdots, n\}$ denote the state space interval of the $i$-th dimension, for given $x_0$ and constant $u_c$ with a fixed time $\tau$.

Let $\epsilon_i$ denote the generalization error for the $i$-th dimension. We have the states $x_i$ on the trajectory of the true flow map $\phi^\tau(x_0, u)$ satisfy:

$$x_i \in [\inf_i + \epsilon_i, \sup_i - \epsilon_i]. \tag{4.7}$$

Therefore, one can treat $\epsilon$ as the deterministic perturbation in each step in the abstraction process and use it to shrink the interval after passing the infimum and supremum of each dimension through the approximated dynamics function. In terms of implementation, the generalization error is applied using the following procedure:

1. Divide $X$ into two vectors:
   - $X_{inf} = \{\inf_1, \cdots, \inf_n\}$
   - $X_{sup} = \{\sup_1, \cdots, \sup_n\}$
2. Compute $a = \psi_{NN}(X_{inf}, u_c)$ and $b = \psi_{NN}(X_{sup}, u_c)$
3. Compute $\inf'_i = \min(a_i, b_i) + \epsilon_i, \sup'_i = \max(a_i, b_i) - \epsilon_i$
4. Construct $x'_i = [\inf'_i, \sup'_i] \in X'$

Since the neural network is trained on a dataset with some fixed $\tau$, $X'$ acts as the approximation of $\phi^\tau(X, u)$

# Chapter 5

# Case Study

In this work, we mainly focus on realizing a reach-avoid problem for a classic car control model [2].

## 5.1 Model of Robot Car

The diagram of the robot car model is illustrated in Figure 5.1. In this model, $\mathcal{X} \subseteq \mathbb{R}^3$,

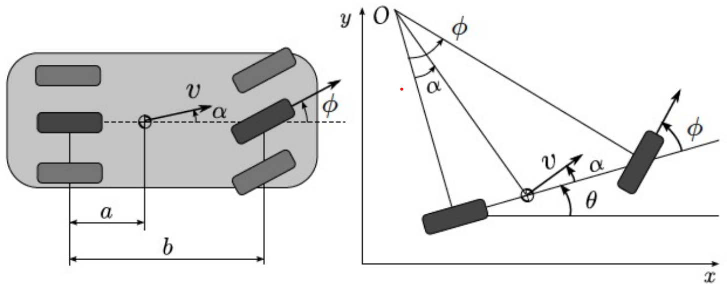

Figure 5.1: Model of Robot Car

which contains the position of the vehicle $(x, y)$, and its orientation $\theta$, $\mathcal{U} \subseteq \mathbb{R}^2$, which contains the velocity $v$ and steering angle $\phi$. The dynamics function of the system is given by the following ODE:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos(\gamma+\theta)\cos(\gamma)^{-1} \\ v\sin(\gamma+\theta)\cos(\gamma)^{-1} \\ v\tan(\phi) \end{bmatrix}, \gamma = \arctan(\tan(\phi)/2) \tag{5.1}$$

## 5.2 Reach-avoid task with LTL specifications

We perform the reach-avoid task on the environment, as shown in Figure 5.2.

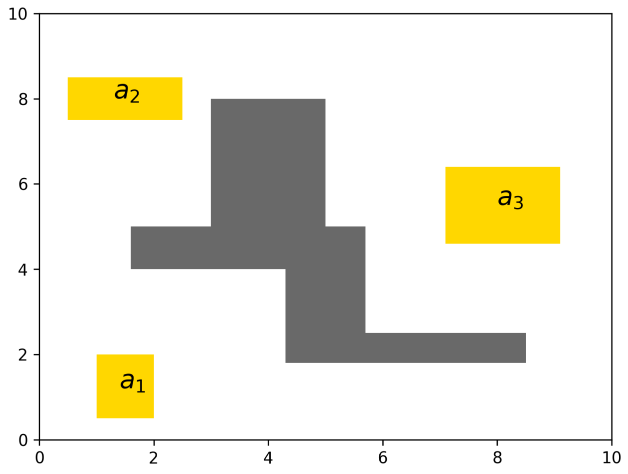

Figure 5.2: Environment for the reach-avoid task.

