Åse Victoria Neverlien

Feedback control and AI in closed loop: Stability and robustness

Master's thesis in Cybernetics and Robotics
Supervisor: Jan Tommy Gravdahl
August 2019

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Åse Victoria Neverlien

Feedback control and AI in closed loop: Stability and robustness

Master's thesis in Cybernetics and Robotics
Supervisor: Jan Tommy Gravdahl
August 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk
Institutt for teknisk kybernetikk

MSc thesis assignment

Name of the candidate:	Åse Victoria Neverlien
Subject:	Engineering Cybernetics
Title:	Feedback control and AI in closed loop: Stability and robustness

Background: *The topic of this thesis work is to gain insights into the stability properties of control systems when Artificial Intelligence (AI) or Machine Learning (ML) methods are introduced into the control loop. The work consists of a general part, and a specific part where a method from literature should be applied to a specific example.*

Tasks:

1. Perform a literature review on the topic of stability and robustness of control systems when combining AI/ML and feedback control. A starting point can e.g. be [1], but a broad literature review should be undertaken.
2. Study and present the compressor surge control problem and the two-state Greitzer compressor model in, see [2].
3. Based on the results in [1], repeat the analysis for the compressor model, more specifically use the model in equations (7)-(10) in [2] and the control in equation (18) in [3].

To be handed in by: 2019-08-12

Jan Tommy Gravdahl

Jan Tommy Gravdahl
Professor, supervisor

Co-supervisor: Signe Moe

[1] Richards, Berkenkamp and Krause, “The Lyapunov Neural Network: Adaptive Stability Certification for Safe Learning of Dynamic Systems,” https://arxiv.org/abs/1808.00924, 2018
[2] J.T. Gravdahl and O. Egeland, "Compressor surge and rotating stall: Modeling and control ", Series: Advances in Industrial Control, Springer Verlag, London, 1999
[3] Gravdahl, J.T. and Egeland, O. Compressor surge control using a close-coupled valve and backstepping, Proc. of the 1997 American Control Conference, Albuquerque, NM., pp. 982 -986, vol.2, June 1997.

Preface

This thesis is made as a completion of the Master of Science education in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). The research has been made in collabration with the Department of Engineering Cybernetics at NTNU.

I would like to thank my supervisor at NTNU Jan Tommy Gravdahl and my co-supervisor Signe Moe for their support. Jan Tommy for giving me constructive feedback and advise in our regular meetings, and Signe for providing valuable insights and being available and assistive throughout the research period.

The following paragraph describes the implementation and code the author has found existing libraries for. However, significant understanding and modification was required to tailor the APIs to fit this thesis problems.

The code required to solve the Neural Network Lyapunov function is mainly created by Richards et al. (2018).

Abstract

Historically, one would handle surge in compressor systems by operating far from the surge line, and by doing so, simply avoiding the risk of surge. This will ensure both the mechanical integrity and the safety of the machines, but it will also limit the range of mass flow for which the compressor can be used. As surge can cause a reduction in performance, or even damage the compressor, it is of major interest to be able to predict unstable operation of the compressor in order to expand the range of mass flow and thus expand the operating range of the compressor.

In this thesis, an anti-surge controller is proposed using a close-coupled valve(CCV) and a state-of-the-art neural network(NN) Lyapunov function candidate for the Moore-Greitzer compressor. A CCV in combination with the Moore-Greitzer compressor can stabilize the system by modifying the characteristics of the compressor. Lyapunov control theory has shown useful in order to determine and construct a safe region for closed-loop dynamical systems. The objective of this thesis has been to gain an overview of the state-of-the-art regarding stability and robustness in nonlinear dynamical systems in order to maximize the safe level set that can be used as a safety certificate for a given nonlinear, closed-loop dynamical compressor system. To the best of the authors' knowledge, an anti-surge controller for the Moore-Greitzer compressor has never been implemented with the combination of Lyapunov control theory and NNs before.

Two control laws for the Moore-Greitzer compressor in series with a CCV is created using a NN Lyapunov function. The first control law resulted in a global asymptotically stable equilibrium beyond the original surge line. The second control law focused on minimization of the pressure drop across the CCV, and the equilibrium point was shown to be locally asymptotically stable. A training algorithm that iteratively "grows" an estimate of the largest safe region in the state space was used to show how well the different Lyapunov functions scored. The NN Lyapunov function covered approximately 81% of the estimated region of attraction. These results were compared to a traditional Lyapunov function candidate.

Keywords: *Lyapunov control theory, neural networks, region of attraction, Moore-Greitzer compressor, anti-surge controller, global asymptotic stability.*

Sammendrag

Tradisjonelt ville man håndtert surge i kompressorsystemer ved å operere langt fra "the surge line". Ved å gjøre dette vil systemet være stabilt i åpen sløyfe og man unngår dermed at surge oppstår. Selv om dette vil sikre integriteten til systemet, vil det også begrense operasjonsområdet til kompressoren. Hvis surge skulle oppstå vil det redusere ytelsen til kompressoren, eller i verste fall resultere i at den blir ødelagt.

Målet for denne mastergradsoppgaven har vært å utvikle en anti-surge kontroller for Moore-Greitzer kompressoren kombinert med en CCV, med hensikt å utvide operasjonsområdet. Anti-surge kontrolleren baserer seg på en kombinasjon av Lyapunov kontroll teori og neurale nettverk. Lyapunov kontroll teori har vist seg å være nyttig for både å finne og lage stabile attraksjonsområder for dynamiske systemer i lukket sløyfe, mens neurale nettverk er gode til å lære seg ulineære systemer.

I denne avhandlingen har to kontroll-lover for Moore-Greitzer kompressoren, som baserer seg på en neural nettverk Lyapunov funksjon, blitt utviklet. Den første kontroll-loven resulterte i et globalt asymptotisk stabilt likevektspunkt. Den andre kontroll-loven har fokus på minimering av trykktapet over CCVen og resulterte i et lokalt asymptotisk stabilt likevektspunkt. En treningsalgoritme som iterativt bygger et estimat av det faktiske attraksjonsomrdet er benyttet for å visualisere prestasjonen til Lyapunov funksjonen. Lyapunov funksjonen basert på neurale nettverk dekket tilnærmet 81% av det estimerte attraksjonsområdet. Resultatet ble sammenlignet med en tradisjonell Lyapunov funksjon.

Contents

Abbreviations

AI	=	Artificial Intelligence
AS	=	Asymptotically Stable
CCV	=	Close-Coupled Valve
DL	=	Deep Learning
GAS	=	Globally Asymptotically Stable
GUAS	=	Global Uniform Asymptotic Stability
LMI	=	Linear Matrix Inequality
LP	=	Linear Programming
LQR	=	Linear Quadratic Regulator
ML	=	Machine Learning
NN	=	Neural Network
ROA	=	Region of Attraction
SoS	=	Sum-of-Squares

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Compressor surge represents undesired oscillations in mass flow and pressure, and can cause a reduction in performance or even damage the compressor. It occurs if the flow is throttled beyond the surge line; therefore, the cautious way to handle surge is to ensure that the system is stable in open-loop. As this will limit the range of mass flow for which the compressor can be used, it is preferred to implement an anti-surge controller in order to stabilize the equilibrium beyond the original surge line. As for any safety-critical application, it is unacceptable for a safe level set to contain anything but safe states, as this can damage the application or its surroundings. It is therefore critical that the range of mass flow is never expanded at the expense of safety.

1.2 Problem formulation

The main objective of this master thesis is to design a state-of-the-art anti-surge controller for the Moore-Greitzer compressor in combination with a close-coupled valve. The focus will be maximization of the safe level set $\mathcal{V}$ in order to expand the operating range for the compressor system. The CCV can stabilize the equilibrium by modifying the characteristics of the compressor but at the expense of a pressure drop over the valve. The Lyapunov direct method will be used to determine the stability of the equilibrium for the given compressor parameters. However, there is no general way to find a Lyapunov function candidate, and it is even more challenging to find a Lyapunov function candidate that can ensure the maximization of the safe level set. Therefore, in this thesis, a neural network Lyapunov function candidate is proposed in order to create the anti-surge controller with a focus on maximization of

$\mathcal{V}_\theta$. Dynamical systems can be complicated to model accurately because of all of the parameters involved, such as the mechanics of the system and its surroundings (Gale et al., 2013). While traditional Lyapunov function candidates have restrictions on the dynamics, which can lead to a mismatch between the proposed safe level set and the largest safe level set, NNs have shown to be great nonlinear learners, and can be used to adapt the Lyapunov function candidate to the shape of the largest safe region in the state space.

Along with the main objective, an evaluation of the performance of the code required to solve the NN Lyapunov function (Richards et al., 2018) will be given. The work will be implemented with Python and the toolbox SOSTOOLS in MATLAB.

1.3 Objective and Research Questions

This work will have two primary objectives, respectively:

Objective 1: Construction of Safe Level Sets

To gain an overview over state-of-the-art regarding the construction of safe level sets for closed-loop dynamical systems in general and design an anti-surge controller based on NNs for the Moore-Greitzer compressor with focus on maximization of its safe level set in particular.

Objective 2: Evaluation

Evaluate the performance of the code required to create a Neural Network Lyapunov function (Richards et al., 2018) for a nonlinear system more complicated than the inverted pendulum system in the original code.

Along with the two objectives of this thesis, two research questions regarding stability analysis and safe level set maximization will be investigated:

Research Question 1: Lyapunov Function Candidate

Is it possible to find a suitable Lyapunov function candidate for the Moore-Greitzer compressor system?

Research Question 2: Safe Level Set Maximization

Is it possible to maximize the region of the state-space where a known policy can be applied without necessarily knowing the true region of attraction beforehand?

1.4 Outline

The thesis is organized as follows. The reader interested in an extensive literature study regarding nonlinear, closed-loop dynamical systems in general and compressor surge control in particular, as well as relevant theory, can read Chapters 2 and 3, respectively. Chapter 4 thoroughly describes the implementation procedure of the NN Lyapunov function candidate, along with the Greitzer-Moore compressor system. The results of the NN Lyapunov function candidate for the Greitzer-Moore compressor system are presented in Chapter 5, in addition to a traditional Lyapunov function candidate for comparison. Chapter 6 contains the discussion regarding all results and the restrictions of the algorithm. Chapter 7 and 8 describes suggestions for future work and concluding remarks, respectively. Finally, an appendix with relevant contents regarding Lyapunov control theory and the implementation of anti-surge controllers can be found in the end part of this thesis.

Chapter 2

Related Work

In this Chapter, a literature study regarding both feedback control and artificial intelligence in closed loop, in addition to compressor surge control, will be given. This Chapter is organized as follows: Section 2.1 focuses on approaches to estimate and maximize the region of attraction for nonlinear dynamical systems. In Section 2.2, the implementation of different anti-surge controllers is addressed, with a particular focus on compressors in series with a CCV.

2.1 Feedback control and AI in closed loop

Fontaine et al. (2004) propose a nonlinear control law for an axial flow compressor with a bleed ring. The objective of the article is to stabilize the equilibrium point and maximize the region of attraction. It is achieved with a combination of Lyapunov control theory, L_gV controller, and the Linear Quadratic Regulator method. Barkhordari et al. (2008) propose a method of enlarging the region of attraction of nonlinear systems by applying an input-output liberalization approach to the system. A generalized eigenvalue problem is created which results in a controller that stabilizes the system and, at the same time, maximizes the ROA.

Henrion and Korda (2013) present a method for computing the region of attraction as a convex infinite-dimensional linear programming(LP) formulation, where the LP is solved with a hierarchy of convex finite-dimensional linear matrix inequalities(LMIs). The authors argue that the method is easy to apply as no additional data is required except the problem description. Another strength of the approach is that LMIs convergence can be theoretically guaranteed. The method is computed on four different systems, to show the flexibility of the method as an alternative approach to Lyapunov control theory.

Berkenkamp et al. (2016) consider an approach that learns the region of attraction from experiments on real systems, with a high probability that the safe region only contains safe states, an essential criterion in safety-critical applications. This approach addresses the problem of errors introduced by the model. The ROA can be very different for a model of the system and the real system due to model errors. Additionally, they implement an algorithm based on Lyapunov control theory with the focus of expanding the region of attraction. Their approach consists of using safe Bayesian optimization to learn from experiments where the ROA has been estimated. By doing so, one can learn about the real dynamics and at the same time, ensure that the safety requirements are not violated. The region of attraction is estimated given a known control policy which is based on experiments on the real system. In order to achieve their goal, some assumptions have been made, which will limit the practical use of the algorithm. However, it is an essential theoretical foundation for estimating the true region of attraction for non-linear dynamical systems.

Richards et al. (2018) present a neural network Lyapunov function and a training algorithm for learning accurate safety certificates for nonlinear, closed-loop dynamical systems. Lyapunov control theory is used to provide a safety certificate for identifying safe level sets by proving that the system is stable. The authors combine Lyapunov control theory with NNs, in order to find not just fulfill the criteria of a Lyapunov function, but to maximize the region of the state-space where it is possible to apply a known policy without necessarily knowing the true region of attraction beforehand. With this combination, the authors construct a NN Lyapunov candidate that always inherently yields a provable safety certificate. The algorithm's goal is to, given a known policy, find the largest safe region by adopting the candidate to the shape of the dynamical system's trajectories via classification of states as safe or unsafe. A Sum-of-Squares Lyapunov function candidate is used as a starting point to construct a NN Lyapunov candidate. Finally, the NN Lyapunov function is tested on an inverted pendulum system and the result is compared to more traditional Lyapunov functions. For the inverted pendulum system, the NN Lyapunov function performs much better than the alternative Lyapunov functions.

2.2 Compressor Surge Control

Simon and Valavani (1991) present the theoretical background for creating an anti-surge controller based on Lyapunov control theory, which directly addresses the nonlinear nature of the compressor characteristics. Gravdahl and Egeland (1997) develop an anti-surge controller for a close-coupled valve in series with a compressor in order to prevent the Moore-Greizer model from going into an unstable mode of operation. The compressor surge happens if the compression system is operated below the surge line, and this can cause a reduction in performance or even damage the compressor. The proposed anti-surge controller is implemented by using backstepping, which uses feedback from the mass flow, to derive the control law for the CCV and as a result, ensuring global uniform asymptotic stability(GUAS) beyond the original surge line. It is developed two controllers with different areas of application. The first one is developed for situations where there are no disturbances or only pressure disturbances. The controller only requires an upper bound on the slope of the compressor characteristics. The second controller has its application for cases of both pressure- and mass flow disturbances. As the second controller is more complicated than the first one, it requires the B-parameter to be known, along with the requirements from the first controller. As a result, the model is stabilized beyond the original surge line.

Liaw et al. (2002) investigate surge control in compression systems with uncertain characteristics. System robustness is ensured with Lyapunov control theory, and asymptotically stability of the equilibrium point is proven. Furthermore, the authors argue that since the approach strongly relies on the knowledge of the operating point, a washout filter feedback controller is implemented in order to postpone and restrain the occurrence of Hopf bifurcation. The controller does not require explicit knowledge of the system equilibrium points.

Nieuwenhuizen et al. (2009) addresses the correspondence between the Van Der Pol Equation and the Greitzer Model. As the authors' state, the identification of model parameters can be difficult with the Greitzer model, and often a priori knowledge is combined with a tuning process. With the Van Der Pol equation more general analytically approximations can be applied. In order to have a fair comparison, a coordinate transformation was performed on the Greitzer model so that the position and scaling of the limit cycles would be similar for the two models. The research showed a linear dependency between the stability parameter and the period time for large values of the parameter indicating the non-linearity and strength of damping. It is concluded that it is a promising first step to map the similarity between the two models as the presented systems almost have the same structure, but that further investigation is needed.

Backi et al. (2013) develop an anti-surge controller based on Lyapunov control theory, and provides a full state observer with local stability result for the Greitzer compressor model. Backi et al. (2016) propose an anti-surge controller based on feedback linearization for a close-coupled valve in a compression system. In order to stabilize surge in a compression system, the feedback linearization methodology is implemented by showing that the system is feedback linearizable, and then ensuring absolute stability by using circle criterion analysis. With their method, the authors can show with robustness analysis that the system can be stabilized and surge can be avoided. It is important to note that the controller implemented is not able to stabilize the system for significant feedback gains or constrained input signals. Moreover, it is assumed that the Greitzer stability parameter B and the throttle gain γ are known, which limits the robustness analysis.

Chapter 3

Background Theory

In this Chapter, a basic introduction to compressor surge control and the technologies needed in order to implement a NN Lyapunov function candidate for compressor surge control will be given. This Chapter is organized as follows: In Section 3.1, an introduction to compressor surge control is given. A basic introduction to nonlinear systems with a focus on Lyapunov control theory is presented in Section 3.2. Section 3.3 focuses on artificial intelligence in general and NNs in particular. In Section 3.4 and 3.5, a basic introduction to the sum-of-squares approach and linear quadratic regulators is given, respectively. Finally, in section 3.6 the further usage of the technologies presented in the above sections is presented.

3.1 Compressor Surge Control

A compressor is a mechanical device that increases the pressure of a compressible medium. Ferguson (1963) defines the function of a compressor as:

> *It is the function of a compressor to raise the pressure of a specified mass flow of gas by a prescribed amount using the minimum power input.*

The performance of a compressor depends on the relationship between the mass flow through the compressor and the pressure rise over the compressor. This relationship is illustrated in Figure 3.1, where the surge line is where the system goes into an unstable mode of operation, the stonewall line denotes the limitation of the capacity of the compressor, and the load line is the pressure requirements of the system. The compressor characteristics denote the steady-state pressure rise achieved as a function of the mass flow and, finally, the intersection point between the compressor characteristics and the load line is the steady-state operating point of the compressor (Nieuwenhuizen et al., 2009; Nieuwenhuizen, 2008).

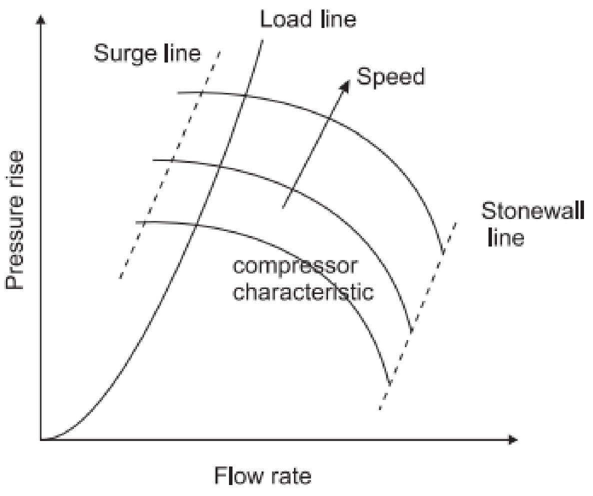


Figure 3.1: *Schematic representation of a compressor map, adopted from Nieuwenhuizen et al. (2009); Nieuwenhuizen (2008). The surge line is where the system goes into an unstable mode of operation, the stonewall line denotes the limitation of the capacity of the compressor, and the load line is the pressure requirements of the system. The compressor characteristics denote the steady-state pressure rise achieved as a function of the mass flow and, finally, the intersection point between the compressor characteristics and the load line is the steady-state operating point of the compressor.*

Compressor surge is a sort of aerodynamic instability which represents undesired oscillations in mass flow and pressure, and it is characterized by a limit cycle in the compressor characteristics. The phenomenon occurs if a compressor system is operated beyond the surge line, where the system goes into an unstable mode of operation. If that happens, it will cause a reduction in performance, but it can also damage the compressor due to high vibrational loads; therefore, it is essential to understand how to avoid surge and address the compressors. Surge can be divided into mild/classic surge and deep surge. Mild/classic surge is characterized by oscillations in both pressure and flow in the compressor system, and in deep surge, the amplitude of the mass flow oscillations is so comprehensive that flow reversal occurs in the compression system (Gravdahl and Egeland, 2012).