The exact environment specifications are detailed as below:

- $\mathcal{X} = [0, 10] \times [0, 10] \times [-\pi, \pi]$
- $\mathcal{U} = [-1, 1] \times [-1, 1]$
- $\mathcal{O} = \{o_1, o_2, o_3, o_4\}$ is the set of obstacles, which is the gray area in 5.2:
  - $o_1 = [1.6, 5.7] \times [4.0, 5.0] \times [-\pi, \pi]$
  - $o_2 = [3.0, 5.0] \times [5.0, 8.0] \times [-\pi, \pi]$

- $o_3 = [4.3, 5.7] \times [1.8, 4.0] \times [-\pi, \pi]$
- $o_4 = [5.7, 8.5] \times [1.8, 2.5] \times [-\pi, \pi]$

- $\mathcal{A} = \{a_1, a_2, a_3\}$ is the set of target area in $\mathcal{X}$, which are the yellow areas in 5.2:
  - $a_1 = [1.0, 2.0] \times [0.5, 2.0] \times [-\pi, \pi]$
  - $a_2 = [0.5, 2.5] \times [7.5, 8.5] \times [-\pi, \pi]$
  - $a_3 = [7.1, 9.1] \times [4.6, 6.4] \times [-\pi, \pi]$

The LTL specification here is the robot should eventually visit area $a_1, a_2, a_3$ in that order once, then go back to $a_1$ without visiting $a_2$, and the LTL formula is

$$\varphi = \mathbf{F}(a_1 \wedge \mathbf{F}(a_2 \wedge \mathbf{F}(a_3 \wedge (\neg a_2 \mathbf{U} a_1)))).$$

During the process, the robot must remain in $\mathcal{X}$ while avoiding $\mathcal{O}$ while taking control inputs from $\mathcal{U}$. The grid size for $\mathcal{X}$ is 0.2, and the grid size for $\mathcal{U}$ is 0.3, following the ones in [7].

## 5.3 Dataset Generation

The training dataset and testing dataset is generated by randomly sampling points $(x, u)$,$x \in \mathcal{X}, u \in \mathcal{U}$ and $y = \Phi(x, u) \in \mathcal{X}$. Here $\Phi(x, u)$ is constructed by performing Runge-Kutta method of order 5(4) on the ODE of the system with sampling time $\tau = 0.03$. For better approximation accuracy around the boundary of domains in both $\mathcal{X}$ and $\mathcal{U}$, the random points chosen for the training set are from a slightly larger domain compared to the environment specification above. The exact domains used are the following:

- Training:
  - $\mathcal{X} = [-1, 11] \times [-1, 11] \times [-3.5, 3.5]$
  - $\mathcal{U} = [-1.2, 1.2] \times [-1.2, 1.2]$
- Testing:
  - $\mathcal{X} = [0, 10] \times [0, 10] \times [-3.5, 3.5]$
  - $\mathcal{U} = [-1, 1] \times [-1, 1]$

There are 5,000,000 points in the training dataset and 50,000 points in the testing dataset.

## 5.4 Neural Network Model Configuration

Here are the configurations used for training:

- Optimizer: Adam with the initial learning rate of 0.01 and no weight decay.
- Learning Rate Scheduler: Reduce learning rate on training loss plateau with default settings in PyTorch [11].
- Dataloader:
  - Training dataset: The batch size is 1024 with shuffling.
  - Testing dataset: Load the entire testing dataset in one batch without shuffling.
- Loss function: Mean square error with sum as the reduction method for each batch and taking the average of each batch at the end of each epoch.
- Epoch: 1000 epochs.

Because an error bound needs to be derived from the trained model, the sum reduction was chosen for the loss function so it would be easier to compare performance while tweaking the model architecture.