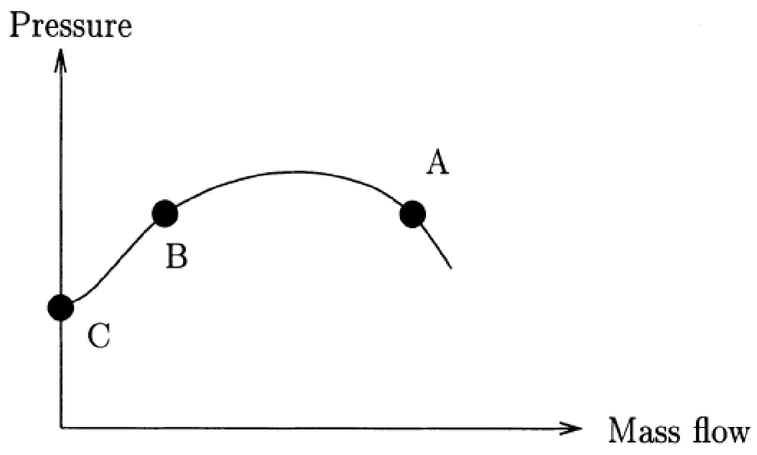


Figure 3.2: *Compressor characteristics, adopted from Gravdahl and Egeland (1997). If the compressor operates at point A, and the mass flow is decreased, the pressure will rise and the mass flow will increase. In this case, the system is stable. If the system operates at point B and the mass flow decreases, so will the pressure, resulting in the mass flow decreasing even more and the operating point moving further to the left. When the pressure upstream of the throttle falls below the compressor delivery pressure in point C, the mass flow will begin to increase up to point B, where the cycle repeats itself.*

In order to understand how surge works, consider Figure 3.2. If the compressor operates at point A, and the mass flow for some reason decreased, then the pressure will rise, forcing the mass flow to increase again; thus, if the compressor operates at point A, the system is self-compensating, and surge will be avoided. However, consider a system that operates at point B. In this case, if the mass flow decreases, so will the pressure, resulting in the mass flow decreasing even more and the operating point moving further to the left. When the pressure upstream of the throttle falls below the compressor delivery pressure in point C, the mass flow will begin to increase up to point B, where the cycle repeats itself. As a result of this observation, in order to stabilize the system, it is crucial that the operating point is in a decreasing slope, and not an increasing slope.

If surge occurred, the noise level would increase, and the piping around the compressor could begin to vibrate. Moreover, it would influence the chemical process connected to the compression system as both the mechanical and thermal load correlated with the surge could damage the system. An example of a system that is first in a stable state, and then goes beyond the surge line and becomes unstable can be seen in Figure 3.3. As can be seen, the oscillations can grow large. One of the problems with surge avoidance is that it limits the range of mass flow for which the compressor can be used, which is why there has been much research on the design of anti-surge controllers. The model that has shown the be most popular is the Greitzer model. The Greitzer model simplifies the compressor system to a plenum volume with an inlet duct from the compressor and an outlet duct to the throttle valve. For more information about the work related to the design of anti-surge controllers, the reader is referred to Chapter 2.

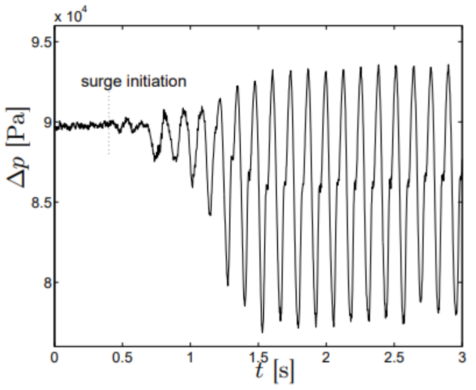


Figure 3.3: *Example of pressure oscillations in surge, adopted from Nieuwenhuizen et al. (2009); Nieuwenhuizen (2008). An example of a system that is first in a stable state, and then goes beyond the surge line and becomes unstable.*

The cautious way of dealing with surge, is to simply avoid it, which is achieved by operating far from the line where the system goes into an unstable mode of operation. Considering Figure 3.2, this is operating point A. It will ensure both the mechanical integrity of the machines and the safety, but it also limits the range of mass flow for which the compressor can be used. An alternative method is to implement active surge control, which will stabilize an extended region of operation for the compressor so that the productivity will be higher and the operation will be safer (Yoon et al., 2012). In this case, feedback is used to stabilize the unstable region left of the surge line.

One way to avoid surge is to add a close-coupled valve (CCV) in combination with the Greitzer compressor, as it can stabilize the system by modifying the characteristics of the compressor. The CCV can stabilize the equilibrium beyond the original surge line, at the expense of a pressure drop over the valve. As the term close-coupled implies, there is no significant mass storage between the compressor outlet and the valve as the distance between them is too small (Simon and Valavani, 1991). A figure of the schematic representation of the system is shown in Figure 3.4.

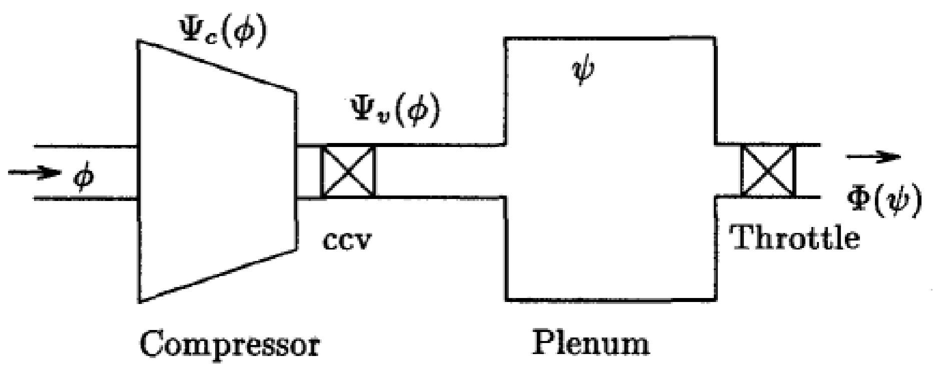


Figure 3.4: *Schematic representation of a compressor in combination with a close-coupled valve, adopted from Gravdahl and Egeland (1997).* $\Psi_c(\Phi)$ *and* $\Psi_v(\Phi)$ *are the compressor pressure rise and valve pressure drop, respectively, and* Φ *is the axial mass flow coefficient. The pressure rise over the compressor is the sum of the pressure rise over the compressor and the pressure drop over the valve. There is no significant mass storage between the compressor outlet and the CCV.*

3.2 Nonlinear Systems

This section is inspired by Khalil (2015).

Control theory is divided into two branches: systems that satisfy the superposition principle, and those that defy it. The superposition principle can be defined by two properties: additivity and homogeneity (Khalil, 2015). Additivity can be defined with the following mathematical equation

Definition: Additivity

$$F(x_1 + x_2) = F(x_1) + F(x_2)$$

and homogeneity says that the output is always directly proportional to the input and is defined as follows

Definition: Homogeneity

$$F(ax) = aF(x)$$

for a scalar a. Linear algebra is an active field with many applications in engineering physics, and a system is linear if it satisfies the superposition principle. The linear state-space model takes the form

$$\begin{aligned} \dot{\mathbf{x}} &= A(t)\mathbf{x} + B(t)\mathbf{u} \\ \mathbf{y} &= C(t)\mathbf{x} + D(t)\mathbf{u} \end{aligned} \tag{3.1}$$

where

$\mathbf{x}$ is the state vector,	$\mathbf{x} \in \mathbb{R}^n$;
$\mathbf{y}$ is the output vector,	$\mathbf{y} \in \mathbb{R}^q$;
$\mathbf{u}$ is the input/control vector,	$\mathbf{u} \in \mathbb{R}^p$;
A(t) is the state matrix,	$\mathbf{dim[A(t)]} = n \times n$
B(t) is the input matrix,	$\mathbf{dim[B(t)]} = n \times p$
C(t) is the output matrix,	$\mathbf{dim[C(t)]} = q \times n$
D(t) is the feedforward matrix,	$\mathbf{dim[A(t)]} = q \times p$

In this case, the matrices are time-variant, but they can also be time-invariant.

Nonlinear systems are systems that defy the superposition principle. In order to mathematically describe and solve these systems, more advanced mathematics has to be performed. There is no doubt that linear systems are much easier to solve and that there are powerful tools for solving linear mathematics. If possible, it is beneficial first to linearize a nonlinear system about some operating point in order to learn as much as possible from the linearization concerning the behavior of the system. However, there are two limitations of linearization. First, linearization cannot predict the global behavior of the state-space system. The reason for this is that the linearization is an approximation in the neighborhood of an operating point, and it can only predict the behavior near that point, and not the behavior far from it. Secondly, the dynamics of nonlinear systems are much more complex than the dynamics of linear systems. Consequently, linearization does not fully capture the behavior of a nonlinear system.

A nonlinear system can be described by

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(t, x, u) \\ \mathbf{y} &= \mathbf{h}(t, x, u) \end{aligned} \tag{3.2}$$

Nonlinear control theory applies to real-world systems. As mentioned earlier, linear systems are much easier to solve, and several methods have been developed in order to solve them. Some examples for solving linear time-invariant systems are Laplace Transform, Fourier transform, Z transform, Bode plot, and Nyquist stability criterion. These are mathematical techniques of high generality. However, the mathematical techniques developed to handle nonlinear systems are much less general and only apply to specific categories of systems. One of these mathematical techniques is called Lyapunov stability.

3.2.1 Lyapunov Stability

3.2.1.1 Lyapunov Indirect Method

Let $x = 0$ be an equilibrium point for

$$\dot{x} = f(x) \qquad f : \mathbb{D} \Rightarrow \mathbb{R}^n \quad \text{is } C^1$$

The first step of Lyapunov indirect method is to linearize the system about $x = 0$, $\dot{x} = Ax$. A is the state matrix and is defined as

$$A = \frac{\partial f}{\partial x}|_{x=0} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}_{|x=0} \tag{3.3}$$

The next step is to find the eigenvalues, $\lambda_1(A)$, ..., $\lambda_n(A)$, of the system. In order to say if the system is locally asymptotically stable, unstable or if there is no conclusion, the eigenvalues have to be evaluated. If
$\forall i \quad Re(\lambda_i) < 0 \qquad \Rightarrow x = 0$ is locally asymptotically stable.
However, if
$\exists i \quad Re(\lambda_i) > 0 \qquad \Rightarrow x = 0$ is unstable.
One can not say whether the system is stable or unstable if

$\forall i \quad Re(\lambda_i) \leq 0$
$\exists i \quad Re(\lambda_i) = 0$
Lyapunov indirect method is easy to use, and can in many cases be very helpful in order to determine if a equilibrium point is locally stable or not. The linearization can only predict the behavior near the equilibrium point, and cannot predict the global behavior of the state-space system.

3.2.1.2 Lyapunov Direct Method

While Lyapunov indirect method only contains information about whether the equilibrium point is locally stable or not, the Lyapunov direct method can determine global stability. The Lyapunov function candidate V: $\mathbb{D} \Rightarrow \mathbb{R}$ is a continuously differentiable function. It is used to consider the energy of the trajectories, thinking that the trajectories have energy associated with them. The idea is that as the trajectories are approaching the equilibrium point, the trajectories are getting weaker, and they, therefore, have less energy. If all the trajectories associated with the equilibrium point is decreasing towards zero, they must all be heading towards the equilibrium point. Hence, the Lyapunov function is called the *energy function* because the function shows how the energy changes as one move along the trajectory. The energy function will be zero at the equilibrium point(x_e), $V(x_e) = V(0) = 0$.

Generally speaking, a function V is positive definite if the function is always greater than zero. Formally,

$$V(x) > 0 \qquad \textbf{(positive definite)} \tag{3.4}$$

The function is negative definite if it is always less than zero

$$V(x) < 0 \qquad \textbf{(negative definite)} \tag{3.5}$$

It is positive semi-definite if it is greater than or equal to zero

$$V(x) \geq 0 \qquad \textbf{(positive semi-definite)} \tag{3.6}$$

Finally, it is negative semi-definite if it is less than or equal to zero

$$V(x) \leq 0 \qquad \textbf{(negative semi-definite)} \tag{3.7}$$

With this in mind, a formal definition of the Lyapunov function is given below.

Definition: Lyapunov function

V is a Lyapunov function for $x = 0$ iff
i) V is C^1 (continuously differentiable)
ii) $V(0) = 0$
$V(x) > 0$ in $\mathbb{D}\backslash\{0\}$
iii) $\dot{V}(0) = 0$
$\dot{V}(x) \leq 0$ in $\mathbb{D}\backslash\{0\}$
If, morover,
$\dot{V}(x) < 0$ in $\mathbb{D}\backslash\{0\}$
then V is a strict Lyapunov function for $x = 0$.

How to apply the method:

1. When using Lyapunov direct method, the first thing to do is to choose a Lyapunov function candidate V(x).
2. The next step is to determine if V(x) is a Lyapunov function or a strict Lyapunov function for the equilibrium point. This is done by proving the criteria for the definition of the Lyapunov function. The function is a Lyapunov function(and not a Lyapunov function candidate) only if V(x) meets these criteria.
3. If the answer is yes then the equilibrium point is stable or asymptotically stable. However, if the answer is no, the process starts at the beginning again, and a new Lyapunov function candidate is chosen.

As shown, if the Lyapunov function candidate is not a Lyapunov function, the process will start over. Usually, one cannot tell if the equilibrium point is unstable or not, only that with the chosen Lyapunov function candidate, it is not possible to tell if the equilibrium point is stable. However, if the Lyapunov function candidate shows that the derivative is positive definite, the system is in fact, unstable. It does not matter how many candidates are tested: the system tends to go to infinity.

The Lyapunov direct method can determine if an equilibrium point is globally stable or not. For an equilibrium point to be globally asymptotically stable, there must:

1. $\exists$ strict Lyapunov function $V : \mathbb{R}^n \Rightarrow \mathbb{R}$ for $x = 0$.
2. V is radially unbounded.

If these two conditions are met, $x = 0$ is globally asymptotically stable.

Definition: Radially unbounded

V(x) is radially unbounded iff
$$\|x\| \to \infty \Rightarrow V(x) \to \infty$$

With this method, it is also possible to prove if an equilibrium point is exponentially and globally exponentially stable. For the point to be exponentially stable, there must exist constants $a, k_1, k_2, k_3 > 0$ such that

1. V is C^1.
2. $k_1\|x\|^a \leq V(x) \leq k_2\|x\|^a \qquad \forall x \in \mathbb{D}$.
3. $\dot{V}(x) \leq k_3\|x\|^a \qquad \forall x \in \mathbb{D}$

If these conditions are satisfied, $x = 0$ is exponentially stable. Moreover, if the conditions above is satisfied with $\mathbb{D} = \mathbb{R}^n$, the equilibrium point is *globally exponentially stable.*

Lyapunov direct method is great in many ways, it gives global results, and the method is general. The downside of the Lyapunov direct method is that there is no general way to find V(x), and this can be a very tricky and time-consuming process.

3.3 Artificial Intelligence

Artificial intelligence (AI) is a branch of computer science that addresses problems requiring human-like reasoning and intelligence, and AI techniques have proven to be very useful as alternative approaches to traditional techniques (Shteimberg et al., 2012).

Definition: Artificial Intelligence

The study of how to make computers do things at which, at the moment, people do better.
(Rich and Knight, 1991)

Artificial intelligence is the overall definition, and it contains several subfields. In this section, machine learning and deep learning will be discussed in general, and neural networks in particular.

3.3.1 Machine Learning

Definition: Machine Learning

A computer program is said to learn from experience E with respect to some task T and some performance P, if its performance on T, as measured by P, improves with experience E.
(Mitchell, 1997)

Machine Learning (ML) is about learning to do better in the future based on what was experienced in the past. The characteristic of ML is its ability to learn from data rather than being programmed to follow a predefined set of rules. The control approaches presented later in this thesis will be based on machine learning. ML is usually categorized into four main approaches: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. The approaches are different, and they intend to solve different problems.

In supervised learning, given a training set of N input-output pairs $(x_1, y_1), (x_2, y_2)$,...,(x_N, y_N), where each y_i was generated by an unknown function $y = f(x)$. Here x_i is the feature vector of the i-th example and y_i is its label. The task of supervised learning is to find a function h that approximates the true function f (Russell and Norvig, 2016).

In unsupervised learning, patterns are learned from the input even though no explicit feedback is supplied. The training data is unlabeled, and the system has to learn by it self (Russell and Norvig, 2016).

With semi-supervised learning, the training data is partly labeled just as the input-output pairs in supervised learning, but a significant amount is also unlabeled. The combination of the unlabeled data and the labeled data in a training set can be viewed as a more realistic scenario than supervised learning and has an improvement in learning accuracy over unsupervised learning (Russell and Norvig, 2016).

In reinforcement learning the agent learns from rewards and punishments. The goal is to maximize its reward, given the current state of the environment. It is up to the agent to decide which action that lead to the reinforcement (Russell and Norvig, 2016).

3.3.2 Deep Learning

Deep learning (DL) is a specific kind of machine learning that scores high on problems with high-dimensional data and high-dimensional spaces (Goodfellow et al., 2016). With DL methods, the computer learns complicated concepts by building them out of simpler ones. Human engineers do not design these layers of features, but they are learned from data by using a learning procedure (LeCun et al., 2015).

Artificial Neural Networks(ANNs)

The human brain can be seen as a highly complex, nonlinear, and parallel computer: an information-processing system. The brain uses neurons to organize its structural components and can perform numerous tasks extremely fast (Haykin, 1994). ANNs had its motivation from how the brain works, and in its general form, a NN is a machine designed to model the way the brain performs a particular task or function of interest.

Definition: Neural Network

A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. Knowledge is acquired by the network from its environment through a learning process.
2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

(Haykin et al. (2009))

The learning algorithm performs the learning process, and the function is designed to modify the synaptic weights of the network in order to achieve the desired objective.