In order to derive a more rigorous generalization error from the neural networks, we train one model for each dimension of the state space. Also, since we need to calculate the Lipschitz constant of each model accurately, each model only has a single hidden layer with 200 neurons. The only difference among the models is the activation function. For $x, y$ we use sine as the activation function, and for $\theta$ we use hyperbolic tangent as the activation function. The final loss values from the testing dataset are [3.7e-3, 1.1e-2, 1.0e-3] for $x, y, \theta$ respectively. For comparison, we provide test results both with and without the generalization error.

## 5.5 Experimental Results

We run two scenarios with the foregoing settings. The first one is the case in which we assume there are no perturbations and see if the proposed algorithm could generate similar results with the neural network approximated dynamics as the actual ones. The second one is to include the generalization errors as the deterministic perturbations to synthesize controllers for the actual with formal guarantees.

### 5.5.1 Test case without perturbations

In this case, we perform a test without the perturbations from the generalization error in the abstraction process to synthesize controllers with both the true dynamics and the NN-approximated dynamics. The sizes of the winning set are 74,994 for true dynamics and 75,607 for NN-approximated dynamics. The initial state for the sample trajectories here is (3.0,1.0,-3.4). Figure 5.3 shows one trajectory generated by performing abstraction with the true dynamics, while Figure 5.4 shows the one from the same initial condition with the NN-approximated dynamics. The trajectory from NN-approximated dynamics clearly satisfies the LTL specification.

### 5.5.2 Experiments with perturbations

In this scenario, in order to obtain more accurate simulation results, we change the grid size for $\mathcal{X}$ from 0.2 to 0.1 for all dimensions in the state space, and the abstraction sampling time from 0.3 s to 0.15 s. The detailed reasons for this change will be detailed in the observations section. We perform the abstraction with this setting for control synthesis with respect to both true dynamics and NN-approximated dynamics.

With the true dynamic, the size of the winning set is 597,066, while the one with the NN-approximated dynamics is 597,853. The initial state for the sample trajectories here is (0.1, 0.9, -3.5). As in the previous subsection, the trajectory with the controller obtained from the synthesis process with respect to the true dynamics is illustrated in Figure 5.5. For the approximated dynamics case, the generalization error is calculated using the procedure detailed in Section 4.2. Figure 5.6 shows the trajectory with the controller in this case with generalization error applied as the deterministic perturbations. Obviously, with the robust controller synthesized with respect to the NN-approximated dynamics, the trajectory satisfies the given LTL specification. The detailed parameters for calculating the generalization errors can be found in Table 5.1.

Table 5.1: Parameters in generalization error calculation

| | $L_f$ | $L_\phi$ | $L_\psi$ | $\tau$ | $\delta$ | $\alpha$ | $\epsilon$ |
|---|---|---|---|---|---|---|---|
| $x$ | 2.13 | 10.66 | 1.58 | 0.03 | 5e-4 | 1.1e-3 | 7.2e-3 |
| $y$ | 2.13 | 10.66 | 1.58 | 0.03 | 5e-4 | 1.7e-3 | 7.8e-3 |
| $\theta$ | 3.76 | 3.92 | 1.58 | 0.03 | 5e-4 | 6.7e-4 | 3.4e-3 |

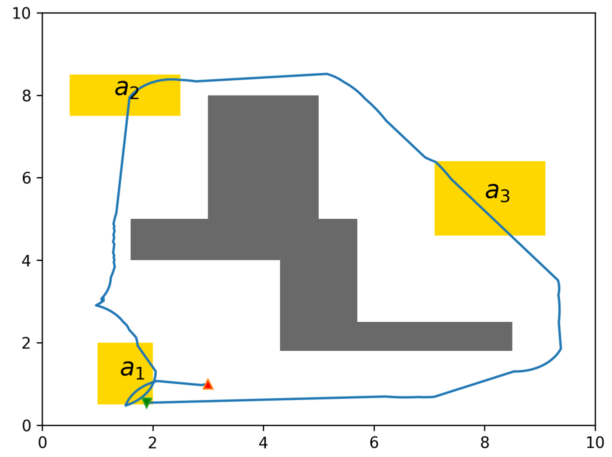

Figure 5.3: A trajectory with the controller synthesized from the abstraction with the true dynamics, which is proposed in [7].