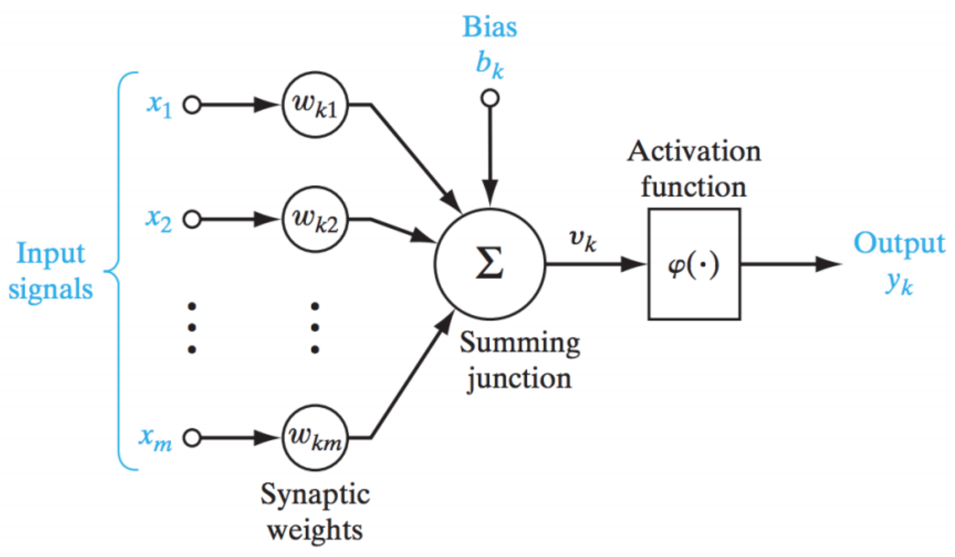


Figure 3.5: *Neuron model, adopted from Haykin (1994). The input signal x_j of synapse j, which is connected to a neuron k, is multiplied by the synaptic weight ω_{kj}. The product of this multiplication $u_k = x_j \cdot \omega_{kj}$ is then entering the summing junction where it is added with a bias b_k. The bias b_k can increase or decrease the net input of the activation function: determined by whether it is positive or negative. This sum v_k is the input to the activation function $\varphi(\cdot)$, a function which limits the amplitude of the output y_k of a neuron. The given output is used as an input to the next neuron, and so on.*

A neuron is an information processing unit, and it is the basic building block of a NN. It is build up by several elements: the synaptic weights, the summing junction, and the activation function (Haykin, 1994). The process can be described as follows: the input signal x_j of synapse j, which is connected to a neuron k, is multiplied by the synaptic weight ω_{kj}. The product of this multiplication $u_k = x_j \cdot \omega_{kj}$ is then entering the summing junction where it is added with a bias b_k. The bias b_k can increase or decrease the net input of the activation function: determined by whether it is positive or negative. This sum v_k is the input to the activation function $\varphi(\cdot)$, a function which limits the amplitude of the output y_k of a neuron. Finally, the given output is used as an input to the next neuron, and so on. An illustration of the process can be seen in Figure 3.5. The process can be described with the following mathematical equations:

$$u_k = \sum_{j=1}^{m} \omega_{kj} x_j \tag{3.8}$$

$$v_k = u_k + b_k \tag{3.9}$$

$$y_k = \varphi(u_k + b_k) = \varphi(v_k) \tag{3.10}$$

3.3.2.1 Types of activation functions

The *Threshold function* can be defined as:

$$\varphi(v) = \begin{cases} 1 & if\ v \geq 0 \\ 0 & if\ v < 0 \end{cases} \tag{3.11}$$

It is a boolean function, and can only take the values 1(if it is activated) or 0(not activated).
A *Sigmoid function* is often defined by the formula

$$\varphi(v) = \frac{1}{1 + exp(-av)} \tag{3.12}$$

where a is a slope parameter, which differ the slope. The function is bounded, differentiable, it is defined for all real input values and its derivative is positive semi-definite.
The *Hyperbloic tangent function(Tanh function)* is defined as:

$$\varphi(v) = tanh(v) = \frac{2}{1 + exp(-2v)} - 1 = 2 \cdot sigmoid(2v) - 1 \tag{3.13}$$

The tanh(·) function is a scaled sigmoid function, whereas the gradient is stronger for tanh(·) than sigmoid.
The *ReLU(rectified linear unit)* activation function can be defined as

$$\varphi(v) = max(0, v) \tag{3.14}$$

The ReLU activation function gives the output v if it is positive, and 0 otherwise. Since ReLu(·) does not activate input signal below zero, fewer neurons are activated, which improves the efficiency of the network; however, this introduces a problem known as *the dying ReLU problem.* For negative input v the ReLU will give a horizontal line equal zero, where the gradient can go towards 0. The problem is that the neurons that enter that state will stop to respond to variations in input, which will result in parts of the network becoming passive. One way of solving the dying ReLU problem is to use Leaky ReLU instead (Bhimra et al., 2019).

$$\varphi(v) = \begin{cases} v & if\ v > 0 \\ 0.01v & otherwise \end{cases} \tag{3.15}$$

Instead of the function being zero when $x < 0$, a leaky ReLU will instead have a small negative slope.

Which activation function to use, depends on the characteristics of the system. For example, if the characteristics of the function are known, it could be wise to choose an activation function that will approximate the function faster as this leads to a faster training process. However, if the nature of the function is unknown, ReLU might be the right choice, as it is a general approximator.

3.3.2.2 Feedforward Neural Network

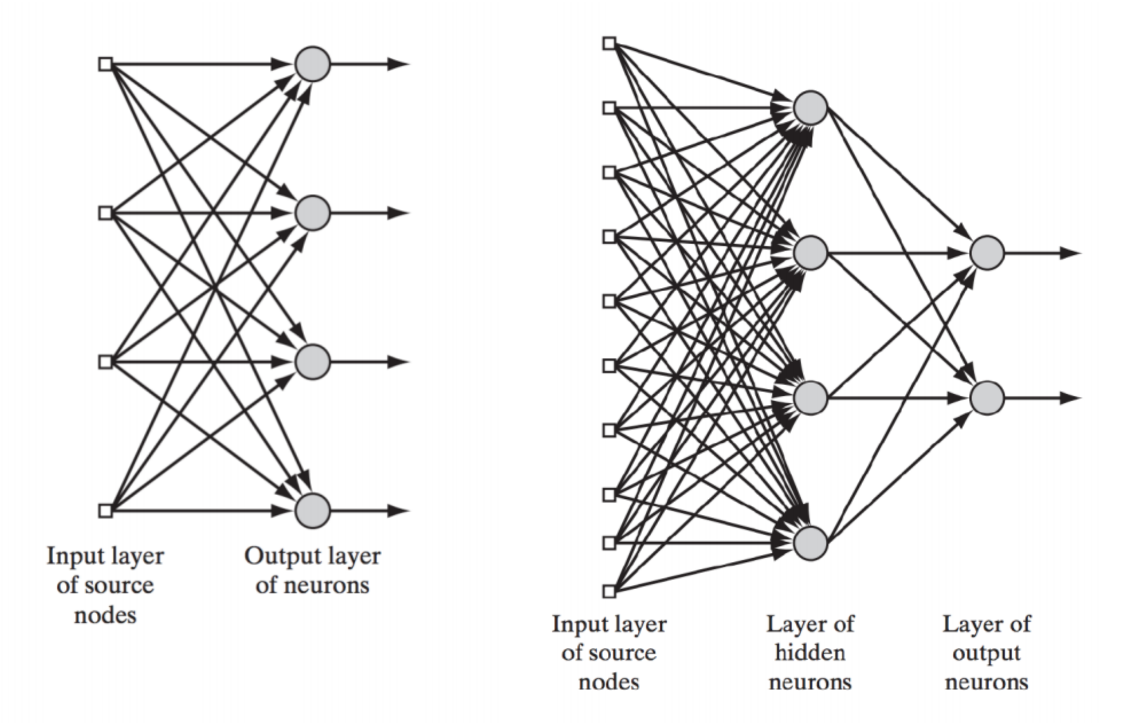


Figure 3.6: *(a) Single-layer feedforward (b) Multilayer feedforward, adopted from Haykin (1994). A single-layer feedforwad NN only has input layers and output layers. A multilayer feedforward NN has layers of hidden neurons, in addition to the input and output layers. The number of layers in the input layers equals the number of input variables to the system, and the number of neurons in the output layers is the number of outputs associated with each input. Hidden layers are used to solve nonlinear functions. There is no feedback in either of the layers.*

The characteristics of a feedforward NN are that the data travels in one direction. More precisely, this means that the connection between the neurons does not form a cycle; therefore, there is no feedback in the system. The feedforward NN may or may not have hidden layers and can be classified as a *single-layer feedforward neural network* or a *multilayer feedforward neural network.* A single-layer feedforwad NN only has input layers and output layers. The number of source nodes in the input layers equals the number of input variables to the system, and the number of neurons in the output layers is the number of outputs associated with each input. A multilayer feedforward NN has layers of hidden neurons, in addition to the input and output layers. Hidden layers are used to solve nonlinear functions. An illustration can be seen in Figure 3.6.

3.3.2.3 Back-Propagation

By introducing hidden layers to the network nonlinear functions can be solved. However, this introduces a new problem. With hidden layers, the training data cannot decide what values are assigned to the hidden neurons (Haykin, 1994). This can result in an error at the output. In order to solve this, it is possible to back-propagate the error from the output layer to the hidden layers. The concept is that a hidden layer j is responsible for some fraction of the error at the outputs which it connects. So the fraction of error is divided, depended on the strength of its connection between the hidden node and the output node, and propagated back the error.

3.4 Sum-of-Squares approach

Given a multivariate polynomial $p(x)$, the Sum-of-Squares (SoS) approach requires the existence of polynomials $f_1(x), ..., f_m(x)$ such that

$$p(x) = \sum_{i=1}^{m} f_1^2(x) \tag{3.16}$$

where $f(x) \geq 0$ for all $x \in \mathbb{R}^n$ (Prajna et al., 2002). This means that there must exist a positive semi-definite matrix Q with monomials $Z(x)$, such that

$$p(x) = Z^T(x)QZ(x). \tag{3.17}$$

SOS programs are solved by reformulating them as semi-definite programs (SDP). The toolbox SOSTOOLS (Prajna et al., 2002) in MATLAB first automates the conversion from SoS programs to SDPs, it then calls the SDP solver, and finally, converts the solution from the SDPs back to SOS programs. There are several SDP solvers, but in this thesis, the SeDuMi (Self Dual Minimization) solver in MATLAB is used.

SOSTOOLS has been frequently used to solve robustness related issues. The reason for this is that the toolbox combines dynamical system theory, real algebraic geometry, and semi-definite programming in order to provide a promising framework to handle the related issues. The method is based on Lyapunov control theory in order to ensure stability.

3.5 Linear Quadratic Regulators

Consider the following linear, time-invariant system

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + Bu(t), \qquad x(t_0) = x_0 \tag{3.18}$$

where $\mathbf{x}$ is the state space vector, $u(t)$ is the control input, A is the state matrix, B is the input matrix and x_0 is the initial condition of the states at time t_0 (Jin and Lin, 2012). Linear quadratic regulators (LQR) is used in optimal-control theory as a method for finding the state-feedback control input $u(t)$ defined within $[t_0, T]$, such that the quadratic cost function J_{LQR} is minimized, thus, the undesired deviations are minimized.

$$J_{LQR} = \int_{t_0}^{T} (\mathbf{x}^T q \mathbf{x} + u^T R u) dt \tag{3.19}$$

The quadratic cost function is defined in Equation 3.19, where the positive semi-definite matrix Q is the weight on the system states, and the positive definite matrix R is the weight on the control input. Given the control input

$$u(t) = -R^{-1}B^T P\mathbf{x}(t) = -\mathbf{K}\mathbf{x}(t), \tag{3.20}$$

which optimizes the quadratic cost function, the algebraic Riccati equation can be written as

$$A^T P + PA - PBR^{-1}B^T P = -Q \tag{3.21}$$

where, P is the unique positive definite solution to the Riccati equation and the LQR gain $\mathbf{K} = R^{-1}B^T P$.

3.6 Anti-Surge Controller for the Moore-Greitzer Compressor

In section 3.1 a general introduction to compressor surge was given, mostly to provide the reader with a basic understanding of what surge is and why it should be avoided. In Chapter 4, the implementation of an anti-surge controller for the Moore-Greitzer compressor will be presented. Lyapunov control theory, presented in Section 3.2, is used to ensure stability of the equilibrium point. The Lyapunov candidate is implemented with NNs in order to create an anti-surge controller with focus on maximization of the region of attraction for the Moore-Greitzer compressor. Therefore, a very general introduction to AI and NNs was given in Section 3.3. The SoS approach, presented in section 3.4, is used as a starting point for the construction of the NN Lyapunov candidate. The results will be presented in Chapter 5 and discussed in Chapter 6. Furthermore, the results of the NN Lyapunov function will be compared with a LQR Lyapunov function. A basic introduction to linear quadratic regulators is given in Section 3.5. This comparison is presented to show how a Lyapunov candidate based on NNs can expand its region of attraction compared to traditional candidates.

Chapter 4

Implementation

In this Chapter, the closed-loop dynamics of the Moore-Greitzer compressor system is derived and a NN Lyapunov function is implemented. This Chapter is organized as follows: in Section 4.1 the closed-loop dynamical model is derived and in Section 4.2 the implementation of the Lyapunov function is presented. Finally, information regarding the implementation of the NN Lyapunov Function for the Moore-Greitzer compressor system in Python can be found in Section 4.3.

4.1 The Moore-Greitzer compressor model

Consider the Moore-Greitzer compressor model in combination with a CCV given by the following equations:

$$\dot{\psi} = \frac{1}{B}\Big(\phi - \Phi(\psi)\Big) \tag{4.1}$$

$$\dot{\phi} = B\Big(\Psi_c(\phi) - \psi\Big) \tag{4.2}$$

where ϕ is the mass flow coefficient(annulus averaged, axial velocity divided by wheel speed (Moore and Greitzer, 1986)), ψ is the plenum pressure coefficient (pressure divided by density and the square of wheel speed (Moore and Greitzer, 1986)), $\Phi(\psi)$ is the throttle characteristics and the constant $B > 0$ is the Greitzer's B-parameter defined as

$$B = \frac{U}{2a_s}\sqrt{\frac{V_p}{A_c L_c}} \tag{4.3}$$

Here, U is the compressor blade tip speed, a_s is the speed of sound, A_c is the flow area, V_p is the plenum volume and L_c is the length of ducts and compressor. The characteristics of the compressor can be modeled as

$$\Psi_c(\phi) = \psi_{c0} + H\left[1 + \frac{3}{2}\big(\frac{\phi}{W} - 1\big) - \frac{1}{2}\big(\frac{\phi}{W} - 1\big)^3\right] \tag{4.4}$$

Here, the parameters $H > 0$ is the semi-height of cubic axisymmetric characteristics, $W > 0$ is the semi-width of cubic characteristics and $\psi_{c0} > 0$ is the shut-off value of axisymmetric characteristic (Moore and Greitzer, 1986).

$$\Phi(\psi) = \gamma\sqrt{\psi} \tag{4.5}$$

$$\Psi_v(\Phi) = \frac{1}{\gamma^2}\Phi^2 \tag{4.6}$$

Equation 4.5 states the throttle characteristics and Equation 4.6 is the CCV characteristics, where γ is the throttle gain and $\gamma > 0$ is proportional to the valve opening. Without loss of generality, the system is transformed such that the equilibrium point is at the origin, where

$$\begin{aligned}
&\hat{\phi} = \phi - \phi_0 \\
&\hat{\psi} = \psi - \psi_0 \\
&\hat{\Psi}_e(\hat{\phi}) = \Psi_e(\hat{\phi} + \phi_0) - \Psi_e(\phi_0) \\
&\hat{\Psi}_c(\hat{\phi}) = \Psi_c(\hat{\phi} + \phi_0) - \Psi_c(\phi_0) \\
&u = \hat{\Psi}_v(\hat{\phi}) = \Psi_v(\hat{\phi} + \phi_0) - \Psi_v(\phi_0) \\
&\hat{\Phi}(\hat{\psi}) = \Phi(\hat{\psi} + \psi_0) - \Phi(\psi_0)
\end{aligned} \tag{4.7}$$

The compressor is in equilibrium when $\dot{\phi} = \dot{\psi} = 0$. The system can now be seen as

$$\dot{\hat{\psi}} = \frac{1}{B}\Big(\hat{\phi} - \hat{\Phi}(\hat{\psi})\Big) \tag{4.8}$$

$$\dot{\hat{\phi}} = B\Big(\hat{\Psi}_c(\hat{\phi}) - \hat{\psi} - u\Big) \tag{4.9}$$

where

$$\hat{\Phi}(\hat{\psi}) = \gamma\sqrt{\hat{\psi} + \psi_0} - \gamma\sqrt{\psi_0} \tag{4.10}$$

$$\hat{\Psi}_c(\hat{\phi}) = -k_3\hat{\phi}^3 - k_2\hat{\phi}^2 - k_1\hat{\phi} \tag{4.11}$$

Equation 4.11 gives the compressor characteristics where $k_1 = \frac{3H\phi_0}{2W^2}(\frac{\phi_0}{W} - 2)$, $k_2 = \frac{3H}{2W^2}(\frac{\phi_0}{W} - 1)$ and $k_3 = \frac{H}{2W^3}$. u is the control variable, and represents the pressure drop over the CCV. The steady-state values of mass flow and plenum pressure are ϕ_0 and ψ_0, respectively. The state equations in this new local coordinate system will be used in the remainder of this thesis. A figure of the schematic representation of the system is shown in Figure 3.4. The CCV modifies the compressor characteristics by making the throttle line cross $\Psi_e(\hat{\phi})$ where the slope is negative, instead of where it earlier was a positive slope, this can be seen in Figure 4.1.

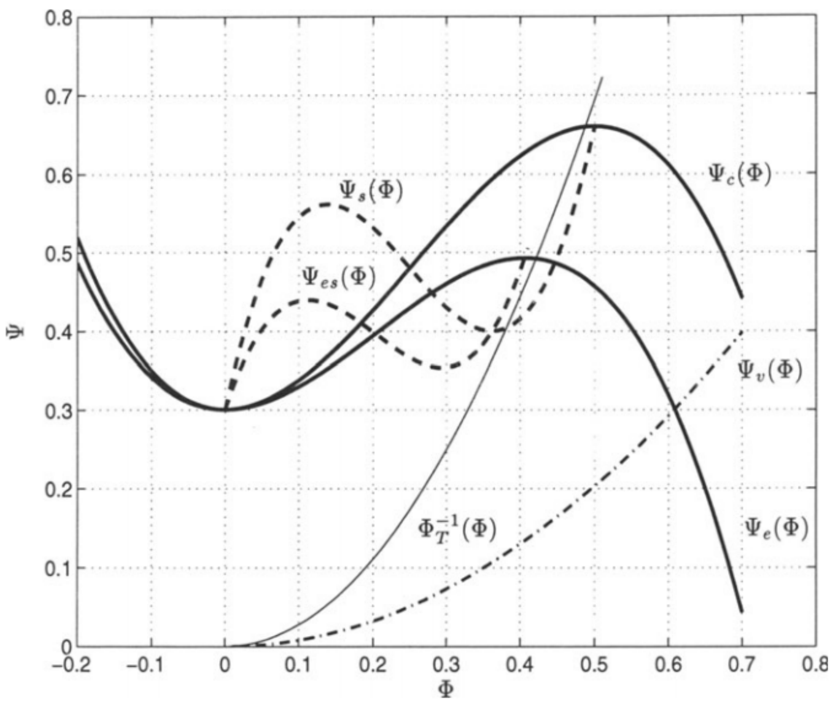


Figure 4.1: *Compressor and throttle characteristics, adopted from Gravdahl and Egeland (1997). $\Psi_c(\Phi)$ and $\Psi_v(\Phi)$ are the compressor pressure rise and valve pressure drop, respectively. $\Psi_e(\Phi)$ is the equivalent compressor characteristic and ${\Psi_T}^{-1}(\Phi)$ is the without the CCV. Since this study only concerns with pure surge control, stall in the system is not considered, and the stall characteristic $\Psi_{es}(\Phi)$ and $\Psi_s(\Phi)$ is not relevant. Without the CCV, the equilibrium of the system is at the intersection between the compressor characteristic $\Psi_c(\Phi)$ and the throttle characteristic ${\Psi_T}^{-1}(\Phi)$, and the intersection is at a point of positive slope. In this case, the equilibrium is unstable. By introducing the CCV, the throttle line $\Psi_v(\Phi)$ crosses the equivalent characteristic in an area of negative slope. This new equilibrium is stable.*

By defining $\hat{\psi} = x_1$, $\hat{\phi} = x_2$ and $\psi_0 = x_{1_0}$, the system given by Equation 4.8 and Equation 4.9 can be written as:

$$\begin{aligned}\dot{x}_1 &= \frac{1}{B}[x_2 - \gamma\big(\sqrt{x_1 + x_{1_0}} - \sqrt{x_{1_0}}\big)] \\ \dot{x}_2 &= B\big(-k_3x_2^3 - k_2x_2^2 - k_1x_2 - x_1 - u\big)\end{aligned} \tag{4.12}$$

where $f(\mathbf{x}) = \dot{\mathbf{x}}$ represents the dynamical system. The linear control law is defined as

$$u = -\mathbf{Kx} \tag{4.13}$$

where u is the pressure drop across the CCV and $\mathbf{K}$ is fixed to the LQR solution of the dynamics. The dynamics of the system is discretized with time step $\Delta t = 0.01$.

The system in Equation 4.12 can be written on state-space form as

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} \tag{4.14}$$

where

$$A = \begin{bmatrix} -\frac{\gamma}{2B\sqrt{x_{1_0}}} & \frac{1}{B} \\ -B & -k_1 B \end{bmatrix} \; and \; B = \begin{bmatrix} 0 \\ -B \end{bmatrix} \tag{4.15}$$

4.2 Neural Network Lyapunov Function Candidate

The following section describes the implementation of the code required to solve the Neural Network Lyapunov function and the algorithm used to iteratively grow an estimate of the true ROA, and is mainly created by Richards et al. (2018). However, significant understanding and modification were required to tailor the APIs to fit this thesis problem.

Consider the discrete-time, time-invariant, deterministic dynamical system of the form

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \tag{4.16}$$

where

t is the time step index,	$t \in \mathbb{N}$;
$\mathbf{x}_t$ is the state input at time step t,	$\mathbf{x}_t \in \chi \subset \mathbb{R}^d$
$\mathbf{u}_t$ is the control input at time step t,	$\mathbf{u}_t \in \mathcal{U} \subset \mathbb{R}^p$
π is the feedback policy,	$\pi : \chi \to \mathcal{U}$
$\mathbf{x}_{t+1} = f_\pi(\mathbf{x}_t)$ is the resulting closed-loop dynamical system,	$f_\pi(\mathbf{x}) = f(\mathbf{x}, \pi(\mathbf{x}))$

It is assumed that the policy π is given and that it is safe to use within a subset S_π of the state-space χ, where S_π is defined as the region of attraction for f_π. In this particular case, this means that every trajectory of f_π that begins at some $\mathbf{x} \in S_\pi$ remains in S_π and will converge to the equilibrium point $x_e \in S_\pi$, where $f_\pi(x_e) = x_e$, as time approaches infinity. Without loss of generality, it is assumed that the equilibrium point is at the origin, $x_e = 0$. From now on, S_π stands for the true largest ROA in χ under the policy π. The control policy π determines, given the current state, the appropriate control action that drives the system to some goal state, which in this case is the equilibrium point.

4.2.1 Construct safe level sets for a general $f_\pi(\mathbf{x}_t)$

In safety-critical applications, it is unacceptable for a safe region S_π to contain anything but safe states. The consequences could, in the worst case, be dramatic, which makes it critical to learn accurate safety certificates for nonlinear, closed-loop dynamical systems. In order to ensure safe learning, a safety certificate for a state must be verified before it is explored (Richards et al., 2018). As long as the trajectory lies within the region of attraction, the system can collect data during learning and always recover to a known safe point (Khalil, 2015).

One method to determine and construct a safe region S_π for the closed-loop dynamical system $\mathbf{x}_{t+1} = f_\pi(\mathbf{x}_t)$ is the Lyapunov Direct Method. The Lyapunov direct method can, in this particular case determine if an equilibrium point is not only stable but also globally asymptotically stable.

Theorem 4.1 ***(Lyapunov's stability theorem (Richards et al., 2018))*** *Suppose f_π is locally Lipschitz continuous and has an equilibrium point at $x_e = \boldsymbol{0}$. Let $V : \chi \to \mathbb{R}$ be locally Lipschitz continuous on χ. If there exists a set $\mathcal{D}_V \subseteq \chi$ containing $\boldsymbol{0}$ on which V is positive-definite and $\Delta V(\boldsymbol{x}) := V(f_\pi(\boldsymbol{x})) - V(\boldsymbol{x}) < 0$, $\forall \boldsymbol{x} \in \mathcal{D}_V \setminus \{\boldsymbol{0}\}$, then $x_e = \boldsymbol{0}$ is an asymptotically stable equilibrium point. In this case, V is known as a Lyapunov function for the closed-loop dynamics f_π and $\mathcal{D}_V$ is the Lyapunov decrease region for V.*

If the Lyapunov function candidate $V(\mathbf{x})$ fulfills the criteria of a strict Lyapunov function, then the equilibrium point $x_e = \mathbf{0}$ is asymptotically stable. One of the criteria for a strict Lyapunov function is that the derivative of $V(\mathbf{x})$ is negative definite, except at $\mathbf{0}$ where $\dot{V}(\mathbf{x}) = \mathbf{0}$. This can be difficult to certify throughout entire trajectories; therefore, it is easier to instead verify the *one-step decrease condition* $\Delta V(\mathbf{x}) = V(f_\pi(\mathbf{x})) - V(\mathbf{x}) < 0$ for every state $\mathbf{x}$ in the a level set of the Lyapunov function candidate V.

Corollary 4.1.1 ***(Safe level sets (Richards et al., 2018)):*** *Every level set $\mathcal{V}(c) := \{\boldsymbol{x} \mid V(\boldsymbol{x}) \leq c\}$, $c \in \mathbb{R}_{>0}$ contained within the decrease region $\mathcal{D}_V$ is invariant under f_π. That is, $f_\pi(\boldsymbol{x}) \in \mathcal{V}(c)$, $\forall \boldsymbol{x} \in \mathcal{V}(c)$. Furthermore, $lim_{t\to\infty} \boldsymbol{x}_t = \boldsymbol{0}$ for every $\boldsymbol{x}_t$ in these level sets, so each one is a ROA for f_π and $x_e = \boldsymbol{0}$.*

If trajectories start in a level set $\mathcal{V}(c)$ contained in the decrease region $\mathcal{D}_V$, they will remain in $\mathcal{V}(c)$ and converge to $x_e = \mathbf{0}$. It can be very difficult to verify that $\dot{V}(\mathbf{x})$ is less than zero throughout the entire continuous subset $\mathcal{D}_V \subseteq \chi$. With that said, a Lyapunov function candidate is a Lyapunov function $V(\mathbf{x})$ if it is continuously differentiable, $V(x_e) = \mathbf{0}$, $V(\mathbf{x})$ is positive definite, $\dot{V}(\mathbf{x}) = \mathbf{0}$ and $\dot{V}(\mathbf{x})$ is negative *semi-definite*. This means that the Lyapunov function $V(\mathbf{x})$ decreases monotonically along trajectories, implying that once a trajectory enters a level set, say given by $\mathcal{V}(\mathbf{x}) = c$, it can never leave the set $\mathcal{V}(c) := \{\mathbf{x} \in \mathbb{R}^n \mid V(\mathbf{x}) \leq c\}$. This is known as LaSalle's *invariance principle* (Khalil, 2015). If there exists a Lyapunov function whose derivative along the trajectories of the system is negative semi-definite, and it is possible to establish that no trajectory can stay identically at points where the derivative of the Lyapunov function is $\mathbf{0}$, except at the origin, then the origin is asymptotically stable.

4.2.2 Suitable Lyapunov function candidates

The downside with the Lyapunov direct method is that there is no general way to find a Lyapunov function $V(\mathbf{x})$, and it can, therefore, be very tricky. In control theory, there is a couple of common candidates such as Quadratic form and Sum-of-Squares(SoS).

The problem with both the Quadratic form and the SoS Lyapunov function is that their geometry might not fit the safe region S_π, so the region of attraction could be much more extensive than what is included.

SoS Lyapunov functions
The Sum-of-Squares approach enforces $V(\mathbf{x})$ to be a polynomial of the form $V(\mathbf{x}) = m(\mathbf{x})^T\mathbf{Q}m(\mathbf{x})$. Here, $m(\mathbf{x})$ is a vector of a priori fixed monomial features in element of $\mathbf{x}$ and $\mathbf{Q}$ is an unknown positive semi-definite matrix (Richards et al., 2018). As a result, the Lyapunov function candidate is a quadratic function on a monomial feature space, which accordingly allows Lyapunov functions to have shapes beyond simple ellipsoids. The problem with the SoS Lyapunov approach is that it requires polynomial dynamics, which can lead to a shape mismatch between the level set $\mathcal{V}(c)$ and the true largest region of attraction S_π. As the objective is to find the true ROA S_π and not just a part of it, this method comes up short. This is why it is very desirable to find a method that can do better for a more general closed-loop dynamical system $f_\pi(\mathbf{x}_t)$.

4.2.3 Neural Network Lyapunov Function

There are several ways to choose a Lyapunov function candidate, and many may fulfill the criteria of a Lyapunov function for the dynamical system $f_\pi(\mathbf{x}_t)$, and even though that is very good, it is not the entire objective. The objective is to maximize the region of the state-space χ where it is possible to apply the policy π without necessarily knowing the true ROA beforehand.

While the SoS Lyapunov function candidate does not ensure maximization of the safe level set, it can be used as a starting point to create a NN Lyapunov function candidate. The SoS Lyapunov function candidate $V(\mathbf{x}) = m(\mathbf{x})^T\mathbf{Q}m(\mathbf{x})$ is a Euclidean inner product on the transformed space $\mathcal{Y} = \{\phi(\mathbf{x}), \forall\mathbf{x} \in \chi\}$ with $\phi(\mathbf{x}) := \mathbf{Q}^{1/2}m(\mathbf{x})$ (Richards et al., 2018). However, as it has been proven difficult to determine the best choice of $m(\mathbf{x})$, and since, additionally, the Lyapunov candidate performance depends on the choice of $m(\mathbf{x})$, it is proposed a NN Lyapunov function candidate $V_\theta(\mathbf{x}) = \phi_\theta(\mathbf{x})^T\phi_\theta(\mathbf{x})$ to learn the essential features instead of choosing them manually. Here, $\phi_\theta : \mathbb{R}^d \to \mathbb{R}^D$ is a feed-forward NN with parameter vector θ.

In order to construct safe sets, the assumptions of the Lyapunov theory, Theorem 4.1, must be satisfied. It is also important to ensure that the NN Lyapunov function candidate is positive definite and satisfies the Lipschitz continuity requirements in Theorem 4.1. This is achieved with Theorem 4.2.

Theorem 4.2 ***(Lyapunov neural network (Richards et al., 2018)):*** *Consider $V_\theta(\boldsymbol{x}) = \phi_\theta(\boldsymbol{x})^T \phi_\theta(\boldsymbol{x})$ as a Lyapunov function candidate, where ϕ_θ is a feed-forward NN. Suppose, for each layer ℓ in ϕ_θ, the activation function φ_ℓ and weight matrix $\boldsymbol{W}_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ each have a trivial nullspace. Then ϕ_θ has a trivial nullspace, and V_θ is positive-definite with $V_\theta(\boldsymbol{0}) = 0$ and $V_\theta(\boldsymbol{x}) > 0, \forall \boldsymbol{x} \in \chi \backslash \{\boldsymbol{0}\}$. Furthermore, if φ_ℓ is Lipschitz continuous for each layer ℓ, then V_θ is locally Lipschitz continuous.*

Richards et al. (2018) provided a formal proof of Theorem 4.2, which states that if the NN ϕ_θ is Lipschitz continuous, then the NN Lyapunov function candidate V_θ is locally Lipschitz continuous, because of it being quadratic and differentiable with respect to the NN ϕ_θ. In order to show that the NN ϕ_θ is Lipschitz continuous, the activation function φ_ℓ must be Lipschitz continuous for each layer ℓ in ϕ_θ. Furthermore, it is ensured that the weight matrix $\mathbf{W}_\ell$ has a trivial nullspace by defining it as:

$$\mathbf{W}_\ell = \begin{bmatrix} \mathbf{G}_{\ell 1}^T \mathbf{G}_{\ell 1} + \varepsilon \mathbf{I}_{d_{\ell-1}} \\ \mathbf{G}_{\ell 2} \end{bmatrix} \tag{4.17}$$

where $\mathbf{G}_{\ell 1} \in \mathbb{R}^{q_\ell \times d_{\ell-1}}$ for some $q_\ell \in \mathbb{N}_{\geq 1}$, $\mathbf{G}_{\ell 2} \in \mathbb{R}^{(d_\ell - d_{\ell-1}) \times d_{\ell-1}}$, $\mathbf{I}_{d_{\ell-1}} \in \mathbb{R}^{d_{\ell-1} \times d_{\ell-1}}$ is the identity matrix and $\varepsilon \in \mathbb{R}_{>0}$ is a constant. As long as the constant ε is greater than zero, the weight matrix $\mathbf{W}_\ell$ has full rank and a trivial nullspace. q_ℓ is set to be the minimum integer required, in order to minimize the number of free parameters required by the NN Lyapunov function candidate V_θ. This means that each entry in $\mathbf{G}_{\ell 1}^T \mathbf{G}_{\ell 1} \in \mathbb{R}^{d_{\ell-1} \times d_{\ell-1}}$ is independent from the other entries. Given this definition of the weight matrix $\mathbf{W}_\ell$, the NN ϕ_θ contains the parameters $\theta := \{\mathbf{G}_{\ell 1}, \mathbf{G}_{\ell 2}\}_\ell$.

By making sure of zero bias terms, and with a particular choice of weight matrix and activation function, the NN Lyapunov function candidate $V_\theta(\cdot)$ is positive definite. Also, if $V_\theta(\cdot)$ is always positive definite, this implies that $V_\theta(c)$ is a safe level set if $\triangle V_\theta(\mathbf{x}) < 0$ holds throughout.

4.2.4 Learning via Classificaiton

Theorem 4.1 states the requirements for the Lyapunov function candidate to be a Lyapunov function V for the closed-loop dynamical system f_π, where $\mathcal{D}_V$ is the Lyapunov decrease region for V. Theorem 4.2 ensures that a NN Lyapunov function candidate is positive definite and satisfies the Lipschitz continuity requirements in Theorem 4.1. It can be very difficult to verify that $\dot{V}_\theta(\mathbf{x})$ is less than zero throughout the entire continuous subset $\mathcal{D}_{V_\theta} \subseteq \chi$, but with the *one-step decrease condition* $\Delta V_\theta(\mathbf{x}) = V_\theta(f_\pi(\mathbf{x})) - V_\theta(\mathbf{x}) < 0$ it is possible to identify safe level sets which are subsets of the true largest region of attraction S_π in χ under the policy π.

The last thing required in order for the NN Lyapunov function candidate V_θ to be a Lyapunov function for the closed-loop dynamical system f_π, is to implement a training algorithm. The training algorithm will adapt the parameters θ, so V_θ satisfies the one-step decrease condition for the largest region of attraction of D_{V_θ} as possible.

For the time being, in order to simplify the problem, it is assumed that the true largest ROA S_π is known. The next step will then be to optimize over the parameters data θ to maximize the volume $Vol(\cdot)$ of the safe level set $\mathcal{V}_\theta(c)$ of V_θ such that the one-step decrease condition is satisfied throughout the entire true ROA S_π; that is, for each state $\mathbf{x} \in S_\pi$. Mathematically, this problem can be written as:

$$\max_{\theta,c} Vol(\mathcal{V}_\theta(c) \cap S_\pi),\ s.t.\ \Delta V_\theta(\mathbf{x}),\ \forall \mathbf{x} \in \mathcal{V}_\theta(c) \tag{4.18}$$

Without loss of generality, it is fixed that $c = c_s$ where $c_s \in \mathbb{R}_{>0}$ as it is always possible to scale a Lyapunov function by a constant.

Furthermore, if, instead of looking at this as a level set, it is viewed as a decision boundary between safe and unsafe states, it becomes a classification problem. It is achieved by defining that $y = +1$ if it is a safe state, meaning that $\mathbf{x}$ is contained within S_π, and otherwise if the state is unsafe, it is defined as $y = -1$. It is crucial that the estimated safe set $\mathcal{V}(c_s)$ satisfies all the conditions of Theorem 4.1, and this is achieved by defining one rule and one constraint, respectively:

$$\hat{y}_\theta(\mathbf{x}) = sign(c_s - V_\theta(\mathbf{x})) \tag{4.19}$$

$$y = +1 \quad \Rightarrow \quad \Delta V_\theta(\mathbf{x}) < 0 \tag{4.20}$$

Equation 4.19 classifies the ground-truth label y as $+1$ or -1, and equation 4.20 ensures that the ground-truth label y is set to $+1$ only if the one-step decrease condition is fulfilled. The next step is to choose the NN parameters θ so that the rule of equation 4.19 and the constraint of equation 4.20 are upheld, and by doing so the decision boundary $V_\theta(\mathbf{x}) = c_s$ will delineate the boundary of S_π.

Consider the classification loss function $\ell(y, \mathbf{x}; \theta)$ which penalizes misclassification of the ground-truth label y at state $\mathbf{x}$ under rule 4.19 associated with θ. The loss function is chosen to be the perceptron loss function, an algorithm for supervised learning of binary classifiers. It is linear, and the penalty is higher far from the decision boundary $V_\theta(\mathbf{x}) = c_s$. In this particular case, with the signed distance $c_s \setminus V_\theta(\mathbf{x})$ from $V_\theta(\mathbf{x}) = c_s$, which sepatates S_π from the remaining state-space $\chi \setminus S_\pi$, the loss function is defined as $\ell(y, \mathbf{x}; \theta) = max(0, -y \cdot (c_s\ V_\theta(\mathbf{x})))$. The point of a loss function is for the algorithm to learn and reduce the error in the prediction, so given a misclassification, the classifier loss has a magnitude of $\mid c_s - V_\theta(\mathbf{x}) \mid$, and zero otherwise.

In order to make the problem manageable, it is added a Lagrange relaxation term and gradient methods to update the parameters θ. More specific, gradient-based optimization is used together with mini-batches. Then states $\chi_b = \{\mathbf{x}_i\}_i$ are sampled from the state-space χ at random and then ground-truth labels $\{y_i\}_i$ are selected to them. Given the states in the state-space χ_b, the optimization objective is defined as

$$\min_{\theta} \sum_{\mathbf{x}\in\chi_b} \ell(y, \mathbf{x}; \theta),\ s.t.\ y = +1 \Rightarrow \Delta V_\theta(\mathbf{x}) < 0 \tag{4.21}$$

The state-space χ_b is re-sampled after every gradient step. The Lagrangian relaxation is defined as

$$\min_{\theta} \sum_{\mathbf{x}\in\chi_b} \ell(y, \mathbf{x}; \theta),\ +\lambda(\frac{y+1}{2})\, max(0, \Delta V_\theta(\mathbf{x})) \tag{4.22}$$

where $\lambda \in \mathbb{R}_{>0}$ and the second term of equation 4.22 penalizes violations of the constraint defined is equation 4.20 and is called the Lyapunov decrease loss.

However, the problem was simplified by assuming that the true largest ROA S_π was known, so the question is how to get ground truth labels when the true region of attraction is unknown. Secondly, Equation 4.22 does not constrain θ to enforce the one-step decrease condition; it only penalizes violations of it. Therefore, it is vital to verify that the one-step decrease condition holds over a level set $\mathcal{V}_\theta(c)$ when θ is updated.