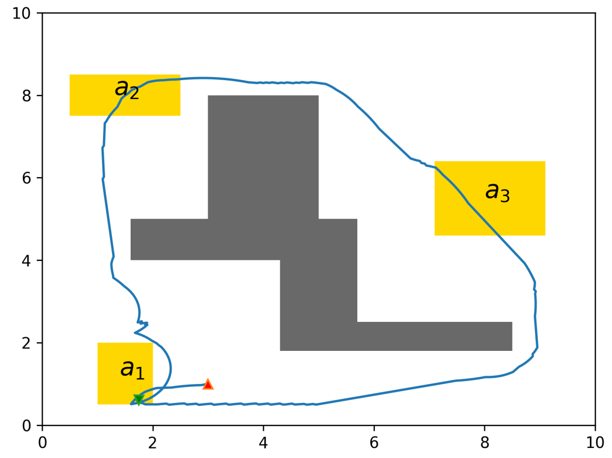

Figure 5.4: A trajectory with the controller synthesized from the abstraction with the NN-approximated dynamics in the test case, without taking the generalization error into consideration.

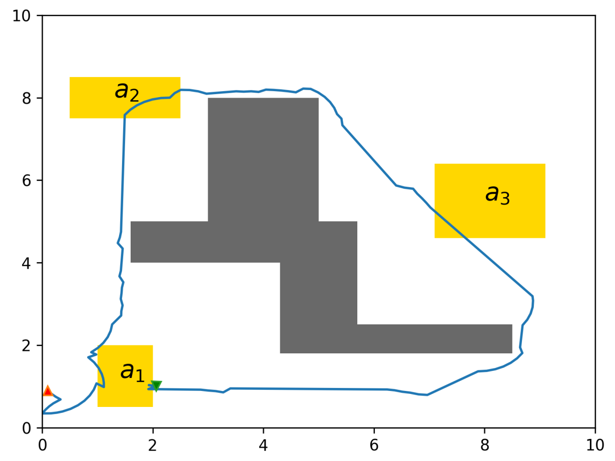

Figure 5.5: A trajectory with the controller synthesized from the abstraction with the true dynamics, with smaller grid size and sampling time.

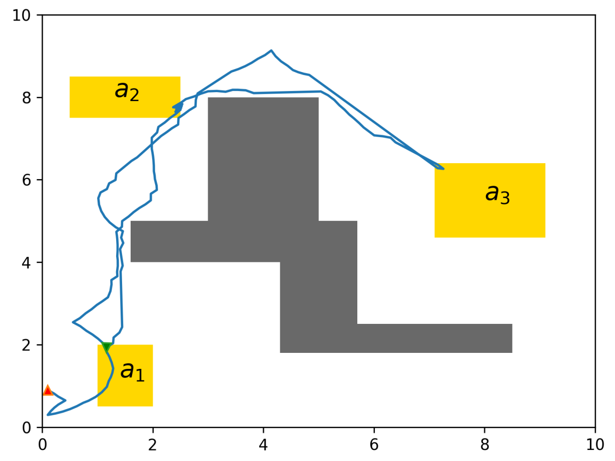

Figure 5.6: A trajectory with the controller synthesized from the abstraction with the NN-approximated dynamics with generalization errors as perturbations.

### 5.5.3 Observations and discussions

Due to the nature of the abstraction-based method, when running concrete examples, the sampling time and grid size of the state space need to be adjusted in conjunction for the final winning set to be non-empty. If the sampling time is reduced without changing the grid size accordingly, there may be a lot of states in the discretized NTS that only have outward transitions pointing to themselves, which in turn results in an empty winning set. However, if the model is trained on a dataset generated with a large sampling time, we could not achieve an accurate approximation without increasing the depth of the model, which in turn creates difficulties when computing the Lipschitz constant of NNs. We also tried to reduce the grid size based on the small sampling time for training, but due to the curse of dimensionality, the abstraction computation quickly became infeasible due to the exponential growth in number of partitions. The workaround here is to train the model with a small sampling time for better accuracy with simple architecture, and then use the trained model as one-step integration similar to a single step in Runge-Kutta method. Therefore, if the example is designed with a large sampling time in mind, we can perform multiple passes through trained NN to achieve the desired sampling time. As shown in Figure 5.4, the trajectory is very similar to Figure 5.3 using this workaround.