Algorithm 1 ROA Classifier Training (Richards et al., 2018)

```
1: Input: closed-loop dynamics $f_\pi$; initialized parametric Lyapunov function candidate $V_\theta : \chi \to \mathbb{R}_{\leq 0}$; Lagrange multiplier $\lambda \in \mathbb{R}_{>0}$; level set "expansion multiplier $\alpha \in \mathbb{R}_{>1}$; forward-simulation horizon $T \in \mathbb{N}_{\geq 1}$.
2: $c_0 \leftarrow max_{\mathbf{x}\in\chi} V(\mathbf{x})$, s.t. $\mathcal{V}_\theta(c_0) \subseteq \mathcal{D}_{v_\theta}$    ▷ compute the initial safe level set.
3: repeat
4:     Sample a finite batch $\chi_b \subset \mathcal{V}_\theta(\alpha c_k)$.
5:     $\mathcal{S}_b \leftarrow \{\mathbf{x} \in \chi_b \mid f_\pi^{(T)}(\mathbf{x}) \in \mathcal{V}_\theta(c_k)\}$.    ▷ forward-simulate the batch with $f_\pi$
6:                                                   over T steps.
7:     Update $\theta$ with (4.22) via batch SGD on $\chi_b$ and labels $\{y_i\}_i$ for points in $\mathcal{S}_b$.
8:     $c_{k+1} \leftarrow max_{\mathbf{x}\in\chi} V_\theta(\mathbf{x})$, s.t. $\mathcal{V}_\theta(c_{k+1}) \subseteq \mathcal{D}_{V_\theta}$.
9: until convergence
```

First, Lyapunov stability theory is used in order to verify that a level set $\mathcal{V}_\theta(c)$ is safe, and this is done by checking the tightened certificate $\Delta V_\theta(\mathbf{x}) < -L_{\Delta V_\theta}\tau$ at a finite set of states that cover $\mathcal{D}_V \subseteq \chi$. The Lipschitz constant $L_{\Delta V} \in \mathbb{R}_{>0}$ of ΔV and $\tau \in \mathbb{R}_{>0}$ is a measure of the density of the states that cover $\mathcal{D}_V$. After a level set $\mathcal{V}_\theta(c)$ is established as safe, the next step is to use $\mathcal{V}_\theta(c)$ to estimate labels y from S_π. As long as the dynamical system f_π is known, the iterative Algorithm 1, which will iteratively "grow" an estimate of S_π, can be implemented. The first step is to choose some initialization of the parameters data θ and use the one-step decrease condition, as it is positive definite, to verify the current safe level set $\mathcal{V}_\theta(c)$. In particular, Lyapunov stability theory is used in order to ensure that a level set $\mathcal{V}_\theta(c)$ is safe, and this is done by checking the tightened certificate $\Delta V_\theta(\mathbf{x}) < -L_{\Delta V_\theta}\tau$ at a finite set of states that cover $\mathcal{D}_V \subseteq \chi$. The Lipschitz constant $L_{\Delta V} \in \mathbb{R}_{>0}$ of ΔV and $\tau \in \mathbb{R}_{>0}$ is a measure of the density of the states that cover $\mathcal{D}_V$. After a level set

$\mathcal{V}_\theta(c)$ is verified as safe, $\mathcal{V}_\theta(c)$ is used to estimate labels y from S_π. V_θ is initialized, and used to find the largest safe level set $\mathcal{V}_\theta(c_0)$ by the one-step decrease condition, and then $\mathcal{V}_\theta(c_0)$ is used to create an estimate of S_π. The next step is to sample states inside this set and slightly around it, and with the dynamical model, it is possible to forward-simulate the samples with some horizon. That is, at iteration $k \in \mathbb{N}_{\geq 0}$, there will be a safe level set $\mathcal{V}_\theta(c_k)$ and an expanded level set $\mathcal{V}_\theta(\alpha c_k)$ for some $\alpha \in \mathbb{R}_{>1}$. The states $\mathcal{V}_\theta(\alpha c_k) \setminus \mathcal{V}_\theta(c_k)$ are forward simulated with the system dynamics f_π for $T \in \mathbb{N}_{\geq 1}$ time steps. Any states that are already inside the safe level set S_π or have mapped inside must lie within the true ROA, and those outside do not. The estimates of the true ROA S_π is used to find the labels y, and then these labels are used with the loss function 4.22 combined with stochastic gradient descent(SGD), in order to update the parameter data θ. These steps are repeated, and the safe level set S_π grows until some stopping criterion is satisfied.

To summarize, the method of iteratively adapting a Lyapunov candidate consists of four steps:

1. Verify a current safe level set $\mathcal{V}_\theta(c_k)$
2. Sample from the expanded level set $\mathcal{V}_\theta(\alpha c_k)$
3. Forward-simulate the samples $\mathcal{V}_\theta(\alpha c_k) \setminus \mathcal{V}_\theta(c_k)$ with the system dynamics f_π
4. Use the estimates of the true ROA S_π to identify labels y and update θ

4.3 Implementation of the NN Lyapunov Function for the Moore-Greitzer compressor system in Python

The implementation of the NN Lyapunov function for the Moore-Greitzer compressor system in Python is comprehensive. However, the implementation of the compressor surge class along with the implementation of a Lyapunov function for compressor surge can be studied in Appendix B. It should be noted that many of the classes used in the presented code are not shown in Appendix B, and the reader interested is referred to the code included along with this thesis.

Chapter 5

Experiments & Results

In this Chapter, the results from the neural network Lyapunov algorithm for the Moore-Greitzer compressor model will be given. This Chapter is organized as follows: Section 5.1 presents the stability analysis for the system. The result providing a GAS equilibrium beyond the original surge line is given in Section 5.2. Finally, the result given a minimization of the pressure drop across the CCV is presented in Section 5.3.

5.1 Stability Analysis of the Moore-Greitzer compressor model

The system was implemented using simulation parameters from Backi et al. (2016), which can be seen in Table 5.1.

A_c		flow area	$0.01m^2$
B		B-Parameter	0.8319
H		coefficient	0.18
L_c		length of ducts and compressor	3m
U		compressor blade tip speed	$80ms^{-1}$
V_p		plenum volume	$1.5m^3$
W		coefficient	0.25
a_s		speed of sound	$340ms^{-1}$
ψ_0, x_{1_0}		operating point for ψ, respective x_1	$0.611, 0.533$
ϕ_0, x_{2_0}		operating point for ϕ, respective x_2	$0.6, 0.3$
γ		throttle gain	$0.768, 0.411$

Table 5.1: *Simulation Parameters for the Moore-Greitzer compressor system in combination with a close-coupled valve, adopted from Backi et al. (2016).*

In the paper it is assumed that the parameters B and γ is known exactly, which introduce some uncertainty to the parameters k_i. The analysis of the compressor system can be divided into two cases: open-loop and closed-loop control.

5.1.1 Open-loop Control

In an open-loop control system there is no feedback, and the control action is independent from the output of the system. In Table 5.1, the operating point for pressure ψ and mass flow ϕ has two given values, same with the throttle gain γ. For the first value of each parameter, the equilibrium point is asymptotically stable since:

$$A_{\gamma=0.768} = \begin{bmatrix} -0.5905 & 1.2019 \\ -0.8320 & -0.8626 \end{bmatrix} \tag{5.1}$$

and the real part of the system's eigenvalues are located in the left half-plane.The eigenvalues are $\gamma_{1,2} = -0.7265 \pm 0.9907i$ for the stable system. In comparison, if considering the second value of each parameter, then the system is unstable as:

$$A_{\gamma=0.411} = \begin{bmatrix} -0.3383 & 1.2019 \\ -0.8320 & 0.8626 \end{bmatrix} \tag{5.2}$$

the real part of the eigenvalues,$\gamma_{1,2} = 0.262 \pm 0.7996i$, are located in the right half-plane. In an open-loop control system, the equilibrium point will be stable for a throttle gain γ greater than a critical value γ_c, and unstable for a throttle gain γ less than the same critical value.

5.1.2 Closed-loop Control

In a closed-loop control system, the feedback between the outputs and inputs of the system can be used to stabilize an unstable system. The equilibrium point will be stable for a throttle gain γ greater than a critical value, and in such case, there would be no need to create an anti-surge controller. However, without feedback, the equilibrium point is unstable for a throttle gain γ less than the same critical value γ_c, and an anti-surge controller can be implemented in order to stabilize the equilibrium.

5.2 Globally Asymptotically Stable Equilibrium Point

For the GAS control law, the saturation constraints for the state variables and control input are defined as follows:

$$\begin{aligned} \hat{\psi}_{max} &= 0.3 \\ \hat{\phi}_{max} &= 0.6 \\ u_{max} &= 0.3 \end{aligned} \tag{5.3}$$

where $\hat{\psi}$ is the plenum pressure coefficient, $\hat{\phi}$ is the mass flow coefficient and the control input u is the pressure drop across the valve. Furthermore, the size of the grid is defined as:

$$Grid = 63001 \tag{5.4}$$

The result can be seen in Figure 5.1 and Figure 5.2. With a high-pressure drop across the CCV, achieved with a high maximum value of the control input u, the equilibrium point is globally asymptotically stable.

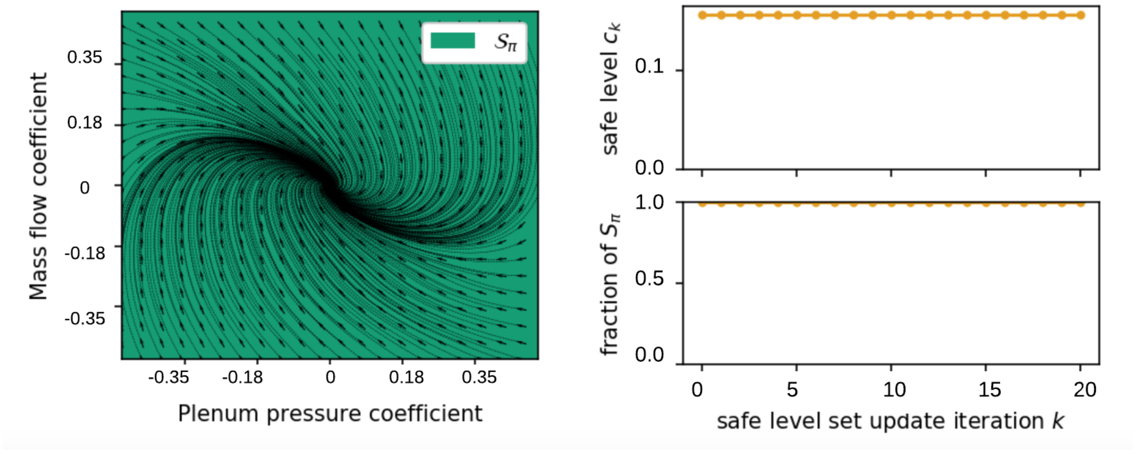


Figure 5.1: *(a) GAS equilibrium point (b) Training behaviour of the NN candidate. Maximum state variables and control input defined in Equation 5.3. All trajectories converge towards the equilibrium point and the region of attraction S_π covers the entire grid. (Mass flow coefficient: annulus averaged, axial velocity divided by wheel speed, Plenum pressure coefficient: pressure divided by density and the square of wheel speed. The system is transformed such that the equilibrium point is at the origin.)*

As one can see in Figure 5.1(a), the region of attraction S_π for the closed-loop system f_π given the fixed policy π covers the entire grid, and, as can be seen from 5.2(a) so does the NN Lyapunov candidate. It can also be seen that all trajectories converge towards the equilibrium point. In Figure 5.2(b), the NN fraction of S_π (the safe set size), is 100% of the ROA. It can be seen that the safe set size is already 100% at the first iteration, and stays that way for all of the 20 iterations.

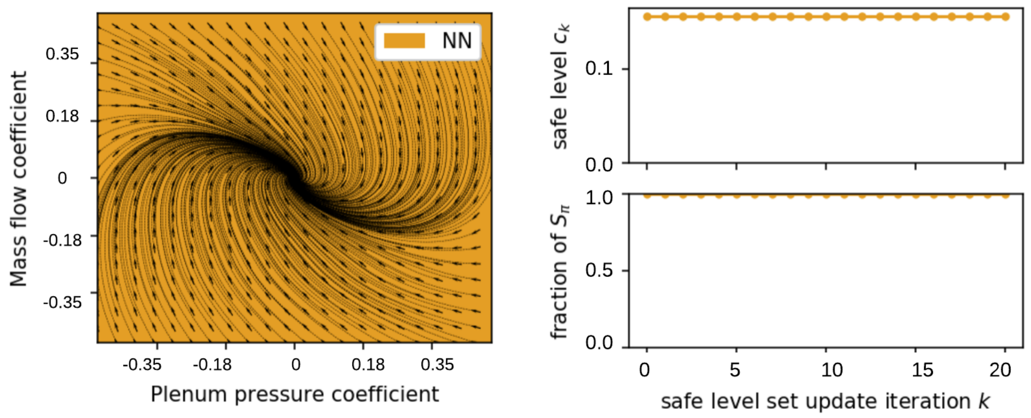
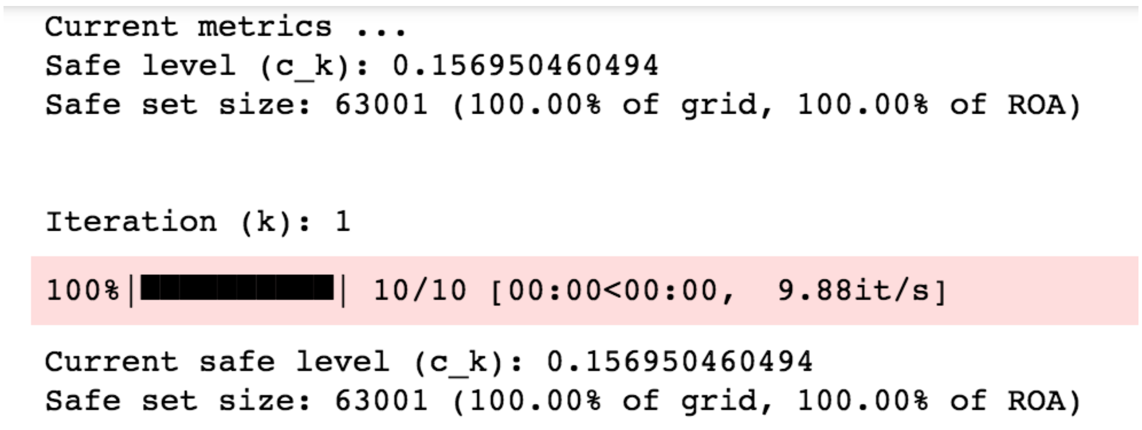


Figure 5.2: *(a) Safe NN Lyapunov candidate level sets (b) Training behaviour of the NN candidate. Maximum state variables and control input defined in Equation 5.3. The NN Lyapunov candidate covers the entire grid. The training behaviour of the neural network stays constant since the safe set size is already 100% at the first iteration. (Mass flow coefficient: annulus averaged, axial velocity divided by wheel speed, Plenum pressure coefficient: pressure divided by density and the square of wheel speed. The system is transformed such that the equilibrium point is at the origin.)*

Figure 5.3 shows that the safe set size is 63001 in the initial state, and hence, covers the entire grid, defined in Equation 5.4, and that the NN Lyapunov function covers 100.00% of S_π. The current safe level c_k is only 0.1569 initially, and, as can be seen in Figure 5.2(b), c_k remains constant for all of the iterations and does not converge to the fixed boundary $c_S = 1$.



```
Current metrics ...
Safe level (c_k): 0.156950460494
Safe set size: 63001 (100.00% of grid, 100.00% of ROA)


Iteration (k): 1

100%|██████████| 10/10 [00:00<00:00,  9.88it/s]

Current safe level (c_k): 0.156950460494
Safe set size: 63001 (100.00% of grid, 100.00% of ROA)
```

Figure 5.3: *Initial safe level and safe set size for the GAS equilibrium. Maximum state variables and control input defined in Equation 5.3. The safe set size was 63001 in the initial state and covered the entire grid, defined in Equation 5.4. The current safe level c_k does not converge to the fixed boundary $c_S = 1$.*

5.3 Locally Asymptotically Stable Equilibrium Point

With a low-pressure drop across the CCV, achieved with a low maximum value of u, the equilibrium point is locally asymptotically stable. For the AS control law, the saturation constraints for the state variables and control input are defined as follows:

$$\begin{aligned} \hat{\psi}_{max} &= 0.46 \\ \hat{\phi}_{max} &= 0.5 \\ u_{max} &= 0.05 \end{aligned} \tag{5.5}$$

The result can be seen in Figure 5.4, where, in (a), the true ROA S_π is represented with the green color, the NN Lyapunov function with orange and the LQR Lyapunov function with blue. The NN Lyapunov function V_θ performs much better than the traditional Lyapunov approach and covers approximately 81% of the true ROA at its best iteration. However, Figure 5.4(b) shows that the current safe level c_k of V_θ grows non-monotonically (where k is the iteration $k \in \mathbb{N}_{\geq 0}$) and does not converge to the fixed boundary $c_S = 1$. The safe level set $\mathcal{V}_\theta(c_k)$ also grows non-monotonically to cover a significant part of S_π. For this example, the safe set size is only 2.46% of the grid defined in Eq. 5.4, while in Figure 5.1, the safe set size is 100% of the grid for the GAS equilibrium point.

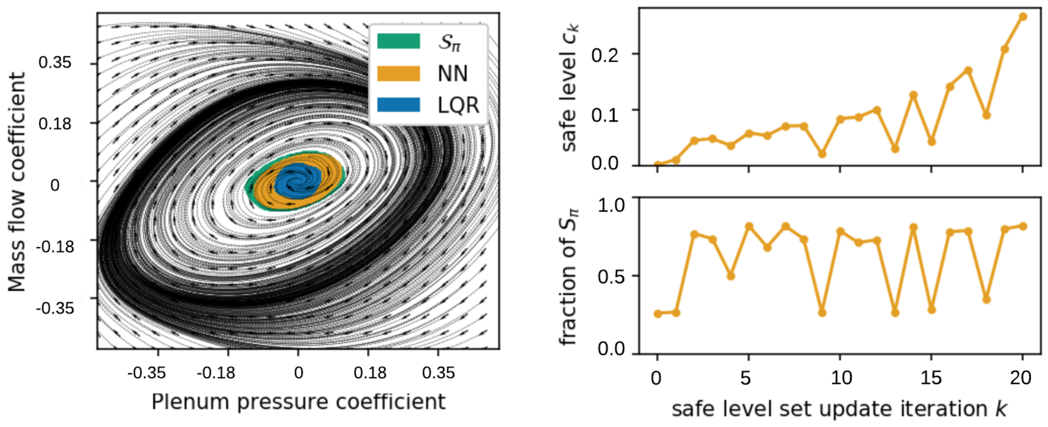


Figure 5.4: *(a) Safe Lyapunov candidate level sets. (b) Training behaviour of the NN candidate. Maximum state variables and control input defined in Equation 5.5. The control law resulted in an asymptotically stable equilibrium. The true ROA S_π is represented with the green color, the NN Lyapunov function with orange and the LQR Lyapunov function with blue. Both the current safe level c_k and the safe level set $\mathcal{V}_\theta(c_k)$ grow non-monotonically. (Mass flow coefficient: annulus averaged, axial velocity divided by wheel speed, Plenum pressure coefficient: pressure divided by density and the square of wheel speed. The system is transformed such that the equilibrium point is at the origin.)*

In Figure 5.5 it is shown that the NN Lyapunov function covers 26.16% of S_π before the first iteration, and that the safe set size is only 0.79% of the grid defined in Eq. 5.4. The results from the twentieth iteration is shown in Figure 5.6, where the the safe set size is 2.46% of the grid and the NN Lyapunov function covers 81.77% of S_π.

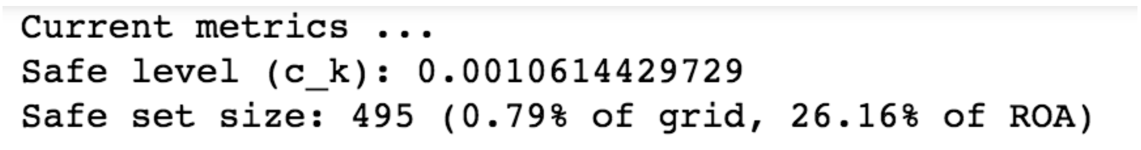
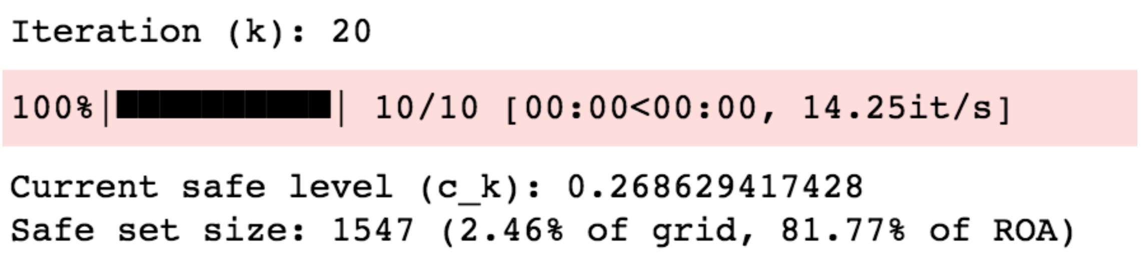

```
Current metrics ...
Safe level (c_k): 0.0010614429729
Safe set size: 495 (0.79% of grid, 26.16% of ROA)
```

Figure 5.5: *Initial safe level and safe set size for AS equilibrium with maximum state variables and control input defined in Equation 5.5. The initial safe set size was 495 and only covered* 0.79% *of the entire grid, defined in Equation 5.4. The safe level c_k was initially significantly small compared to the fixed boundary $c_s = 1$.*



```
Iteration (k): 20
100%|██████████| 10/10 [00:00<00:00, 14.25it/s]
Current safe level (c_k): 0.268629417428
Safe set size: 1547 (2.46% of grid, 81.77% of ROA)
```

Figure 5.6: *Safe level and safe set size for AS equilibrium with maximum state variables and control input defined in Equation 5.5. At the twentieth iteration, the safe set size was* 2.46% *of the grid defined in Equation 5.4. The current safe level has increased to* $c_k = 0.2689$*, but has not converged to the fixed boundary* $c_s = 1$.

In Figure 5.7, 40 more iterations have been run, and by comparing Figure 5.4(b) and Figure 5.7(b), several observations can be made. First of all, the safe level c_k continues to grow, non-monotonically, towards the fixed boundary $c_S = 1$ and at iteration 60, $c_k = 0.876$. Secondly, $\mathcal{V}_\theta(c_k)$ continues to grow non-monotonically, but does not improve much during the 40 new iterations, except that the difference between low and large values decreases.

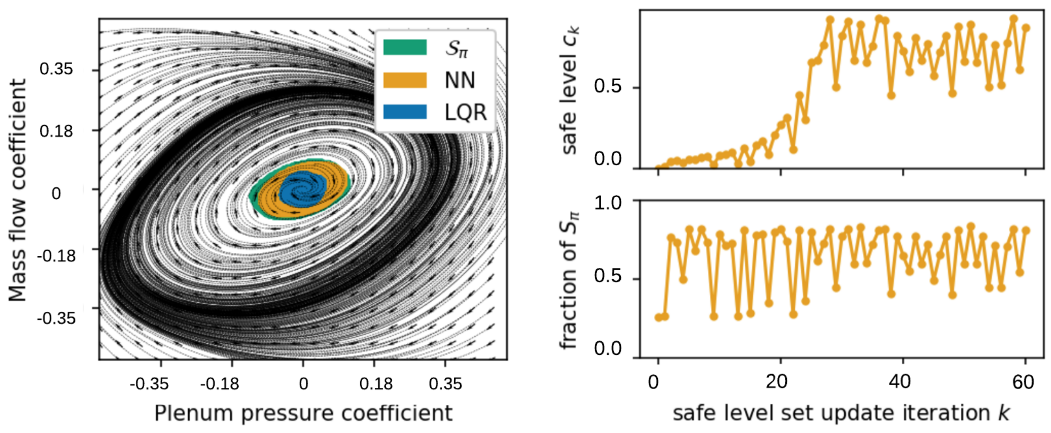


Figure 5.7: *(a) Safe Lyapunov candidate level sets. (b) Training behaviour of the NN candidate. This is the identical control law as in Figure 5.4, only that 40 more iterations have been run. As can be seen, as more iterations are performed, the safe level* c_k *continues to grow, non-monotonically, towards the fixed boundary* $c_S = 1$. *(Mass flow coefficient: annulus averaged, axial velocity divided by wheel speed, Plenum pressure coefficient: pressure divided by density and the square of wheel speed. The system is transformed such that the equilibrium point is at the origin.)*

5.3.1 Alternative Control Input for the Asymptotically Stable Equilibrium

Further analysis shows that the system is asymptotically stable for an even lower maximum value of the control input. For the control law, the saturation constraints for the state variables and control input are defined as follows:

$$\begin{aligned}\hat{\psi}_{max} &= 0.46 \\ \hat{\phi}_{max} &= 0.5 \\ u_{max} &= 0.03\end{aligned} \tag{5.6}$$

The results can be seen in Figure 5.8(a), where the NN Lyapunov function V_θ performs poorly by covering approximately 58% of the true ROA. Figure 5.8(b) shows that the safe level c_k of V_θ declines non-monotonically towards zero. The safe level set $\mathcal{V}_\theta(c_k)$ barely grows from its initial starting point and the fraction of the S_π stays close to constant. Figure 5.10 shows that at iteration 20, c_k is only 2.36×10^{-5}, and that the safe set size is only 0.79% of the grid defined in Equation 5.4.

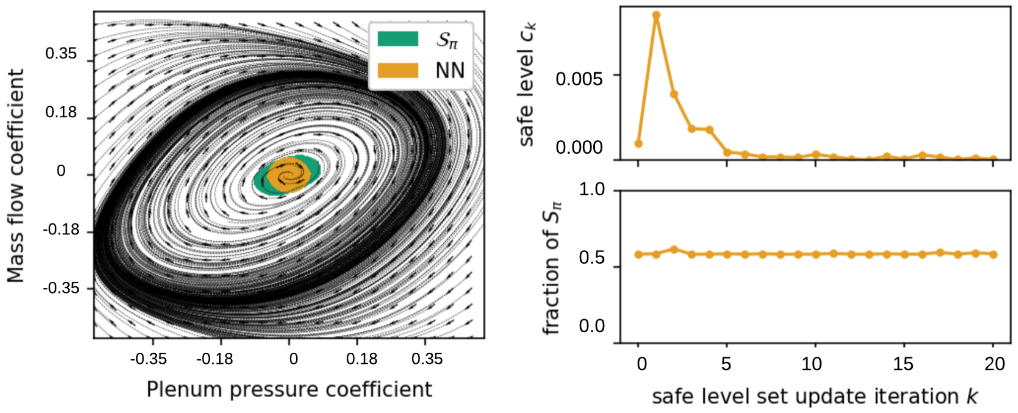


Figure 5.8: *(a) Safe NN Lyapunov candidate level sets (b) Training behaviour of the NN candidate. Maximum state variables and control input defined in Equation 5.6. The control law resulted in an asymptotically stable equilibrium. The true ROA S_π is represented with the green color and the NN Lyapunov function with orange. The safe level c_k of V_θ declines non-monotonically towards zero and the safe level set $\mathcal{V}_\theta(c_k)$ stays close to constant. (Mass flow coefficient: annulus averaged, axial velocity divided by wheel speed, Plenum pressure coefficient: pressure divided by density and the square of wheel speed. The system is transformed such that the equilibrium point is at the origin.)*

The NN Lyapunov function and the LQR Lyapunov function are shown in two separate figures, because they perform equally, and therefore cover each other. The result of the LQR Lyapunov function can be seen in Figure 5.9.

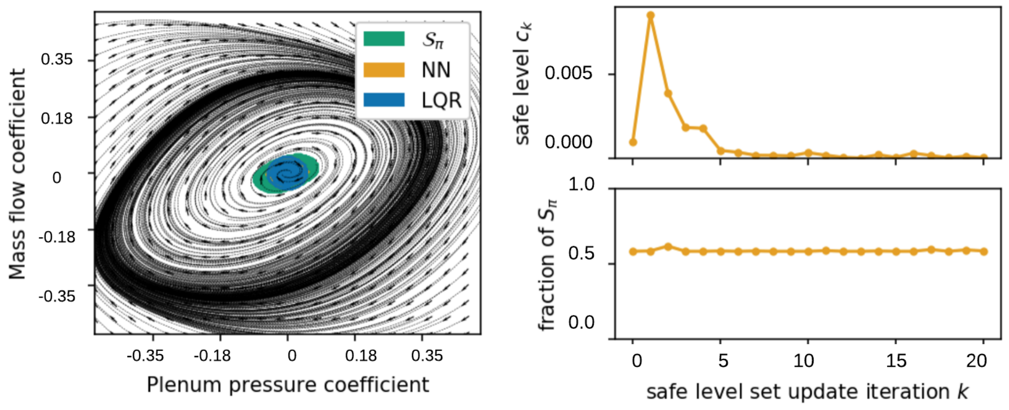


Figure 5.9: *(a) Safe LQR Lyapunov candidate level sets (b) Training behaviour of the NN candidate. Maximum state variables and control input defined in Equation 5.6. The control law resulted in an asymptotically stable equilibrium. The true ROA S_π is represented with the green color and the LQR Lyapunov function with blue. (Mass flow coefficient: annulus averaged, axial velocity divided by wheel speed, Plenum pressure coefficient: pressure divided by density and the square of wheel speed. The system is transformed such that the equilibrium point is at the origin.)*

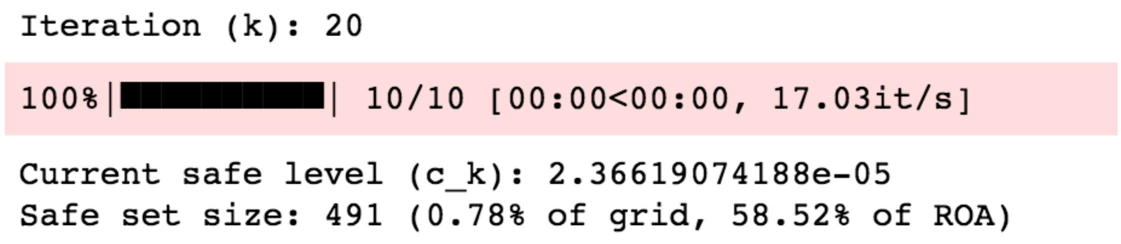

```
Iteration (k): 20
100%|██████████| 10/10 [00:00<00:00, 17.03it/s]
Current safe level (c_k): 2.36619074188e-05
Safe set size: 491 (0.78% of grid, 58.52% of ROA)
```

Figure 5.10: *Safe level and safe set size for AS equilibrium with maximum state variables and control input defined in Equation 5.6. At the twentieth iteration, the safe set size was* 0.78% *of the grid defined in Equation 5.4. The current safe level c_k is significantly small compared to the fixed boundary $c_s = 1$.*

5.3.2 Unstable Equilibrium Point

If the saturation constraint for the control input is set to be even lower, it can be seen in Figure 5.11(a) that V_θ goes beyond S_π and with the proposed Lyapunov candidate the equilibrium is unstable. The same can be observed for the LQR Lyapunov function in Figure 5.12(a). In the given case, the saturation constraints for the state variables and control input are defined as follows:

$$\begin{aligned} \hat{\psi}_{max} &= 0.46 \\ \hat{\phi}_{max} &= 0.5 \\ u_{max} &= 0.01 \end{aligned} \tag{5.7}$$

Figure 5.11(b) shows that the safe level c_k of V_θ declines non-monotonically towards zero. In Figure 5.13 it can be seen that the NN Lyapunov function initially covered 254.69% of S_π, and from Figure 5.14 it can be seen that at iteration 20, V_θ covers the same percentage of the ROA. By comparing the two figures, it shows that the safe set size stays the same and only covers 0.78% of the grid defined in Equation 5.4. In Figure 5.11(b) the training behavior of the NN candidate does not show because it is beyond the limits of S_π.

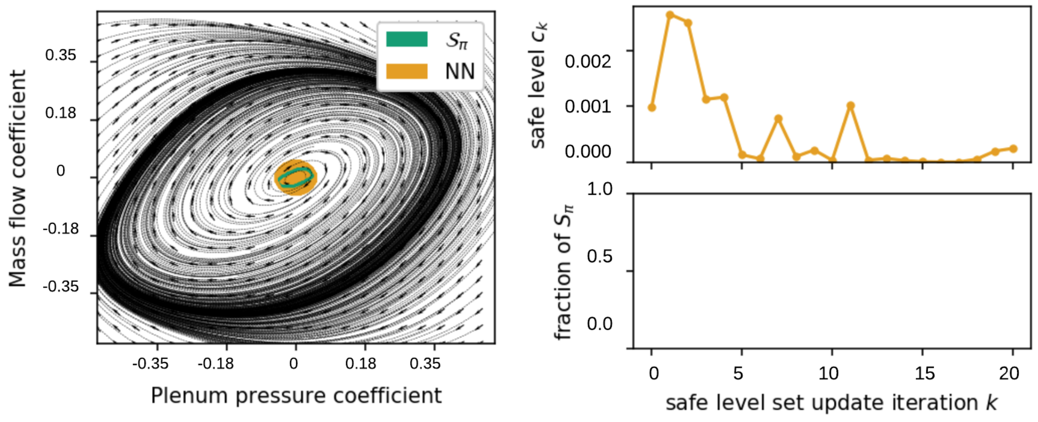


Figure 5.11: *(a) Lyapunov candidate level sets (b) Training behaviour of the NN candidate. Maximum state variables and control input defined in Equation 5.7. The control law resulted in an unstable equilibrium. The true ROA S_π is represented with the green color and the NN Lyapunov function with orange. The NN Lyapunov candidate goes beyond S_π. The safe level c_k of V_θ declines non-monotonically towards zero. (Mass flow coefficient: annulus averaged, axial velocity divided by wheel speed, Plenum pressure coefficient: pressure divided by density and the square of wheel speed. The system is transformed such that the equilibrium point is at the origin.)*

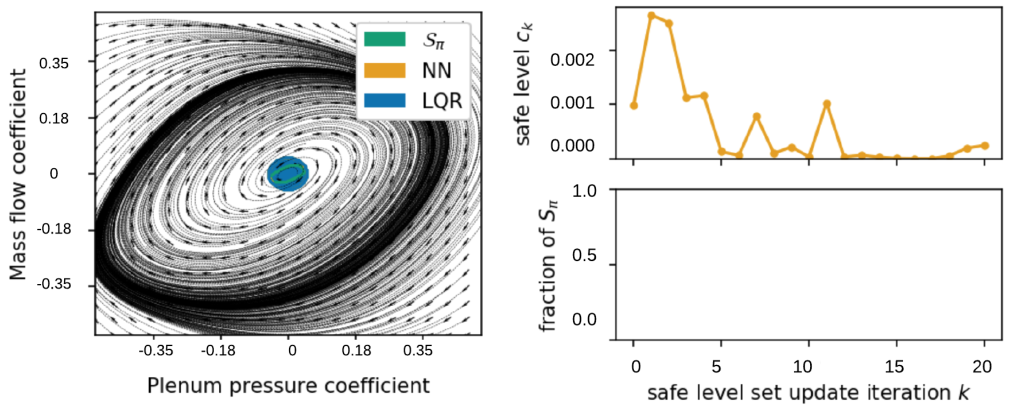
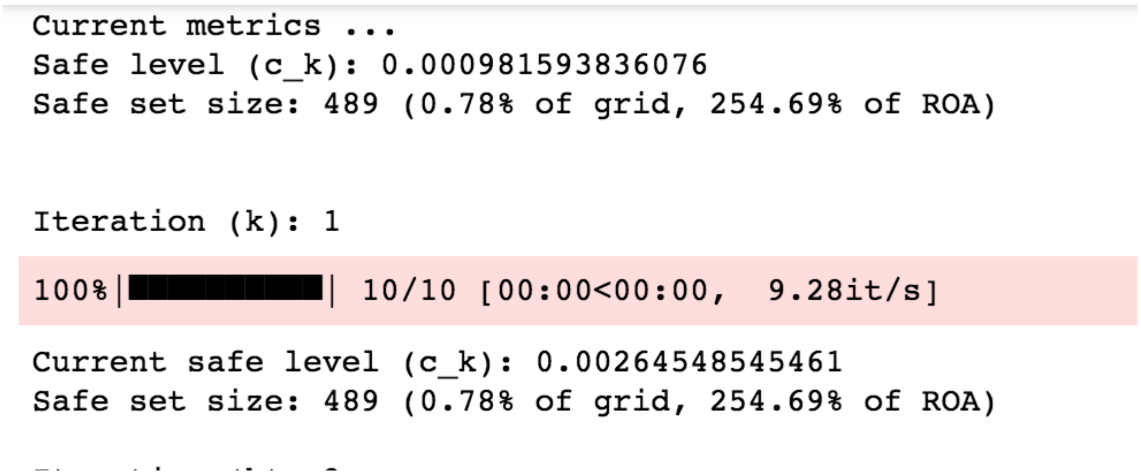


Figure 5.12: *(a) LQR candidate level sets (b) Training behaviour of the NN candidate. Maximum state variables and control input defined in Equation 5.7. The control law resulted in an unstable equilibrium. The true ROA S_π is represented with the green color and the LQR Lyapunov function with blue. The LQR Lyapunov candidate goes beyond S_π. (Mass flow coefficient: annulus averaged, axial velocity divided by wheel speed, Plenum pressure coefficient: pressure divided by density and the square of wheel speed. The system is transformed such that the equilibrium point is at the origin.)*

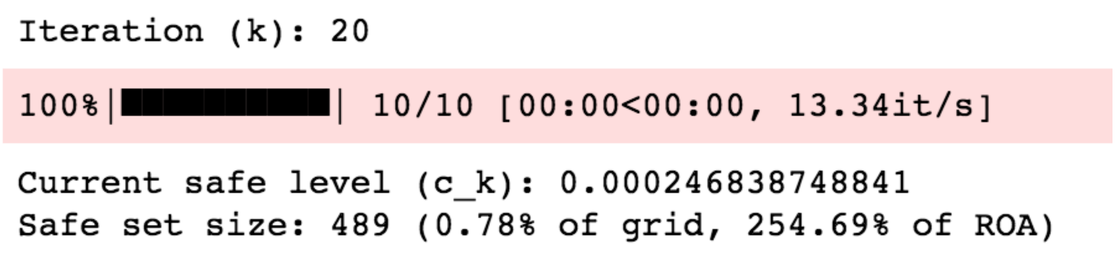

```
Current metrics ...
Safe level (c_k): 0.000981593836076
Safe set size: 489 (0.78% of grid, 254.69% of ROA)


Iteration (k): 1

100%|██████████| 10/10 [00:00<00:00,  9.28it/s]

Current safe level (c_k): 0.00264548545461
Safe set size: 489 (0.78% of grid, 254.69% of ROA)
```

Figure 5.13: *Initial safe level and safe set size for the equilibrium with maximum state variables and control input defined in Equation 5.7. The initial safe set size is* 0.78% *of the grid defined in Equation 5.4; however, it initially covers* 259.69% *of the ROA. The equilibrium is thus unstable for the initial value of the NN Lyapunov function candidate.*



```
Iteration (k): 20
100%|██████████| 10/10 [00:00<00:00, 13.34it/s]
Current safe level (c_k): 0.000246838748841
Safe set size: 489 (0.78% of grid, 254.69% of ROA)
```

Figure 5.14: *Safe level and safe set size for the equilibrium with maximum state variables and control input defined in Equation 5.7. At iteration 20, the safe set size has not improved, and is still* 254.69% *of the ROA. The equilibrium is unstable.*

Chapter 6

Discussion

The following chapter presents a discussion of the objectives and the research questions given in the introduction of this thesis. Moreover, it attempts to explain the results from the preceding chapter that are not self-evident.