Unfortunately, this workaround introduces complications when applying the generalization error during the abstraction process. The generalization error needs to be applied using the procedure in 4.2 after each NN pass, which would result in a larger accumulated disturbance during the interval analysis. As a result, the abstraction is not as accurate, which results in the differences between Figure 5.5 and Figure 5.6.

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusions

In this research paper, we propose an algorithm to approximate the flow map of unknown dynamics of nonlinear systems and then use the approximation to synthesize robust controllers with abstraction-based methods for the given systems. This approach first collects data from the unknown dynamical system, then trains an NN on the collected data to approximate the flow map of the underlying dynamical functions. Next, the generalization error of the approximation is derived by using the Lipschitz constants of the dynamics and the corresponding neural network approximations. As a consequence, the generalization error is regarded as perturbation during the abstraction process. This abstraction is then used to synthesize the robust controller with the formal guarantees to satisfy LTL specifications for the unknown dynamics. Lastly, we demonstrate the effectiveness of the proposed approach with a reach-avoid task for a robot car.

## 6.2 Future Work

In the calculation, the generalization error relies on the Lipschitz constant estimation of the neural networks, which is calculated by using dReal. However, this tool can only find $L_\psi$ for shallow neural networks. As a result, the bottleneck of this method is how to estimate the Lipschitz constants with deep neural networks, as having more hidden layers can greatly improve the approximation accuracy when the underlying dynamical functions are complex or the sampling time is large. Therefore, computing the Lipschitz constant for deep neural networks which approximate more complex nonlinear systems is left to future research.

Another area of future research would be the parallelization of the abstraction process. Since each $(x, u)$ combination is independent from each other, most dynamical function evaluations can be parallelized without any potential issues. This is especially valuable when incorporating NN evaluations in this process.

# References

[1] Charu C Aggarwal et al. Neural networks and deep learning. *Springer*, 10(978):3, 2018.

[2] Karl Johan Åström and Richard M Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2021.

[3] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. *Formal methods for discrete-time dynamical systems*, volume 89. Springer, 2017.

[4] Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, et al. From spot 2.0 to spot 2.10: What's new? In *International Conference on Computer Aided Verification*, pages 174–187. Springer, 2022.

[5] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[6] Ameya D Jagtap and George Em Karniadakis. How important are activation functions in regression and classification? a survey, performance comparison, and future directions. *Journal of Machine Learning for Modeling and Computing*, 4(1), 2023.

[7] Yinan Li and Jun Liu. Rocs: A robustly complete control synthesis tool for nonlinear dynamical systems. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*, pages 130–135, 2018.

[8] Yinan Li, Zhibing Sun, and Jun Liu. Rocs 2.0: An integrated temporal logic control synthesis tool for nonlinear dynamical systems. *IFAC-PapersOnLine*, 54(5):31–36, 2021.

[9] Rupak Majumdar, Necmiye Ozay, and Anne-Kathrin Schmuck. On abstraction-based controller design with output feedback. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, pages 1–11, 2020.

[10] Giambattista Parascandolo, Heikki Huttunen, and Tuomas Virtanen. Taming the waves: sine as activation function in deep neural networks. 2016.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[12] Stanly Samuel, Kaushik Mallik, Anne-Kathrin Schmuck, and Daniel Neider. Resilient abstraction-based controller design. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, pages 1–2, 2020.

[13] Zhibing Sun. Control of non-deterministic transition systems for linear temporal logic specifications. Master's thesis, University of Waterloo, 2021.

[14] Paulo Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009.