The objective of this thesis has been to gain an overview over state-of-the-art regarding stability and robustness in nonlinear dynamical systems in order to maximize the safe level set that can be used as a safety certificate for a given nonlinear, closed-loop dynamical compressor system. The focus in this thesis can be divided into four parts:

1. Present an extensive literature study consisting of scientific papers regarding AI and ML methods introduced into the control loop, in addition to previous research related to compressor surge control.
2. Provide background theory regarding compressor surge control and the most relevant technologies used in order to construct safe level sets for general closed-loop dynamical systems.
3. Design an anti-surge controller for the Moore-Greitzer compressor with a focus on maximization of its safe level set using a neural network Lyapunov function candidate and comment on the results.
4. Present an evaluation of the NN Lyapunov function presented by Richards et al. (2018), and discuss future work.

In the introduction of this thesis, two research questions were formulated. Concerning the first question, research has shown that it is possible to find a suitable Lyapunov function candidate for the Moore-Greitzer compressor system. In addition to the NN Lyapunov function implemented in this thesis, Simon and Valavani

(1991) presented the theoretical background for the creation of an anti-surge controller based on Lyapunov control theory, and Backi et al. (2013); Liaw et al. (2002) implemented an anti-surge controller based on Lyapunov control theory. In order to answer the second research question, the results of the NN anti-surge controllers have to be analyzed.

6.1 Stability Analysis of the Compressor Surge System

Compressor surge represents undesired oscillations in mass flow and pressure, and it can cause a reduction in performance or even damage the compressor. Surge avoidance can be handled with two different approaches. The cautious way is to ensure that the equilibrium point is stable in open-loop. Considering the simulation parameters in Table 5.1, this means that the system is operated with the throttle gain $\gamma = 0.768$. In this case, if Figure 3.2 is considered, the system is operated at point A where the system is self-compensating and surge is avoided. This method ensures that the equilibrium is stable, but it will limit the range of mass flow for which the compressor can be used. The throttle gain γ is proportional to the throttle opening. If the throttle gain is decreased, the equilibrium point moves along the compressor characteristics towards lower values of mass flow. The second approach is to operate the system with a throttle gain of $\gamma = 0.411$. In an open-loop, the equilibrium point would be unstable with both eigenvalues located in the right half-plane. The system is in the given case operated at point B in Figure 3.2, and without an anti-surge controller, surge will occur. However, in closed-loop, an anti-surge controller can be implemented to stabilize the equilibrium point for $\gamma = 0.411$. In this thesis, different control inputs for the compressor surge system were considered.

6.1.1 Globally Asymptotically Stable Equilibrium Point

The first control law for the CCV resulted in a GAS equilibrium beyond the original surge line. This can be seen from Figure 5.1, where the region of attraction S_π for the closed-loop system f_π, given the fixed policy π, covers the entire grid. In Figure 5.2(a) it can be seen the result of the NN Lyapunov function covers the entire grid, and in (b) the NN fraction of S_π, is 100% of the ROA. It can also be seen that the safe set size is already 100% at the first iteration. Since the policy is fixed to the LQR solution, and, with the given policy, the system is globally asymptotically stable, the NN Lyapunov candidate and the LQR Lyapunov candidate provides the same result, and there is no need for the NN to explore safe states. This confirms previous finding in the literature, where in Gravdahl and Egeland (1997), a control law was derived for the CCV that resulted in a global uniform asymptotic stable equilibrium point.

In the code that accompanies the NN Lyapunov function, the maximum value of state and action have to be chosen. Given these saturation constraints, the policy is fixed to the LQR solution for the linearized, discretized system. For the GAS control law, the maximum state variables and control input is defined in Equation 5.3. These values were chosen given intersection point of the compressor and throttle characteristics in Figure 4.1. With the chosen state and action values, the throttle line crosses $\Psi_e(\hat{\phi})$ where the slope is negative, and the equilibrium is stable.

GAS is very desirable as every trajectory converges to the equilibrium point and the objective was to create an anti-surge controller with a focus on the maximization of the safe level set for the given compressor system. Since the equilibrium point is GAS for the given control law, the entire level set is safe, and there is no need for Algorithm 1 to iteratively grow an estimate of the true ROA S_π. Despite the fact that the code required to solve V_θ (Richards et al., 2018) does show that the equilibrium point is GAS, the potential of the algorithm is not fulfilled. In order to evaluate how well the algorithm scores for a complex nonlinear system, it must thus be suggested a second control law for the CCV.

6.1.2 Locally Asymptotically Stable Equilibrium Point

Since the control variable represents the pressure drop over the valve, it is considered as a loss of energy in the compressor system, and it is therefore beneficial to keep the control variable as low as possible. Since the control input u is constrained to be a particular value $u = [-\overline{u}, \overline{u}]$, it is possible to minimize the pressure drop across the CCV, but as a consequence, the system will only be locally asymptotically stable. In Backi et al. (2013) it is argued that the range of mass flow should be as wide as possible, and, as a result, the stability results will be more powerful. Global asymptotic stability is more powerful than locally asymptotic stability; however, the equilibrium does not necessarily need to be GAS as long as the area for which the system is most likely operated within is covered. With this approach, the control input can be minimized for a locally asymptotic stable equilibrium, and thus, the loss in the system will be reduced. Specifically, in that case, since the equilibrium point would be locally asymptotically stable, Algorithm 1 can be used to iteratively grow an estimate of the true ROA, in order to find the largest safe level set $\mathcal{V}_\theta$.

The result of the second control law can be seen in Figure 5.4. For the locally asymptotically stable control law, the maximum state variables and control input is defined in Equation 5.5. These values were chosen with the trial and error method, where the objective was to see how low the control input could be and still provide a stable equilibrium. Further discussion of choice of saturation constraint for the control input is given in the following subsection. With a low pressure drop across the CCV, the equilibrium point is locally asymptotically stable. The NN Lyapunov function V_θ performs much better than the traditional Lyapunov approaches, and covers approximately 81% of the true ROA. However, Figure 5.4 (b) shows that the safe level c_k of V_θ grows non-monotonically, and does not converge to the fixed boundary $c_S = 1$. The safe level set $\mathcal{V}_\theta(c_k)$ also grows non-monotonically to cover a significant part of S_π.

6.1.3 Evaluation of the Neural Network Lyapunov Function

There are several factors with the code created by Richards et al. (2018) that have to be taken into consideration. First, it is not guaranteed that the safe level set $\mathcal{V}_\theta(c)$ will monotonically grow in volume, nor is the convergence of $\mathcal{V}_\theta(c)$ to S_π. In Figure 5.7 it can be seen that the safe level set $\mathcal{V}_\theta(c)$ oscillates considerably for the compressor system, but also that the fraction of the true ROA S_π does improve. However, in Figure 5.8 $\mathcal{V}_\theta(c)$ does not oscillate, but the fraction of S_π stays close to constant. Furthermore, the safe level c_k is not guaranteed to go to the safe level c_S. In Figure 5.7(b), it can be seen that c_k continues to grow, non-monotonically, towards c_S as the number of iterations increases, and at iteration 60, $c_k = 0.876$. In comparison, Figure 5.2(b) shows that c_k remains constant for all iterations. The reason for this can be that since the GAS control law initially covers the entire grid, there is no need for the Algorithm 1 to iteratively adapt safe level sets to the shape of S_π. Finally, Richards et al. (2018) do conclude that the NN Lyapunov candidate V_θ can, without identifying unsafe states as safe, find a subset of the true ROA.

In the code that accompanies the NN Lyapunov function, the maximum values of states and action have to be chosen. This choice will affect the remaining calculations in the system, such as the LQR $\mathbf{K}$ matrix and the estimated region of attraction S_π and it is thus important to choose sufficient values. A sufficient choice requires information regarding inputs and outputs of the system; otherwise, the code will not provide an adequate result. If the values are chosen too high, with the motive of ensuring that the area for which the system is most likely operated within is covered, the code provides an inadequate result. On the contrary, if the saturation constraint for the control input is chosen too low, the code also provides an inadequate result, as can be seen in Figure 5.11. In Figure 5.11(a) it can be seen that the NN Lyapunov function candidate goes beyond the limits of S_π, and Figure 5.13 shows that the safe set size is initially 254.69% of the ROA. The reason for this can be that while the algorithm grows an estimate of S_π, the given NN Lyapunov function candidate initially gives an unstable result and thus, does not decrease during training. Without feedback the equilibrium point is unstable for the throttle gain $\gamma = 0.411$; therefore, it is reasonable that the NN Lyapunov function candidate provides an inadequate result with a control input close to zero. This can be seen from the arrows in Figure 5.13: outside of S_π the arrows point outwards and the trajectories do not converge to the equilibrium point. For comparison, in Figure 5.1, it can be seen that all trajectories converges to the equilibrium point as the entire grid is covered by S_π.

In order to understand the robustness of the NN Lyapunov function, the original system for the inverted pendulum system by Richards et al. (2018) was considered. For the inverted pendulum, it is easy to choose the maximum state and action values.

The first state is the angle from the upright equilibrium point $x_e = 0$, and the second state is the angular velocity. The maximum state values are set to be 180 degrees and 360 degrees per second, respectively. The maximum action input is chosen such that if the pendulum falls over a certain angle, the pendulum will fall down and it can not recover. The Greitzer-Moore compressor system is a significantly more complex system than the inverted pendulum and, accordingly, the choice of maximum state and action values are less intuitive and more comprehensive. Hence, the code that accompanies the NN Lyapunov function requires considerable information about the system to be investigated, as only a small change of values can change how well the algorithm performs. Notwithstanding the above, two control laws for the CCV have been presented in this thesis, ensuring both global and local asymptotic stability, and verifying that the code works well under ideal circumstances.

The code required to solve the NN Lyapunov function created by Richards et al. (2018) is comprehensive but easily understood. Although the performance was not ideal, the code that accompanies the NN Lyapunov function is nevertheless a great step towards using NNs to estimate the region of attraction for nonlinear systems. In addition, the visualization of the different Lyapunov candidates and the true ROA is impressive. Despite the limitations of this method, and consequently, the poor results in Figure 5.9 and Figure 5.11, two control laws for the CCV have been presented in this thesis, showing that the code works well under ideal circumstances. Choice of inputs and outputs of the dynamics affect the performance of the NN Lyapunov function, which is a reasonable demand. Consequently, the evidence from this study suggests that this approach has potential, but that the code could be more robust. Numerical parameters may affect the system, but there may also be inherent errors in the code which was not detected by the author. However, only knowledge of inputs and outputs of the dynamics is required, and the system does not need to have a specific model structure. With the saturation constraints based on the compressor and throttle characteristics in Figure 4.1, the NN Lyapunov function provided a GAS control law.

Chapter 7

Future Work

The following chapter discusses several possible future improvements to this thesis' contributions in order to expand the robustness of the code and thus the robustness of the anti-surge controllers.

The design and development of the control laws for the Moore-Greitzer compressor system can be further investigated in order to make the control laws more robust for the industry. It is recommended that further research regarding the Moore-Greitzer compressor should be undertaken in the following areas:

- In the presented thesis, the output of the system in (4.12) is defined as

  $$y = h(x) = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \tag{7.1}$$

  For further investigation, since it is cheaper and more practical to only measure pressure (Gravdahl and Egeland, 2012), the output of the system in (4.12) should thus be defined as

  $$y = h(x) = x_1 \tag{7.2}$$

- For future work the impact of time delay(s) in the Moore-Greitzer compressor system should be explored.

As previously mentioned, the experiments presented in this thesis suggest that the code required to solve the NN Lyapunov function created by Richards et al. (2018) should be further investigated. The code depends on the choice of maximum state and action values, which have proved difficult to choose for this particular nonlinear system. It is recommended that further research regarding the stability analysis and robustness of the code should be undertaken in the following area:

- For the Moore-Greitzer compressor system, it made sense to choose the saturation constraints based on the compressor and throttle characteristics in Figure 4.1 and a GAS control law was proposed. The saturation constraints for the locally asymptotically stable control law were chosen by lowering the control input as much as possible and still provide a stable equilibrium. Therefore, it is recommended that further research evaluate how well the code performs on a nonlinear system less complex than the Moore-Greitzer compressor system. It is the author's understanding that the code will perform well on a system with definite boundaries.

Chapter 8

Conclusion

The first objective of this thesis has been to gain an overview over state-of-the-art regarding the construction of safe level sets for closed-loop dynamical systems in general and design an anti-surge controller for the Moore-Greitzer compressor with focus on maximization of its safe level set in particular. The second objective has been to evaluate the performance of the code required to create a Neural Network Lyapunov function (Richards et al., 2018) for a nonlinear system more complicated than in the original code.

The presented thesis has developed two anti-surge controllers for a Moore-Greitzer compression system in combination with a close-coupled valve. The design tools used are Lyapunov control theory in combination with NNs. One of the control laws ensures global asymptotic stability for the equilibrium. The second control law only ensures locally asymptotic stability and focuses on the minimization of the pressure drop across the valve.

The code required to create a NN Lyapunov function created by Richards et al. (2018) is comprehensive, but easily understood. The most important limitation lies in the knowledge required regarding inputs and outputs of the system. Although the performance was not ideal, the code that accompanies the NN Lyapunov function is nevertheless a great step towards using NNs to estimate the region of attraction for nonlinear systems. This study has gone some way towards enhancing the understanding of the code, but further analysis should be conducted.

Bibliography

Backi, C. J., Gravdahl, J. T., and Grøtli, E. I. (2013). Nonlinear observer design for a greitzer compressor model. In *21st Mediterranean Conference on Control and Automation*, pages 1457–1463. IEEE.

Backi, C. J., Gravdahl, J. T., and Skogestad, S. (2016). Robust control of a two-state greitzer compressor model by state-feedback linearization. In *2016 IEEE Conference on Control Applications (CCA)*, pages 1226–1231. IEEE.

Barkhordari, M., Motlagh, M. R. J., and Keshavarz, M. (2008). Enlarging region of attraction in input-output linearization method. In *2008 10th International Conference on Control, Automation, Robotics and Vision*, pages 2244–2249. IEEE.

Berkenkamp, F., Moriconi, R., Schoellig, A. P., and Krause, A. (2016). Safe learning of regions of attraction for uncertain, nonlinear systems with gaussian processes. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 4661–4666. IEEE.

Bhimra, M. A., Nazir, U., and Taj, M. (2019). Using 3d residual network for spatio-temporal analysis of remote sensing data. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1403–1407. IEEE.

Ferguson, T. B. (1963). *The centrifugal compressor stage*. Butterworths.

Fontaine, D., Liao, S., Paduano, J., and Kokotovic, P. V. (2004). Nonlinear control experiments on an axial flow compressor. *IEEE transactions on control systems technology*, 12(5):683–693.

Gale, S., Vestheim, S., Gravdahl, J. T., Fjerdingen, S., and Schjølberg, I. (2013). Rbf network pruning techniques for adaptive learning controllers. In *9th International Workshop on Robot Motion and Control*, pages 246–251. IEEE.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Gravdahl, J. T. and Egeland, O. (1997). Compressor surge control using a close-coupled valve and backstepping. In *Proceedings of the 1997 American Control Conference (Cat. No. 97CH36041)*, volume 2, pages 982–986. IEEE.

Gravdahl, J. T. and Egeland, O. (2012). *Compressor surge and rotating stall: modeling and control*. Springer Science & Business Media.

Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

Haykin, S. S., Haykin, S. S., Haykin, S. S., Elektroingenieur, K., and Haykin, S. S. (2009). *Neural networks and learning machines*, volume 3. Pearson education Upper Saddle River.

Henrion, D. and Korda, M. (2013). Convex computation of the region of attraction of polynomial control systems. *IEEE Transactions on Automatic Control*, 59(2):297–312.

Jin, D. and Lin, S. (2012). *Advances in Mechanical and Electronic Engineering*. Springer.

Khalil, H. K. (2015). *Nonlinear control*. Pearson New York.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

Liaw, D.-C., Ren, S.-M., Abed, E., et al. (2002). Surge control of axial flow compression systems via linear and nonlinear designs. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, volume 6, pages 4347–4352. IEEE.

Mitchell, T. M. (1997). Mcgraw-hill science. *Engineering/Math*, 1:27.

Moore, F. and Greitzer, E. (1986). A theory of post stall transients in a axial compressor system, part 1-development of equation. *Journal of Engineering for gas turbine and power*, 108:68–76.

Nieuwenhuizen, M. (2008). *Parameter analysis and identification of the Greitzer model by analogy with the Van der Pol equation.* PhD thesis, PhD thesis, Technische Universiteit Eindhoven, Department Mechanical .

Nieuwenhuizen, M., van Helvoirt, J., and Steinbuch, M. (2009). An analogy between the van der pol equation and the greitzer model. In *2009 European Control Conference (ECC)*, pages 490–495. IEEE.

Prajna, S., Papachristodoulou, A., and Parrilo, P. A. (2002). Introducing sostools: A general purpose sum of squares programming solver. In *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, volume 1, pages 741–746. IEEE.

Rich, E. and Knight, K. (1991). Learning in neural network. *McGraw-Hill, New York*.

Richards, S. M., Berkenkamp, F., and Krause, A. (2018). The lyapunov neural network: Adaptive stability certification for safe learning of dynamic systems. *arXiv preprint arXiv:1808.00924*.

Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,.

Shteimberg, E., Kravits, M., Ellenbogen, A., Arad, M., and Kadmon, Y. (2012). Artificial intelligence in nonlinear process control based on fuzzy logic. In *2012 IEEE 27th Convention of Electrical and Electronics Engineers in Israel*, pages 1–5. IEEE.

Simon, J. and Valavani, L. (1991). A lyapunov based nonlinear control scheme for stabilizing a basic compression system using a close-coupled control valve. In *1991 American Control Conference*, pages 2398–2406. IEEE.

Yoon, S. Y., Lin, Z., and Allaire, P. E. (2012). *Control of surge in centrifugal compressors by active magnetic bearings: Theory and implementation.* Springer Science & Business Media.

Appendix A

Tools of Lyapunov Stability Theory

This section is inspired by Khalil (2015) and provides the basic definitions for Lyapunov control theory.

An equilibrium point is defined as a point in the state space where if the state of the system starts at that point, it will remain in that point for all future time. Furthermore, an equilibrium point is defined as stable if all solutions starting at nearby points stay nearby if not, it is unstable.

Consider the time-invariant system

$$\dot{x} = f(x) \tag{A.1}$$

where $f : \mathbb{D} \Rightarrow \mathbb{R}^n$ is locally Lipschitz and $x = 0 \in \mathbb{D}$ is an equilibrium point of the system. Locally Lipschitz means that the system has a solution and the solution is unique.

(In the following definitions the equilibrium point is assumed to be at the origin,

$x = 0$. *However, if the equilibrium point is not at the origin it is possible to move it to the origin by a transformation* $\tilde{x}$*) without loss of generality.*

Definition: Stability (Lyapunov Stability)

$x = 0$ is <u>stable</u> if and only if(iff)

$$\forall\, \varepsilon > 0 \quad \exists\, \delta(\varepsilon) > 0 \quad \text{s.t.} \quad \big\|x(0)\big\| \ < \delta \Rightarrow \big\|x(t)\big\| < \varepsilon \quad \forall t \geq 0$$

For an equilibrium point to be asymptotically stable all solutions starting at nearby points have to converge to the equilibrium point as time approaches infinity.

Definition: Asymptotically Stability

The equilibrium point $x = 0$ is <u>(locally) asymptotically stable</u> iff

i) it is stable

ii) $\exists\, r > 0 \quad \text{s.t.} \quad \big\|x(0)\big\| \ < r \Rightarrow \lim_{t\to\infty} x(t) = 0 \quad \text{(Convergence)}$

Once it is known that an equilibrium point is asymptotically stable, it is interesting to know how far from that point it is possible for the trajectory to be and still converge to the equilibrium point as time approaches infinity. This region is defined as the region of attraction.

Definition: Region of Attraction

$$B_r = \{r \in \mathbb{R}^n \quad : \quad \|x\| < r\ \}$$

Furthermore, global stability is very desirable and an equilibrium point is globally asymptotically stable if every trajectory converges to the equilibrium point. This implies that the equilibrium point is the unique equilibrium point.

Definition: Global Asymptotic Stability

The equilibrium point $x = 0$ is globally asymptotically stable iff

i) it is stable

ii) $\forall x(0) \quad \lim_{t\to\infty} x(t) = 0$

This implies that $x = 0$ is the only equilibrium point.

Asymptotically stability does not say anything about the rate of convergence; how long time it takes for the trajectory to go to the equilibrium point.

Definition: Convergence

$$\|x(0)\| < r \Rightarrow \lim_{t\to\infty} x(t) = 0$$

First of all, all exponentially stable systems are also asymptotically stable. However, exponential stability is a stronger form of stability, because it demands an exponential rate of convergence.

Definition: Exponential Stability

The equilibrium point $x = 0$ is (locally) exponentially stable iff $\exists\, r, k, \lambda > 0$ such that

$$\|x(0)\| < r \Rightarrow \|x(t)\| \leq k\|x(0)\|\, e^{-\lambda t} \quad \forall t \leq 0$$

Whenever possible, one should always strive to prove global exponential stability; it ensures that every trajectory converges to the equilibrium point with an exponential rate of convergence.

Definition: Globally Exponential Stability

The equilibrium point $x = 0$ is globally exponentially stable iff $\exists\, k, \lambda > 0$ such that

$$\forall x(0) \qquad \|x(t)\| \leq k\|x(0)\|\, e^{-\lambda t} \quad \forall t \leq 0$$

Appendix B

Implementation of Anti-Surge Controllers

Listing B.1 is the class created for compressor surge and Listing B.2 shows the implementation of the NN Lyapunov function and the LQR Lyapunov function given the compressor surge class. In Listing B.2 there are several classes imported which are not shown in this Appendix. However, the necessary code in order to understand the basics of the implementation of the NN Lyapunov function for the Moore-Greitzer compressor is included in Appendix B.

```python
class CompressorSurge(DeterministicFunction):
    """CompressorSurge.

    Parameters
    ----------
    gamma : float
    B_c : float
    k_1 : float
    k_2 : float
    k_3 : float
    x_c : float
    dt : float, optional
        The sampling time.
    normalization : tuple, optional
        A tuple (Tx, Tu) of arrays used to normalize the state and
   actions. It
        is so that diag(Tx) *x_norm = x and diag(Tu) * u_norm = u.

    """

    def __init__(self, gamma, B_c, k_1, k_2, k_3, x_c, dt=1/80,
                 normalization=None):
        """Initialization; see `CompressorSurge`."""
        super(CompressorSurge, self).__init__(name='CompressorSurge')
        self.gamma = gamma
        self.B_c = B_c
        self.k_1 = k_1
        self.k_2 = k_2
        self.k_3 = k_3
        self.x_c = x_c

        self.dt = dt

        self.normalization = normalization
        if normalization is not None:
            self.normalization = [np.array(norm, dtype=config.np_dtype)
                                  for norm in normalization]
            self.inv_norm = [norm ** -1 for norm in self.normalization]


    def normalize(self, state, action):
        """Normalize states and actions."""
        if self.normalization is None:
            return state, action

```

```python
        Tx_inv, Tu_inv = map(np.diag, self.inv_norm)
        state = tf.matmul(state, Tx_inv)

        if action is not None:
            action = tf.matmul(action, Tu_inv)

        return state, action

    def denormalize(self, state, action):
        """De-normalize states and actions."""
        if self.normalization is None:
            return state, action

        Tx, Tu = map(np.diag, self.normalization)

        state = tf.matmul(state, Tx)
        if action is not None:
            action = tf.matmul(action, Tu)

        return state, action

    def linearize(self):
        """Return the linearized system.

        Returns
        -------
        a : ndarray
            The state matrix.
        b : ndarray
            The action matrix.

        """
        gamma = self.gamma
        B_c = self.B_c
        x_c = self.x_c
        k_1 = self.k_1


        A = np.array([[- gamma/(2*B_c*math.sqrt(x_c)), 1/B_c],
                      [-B_c, -k_1*B_c]],
                     dtype=config.np_dtype)

        B = np.array([[0],
                      [-B_c]],
                     dtype=config.np_dtype)

        if self.normalization is not None:
            Tx, Tu = map(np.diag, self.normalization)
            Tx_inv, Tu_inv = map(np.diag, self.inv_norm)

            A = np.linalg.multi_dot((Tx_inv, A, Tx))
            B = np.linalg.multi_dot((Tx_inv, B, Tu))

        sys = signal.StateSpace(A, B, np.eye(2), np.zeros((2, 1)))
        sysd = sys.to_discrete(self.dt)
        return sysd.A, sysd.B

    @concatenate_inputs(start=1)
    def build_evaluation(self, state_action):
        """Evaluate the dynamics."""
        # Denormalize
        state, action = tf.split(state_action, [2, 1], axis=1)
        state, action = self.denormalize(state, action)

        n_inner = 10
        dt = self.dt / n_inner
        for i in range(n_inner):
            state_derivative = self.ode(state, action)
            state = state + dt * state_derivative

        return self.normalize(state, None)[0]

    def ode(self, state, action):
        """Compute the state time-derivative.

        Parameters
        ----------
        states: ndarray or Tensor
            Unnormalized states.
        actions: ndarray or Tensor
            Unnormalized actions.

        Returns
        -------
        x_dot: Tensor
            The normalized derivative of the dynamics

        """
        # Physical dynamics
        gamma = self.gamma
        B_c = self.B_c
        x_c = self.x_c
        k_1 = self.k_1
        k_2 = self.k_2
        k_3 = self.k_3

        pressure, mass_flow = tf.split(state, 2, axis=1)



        x1_dot = (1/B_c)*mass_flow - (1/B_c)*gamma*tf.sqrt(pressure + x_c) + (1/B_c)*gamma*math.sqrt(x_c)
        x2_dot = B_c*(-k_3*tf.pow(mass_flow, 3)-
    k_2*tf.pow(mass_flow, 2) -k_1*mass_flow - pressure -        action)
        state_derivative = tf.concat((x1_dot, x2_dot), axis=1)

        # Normalize
        return state_derivative
```

Listing B.1: class CompressorSurge

```
##Learning a Lyapunov Function for Compressor Surge
#Construct and train a parameterized Lyapunov function for Compressor
    Surge.

from __future__ import division, print_function

import numpy as np
import tensorflow as tf
import safe_learning
import matplotlib.pyplot as plt
import time
import os

from utilities import (LyapunovNetwork, CompressorSurge,
                        balanced_class_weights, binary_cmap, compute_roa
    , monomials, derivative_monomials)

# Nice progress bars
try:
    from tqdm import tqdm
except ImportError:
    tqdm = lambda x: x

#User Options
class Options(object):
    def __init__(self, **kwargs):
        super(Options, self).__init__()
        self.__dict__.update(kwargs)

OPTIONS = Options(np_dtype              = safe_learning.config.np_dtype
    ,
                  tf_dtype              = safe_learning.config.dtype,
                  eps                   = 1e-8,
        # numerical tolerance
                  saturate              = True,
        # apply saturation constraints to the control input
                  use_zero_threshold    = True,
        # assume the discretization is infinitely fine (i.e., tau = 0)
                  pre_train             = True,
        # pre-train the neural network to match a given candidate in a
    supervised approach
                  dpi                   = 150,
                  num_cores             = 4,
                  num_sockets           = 1,
                  tf_checkpoint_path    = "./tmp/
    lyapunov_function_learning.ckpt")

#TensorFlow Session
#Customize the TensorFlow session for the current device.

os.environ["KMP_BLOCKTIME"]     = str(0)
os.environ["KMP_SETTINGS"]      = str(1)
os.environ["KMP_AFFINITY"]      = 'granularity=fine,noverbose,compact
    ,1,0'
os.environ["OMP_NUM_THREADS"]   = str(OPTIONS.num_cores)

config = tf.ConfigProto(intra_op_parallelism_threads  = OPTIONS.
    num_cores,
                        inter_op_parallelism_threads  = OPTIONS.
    num_sockets,
                              allow_soft_placement          = False,
                              device_count                  = {'CPU': OPTIONS
    .num_cores})

try:
    session.close()
except NameError:
    pass
session = tf.InteractiveSession(config=config)

# Set random seed to reproduce results
seed = 1
tf.set_random_seed(seed)
np.random.seed(seed)


## Dynamics

Define the nonlinear and linearized forms of the compressor surge
    dynamics.
# Constants
dt = 0.01    # sampling time

#Compressor system parameters
B_c = 0.832       #B-parameter
x_c = 0.533       #operating point for pressure flow, respective to x_1
x_d = 0.3         #operating point for mass flow, respective to x_2
gamma = 0.411     #throttle gain
H = 0.18          #coefficient
W = 0.25          #coefficient
k_1 = -1.0368     #coefficient
k_2 = 0.864       #coefficient
k_3 = 5.76        #coefficient

```

```

# State and action normalizers
pressure_max = 0.3                 # pressure
massflow_max = 0.6                 # mass flow
u_max      = 0.3               # pressure drop over CCV

state_norm = (pressure_max, massflow_max)
action_norm = (u_max,)

# Dimensions and domains
state_dim      = 2
action_dim     = 1
state_limits  = np.array([[-1., 1.]] * state_dim)
action_limits = np.array([[-1., 1.]] * action_dim)

# Initialize system class and its linearization
pendulum = CompressorSurge(gamma, B_c, k_1, k_2, k_3, x_c, dt, [
    state_norm, action_norm])
A, B = pendulum.linearize()
dynamics = pendulum.__call__

#State Discretization and Initial Safe Set
#Define a uniform discretization, and an initial known safe set as #a
    subset of this discretization.

# Number of states along each dimension
num_states = 251

# State grid
grid_limits = np.array([[-1., 1.], ] * state_dim)
state_discretization = safe_learning.GridWorld(grid_limits, num_states)

# Discretization constant
if OPTIONS.use_zero_threshold:
    tau = 0.0
else:
    tau = np.sum(state_discretization.unit_maxes) / 2

print('Grid size: {}'.format(state_discretization.nindex))
print('Discretization constant (tau): {}'.format(tau))

# Set initial safe set as a ball around the origin (in normalized
    coordinates)
cutoff_radius    = 0.1
initial_safe_set = np.linalg.norm(state_discretization.all_points, ord
    =2, axis=1) <= cutoff_radius

#Fixed Policy
#Fix the policy to the LQR solution for the linearized, discretized #
    system, possibly with saturation constraints.

Q = np.identity(state_dim).astype(OPTIONS.np_dtype)     # state cost
    matrix
R = np.identity(action_dim).astype(OPTIONS.np_dtype)    # action cost
    matrix
K, P_lqr = safe_learning.utilities.dlqr(A, B, Q, R)


policy = safe_learning.LinearSystem(- K, name='policy')
if OPTIONS.saturate:
    policy = safe_learning.Saturation(policy, -1, 1)
Q = np.identity(state_dim).astype(OPTIONS.np_dtype)     # state cost
    matrix
R = np.identity(action_dim).astype(OPTIONS.np_dtype)    # action cost
    matrix
K, P_lqr = safe_learning.utilities.dlqr(A, B, Q, R)


```

```
policy = safe_learning.LinearSystem(- K, name='policy')
if OPTIONS.saturate:
    policy = safe_learning.Saturation(policy, -1, 1)



#Closed-Loop Dynamics Lipschitz Constant
# # Policy (linear)
L_pol = lambda x: np.linalg.norm(-K, 1)

# # Dynamics (linear approximation)
L_dyn = lambda x: np.linalg.norm(A, 1) + np.linalg.norm(B, 1) * L_pol(x
    )

#LQR Lyapunov Candidate
#Define a Lyapunov candidate function corresponding to the LQR solution
     for the linearized system.

# Define the Lyapunov function corresponding to the LQR policy
lyapunov_function = safe_learning.QuadraticFunction(P_lqr)
# Approximate local Lipschitz constants with gradients
grad_lyapunov_function = safe_learning.LinearSystem((2 * P_lqr,))
L_v = lambda x: tf.norm(grad_lyapunov_function(x), ord=1, axis=1,
    keepdims=True)

# Initialize Lyapunov class
lyapunov_lqr = safe_learning.Lyapunov(state_discretization,
    lyapunov_function, dynamics, L_dyn, L_v, tau, policy,
    initial_safe_set)
lyapunov_lqr.update_values()
lyapunov_lqr.update_safe_set()


#Neural Network Lyapunov Candidate
```

```python
#Define a parameterized Lyapunov candidate function with a neural
    network.

layer_dims = [64, 64, 64]
activations = [tf.tanh, tf.tanh, tf.tanh]
lyapunov_function = LyapunovNetwork(state_dim, layer_dims, activations,
     OPTIONS.eps)

# Approximate local Lipschitz constants with gradients
grad_lyapunov_function = lambda x: tf.gradients(lyapunov_function(x), x
    )[0]
L_v = lambda x: tf.norm(grad_lyapunov_function(x), ord=1, axis=1,
    keepdims=True)

# Initialize parameters; need to use the template before parameter
    variables exist in the TensorFlow graph
temp = tf.placeholder(OPTIONS.tf_dtype, shape=[None, state_dim], name='
    states')
temp = lyapunov_function(temp)
session.run(tf.variables_initializer(lyapunov_function.parameters))

# Initialize Lyapunov class
lyapunov_nn = safe_learning.Lyapunov(state_discretization,
    lyapunov_function, dynamics, L_dyn, L_v, tau, policy,
    initial_safe_set)
lyapunov_nn.update_values()
lyapunov_nn.update_safe_set()



#TensorFlow Graph

# Dynamics
tf_states            = tf.placeholder(OPTIONS.tf_dtype, shape=[None,
    state_dim], name='states')
tf_actions          = policy(tf_states)
tf_future_states    = dynamics(tf_states, tf_actions)

# Neural network
tf_values_nn        = lyapunov_nn.lyapunov_function(tf_states)
tf_future_values_nn = lyapunov_nn.lyapunov_function(tf_future_states)
tf_dv_nn            = tf_future_values_nn - tf_values_nn
tf_threshold        = lyapunov_nn.threshold(tf_states, lyapunov_nn.tau)
tf_negative         = tf.squeeze(tf.less(tf_dv_nn, tf_threshold), axis
    =1)


# LQR
tf_values_lqr        = lyapunov_lqr.lyapunov_function(tf_states)
tf_future_values_lqr = lyapunov_lqr.lyapunov_function(tf_future_states)
tf_dv_lqr            = tf_future_values_lqr - tf_values_lqr

#True Region of Attraction
#Compute the true largest region of attraction (ROA) by forward-
    simulating the closed-loop dynamics.

closed_loop_dynamics = lambda x: tf_future_states.eval({tf_states: x})
horizon = 500
tol = 0.1
roa, trajectories = compute_roa(lyapunov_nn.discretization,
    closed_loop_dynamics, horizon, tol, no_traj=False)

#Neural Network Pre-Training
#Pre-train on a spherical Lyapunov function to make sure an initial
    safe set exists.

if OPTIONS.pre_train:
    obj = []
    level_states = state_discretization.all_points[initial_safe_set]

    # Spherical candidate
    P = 0.1 * np.eye(state_dim)
    lyapunov_function      = safe_learning.QuadraticFunction(P)
    grad_lyapunov_function = safe_learning.LinearSystem((2 * P,))
    L_v                    = lambda x: tf.norm(grad_lyapunov_function(x
), ord=1, axis=1, keepdims=True)

    # Initialize class
    lyapunov_pre = safe_learning.Lyapunov(state_discretization,
lyapunov_function, dynamics, L_dyn, L_v, tau, policy,
initial_safe_set)
    lyapunov_pre.update_values()
    lyapunov_pre.update_safe_set()

    # TensorFlow graph elements
    tf_values_pre        = lyapunov_pre.lyapunov_function(tf_states)
    tf_future_values_pre = lyapunov_pre.lyapunov_function(
tf_future_states)
    tf_dv_pre            = tf_future_values_pre - tf_values_pre

    with tf.name_scope('lyapunov_pre_training'):
        tf_losses        = tf.abs(tf_values_nn - tf_values_pre) # / tf.
stop_gradient(tf_values_pre + OPTIONS.eps)
        tf_objective     = tf.reduce_mean(tf_losses, name='objective')
        tf_learning_rate = tf.placeholder(OPTIONS.tf_dtype, shape=[],
name='learning_rate')
        optimizer        = tf.train.GradientDescentOptimizer(
tf_learning_rate)
        lyapunov_update  = optimizer.minimize(tf_objective, var_list=
lyapunov_nn.lyapunov_function.parameters)

        tf_batch_size = tf.placeholder(tf.int32, [], 'batch_size')
        tf_batch      = tf.random_uniform([tf_batch_size, ], 0,
    level_states.shape[0], dtype=tf.int32, name='batch_sample')

if OPTIONS.pre_train:
    # Test set
    test_size = int(1e3)
    idx       = tf_batch.eval({tf_batch_size: int(1e3)})
    test_set  = level_states[idx, :]

    feed_dict = {
        tf_states:         level_states,
        tf_learning_rate:  1e-1,
        tf_batch_size:     int(1e3),
    }
    max_iters = 300

    for i in tqdm(range(max_iters)):
        idx = tf_batch.eval(feed_dict)
        feed_dict[tf_states] = level_states[idx, :]
        session.run(lyapunov_update, feed_dict)

        feed_dict[tf_states] = test_set
        obj.append(tf_objective.eval(feed_dict))

    lyapunov_nn.update_values()
    lyapunov_nn.update_safe_set()

if OPTIONS.pre_train:
    fig, ax = plt.subplots(1, 1, figsize=(3, 2), dpi=OPTIONS.dpi)
    ax.set_xlabel(r'iteration')
    ax.set_ylabel(r'pre-training objective')
    ax.plot(obj, '.-r')
    plt.show()

```

```
#Neural Network Training
#Train the parameteric Lyapunov candidate in order to expand the verifiable safe set towards          .

# Save TensorFlow checkpoint for the neural network parameters
saver = tf.train.Saver(var_list=lyapunov_nn.lyapunov_function.parameters)
ckpt_path = saver.save(session, OPTIONS.tf_checkpoint_path)

with tf.name_scope('roa_classification'):
    # Target the safe level set to extend out towards
    safe_level = tf.placeholder(OPTIONS.tf_dtype, shape=[], name='c_max')
    level_multiplier = tf.placeholder(OPTIONS.tf_dtype, shape=[], name='level_multiplier')

    # True class labels, converted from Boolean ROA labels {0, 1} to {-1, 1}
    roa_labels = tf.placeholder(OPTIONS.tf_dtype, shape=[None, 1], name='labels')
    class_labels = 2 * roa_labels - 1

    # Signed, possibly normalized distance from the decision boundary
    decision_distance = safe_level - tf_values_nn

    # Perceptron loss with class weights
    class_weights = tf.placeholder(OPTIONS.tf_dtype, shape=[None, 1], name='class_weights')
    classifier_loss = class_weights * tf.maximum(- class_labels * decision_distance, 0, name='classifier_loss')

    # Enforce decrease constraint with Lagrangian relaxation
    lagrange_multiplier = tf.placeholder(OPTIONS.tf_dtype, shape=[], name='lagrange_multiplier')
    decrease_loss = roa_labels * tf.maximum(tf_dv_nn, 0) / tf.
   stop_gradient(tf_values_nn + OPTIONS.eps)

    # Construct objective and optimizer
    objective = tf.reduce_mean(classifier_loss + lagrange_multiplier *
   decrease_loss, name='objective')
    learning_rate = tf.placeholder(OPTIONS.tf_dtype, shape=[], name='
   learning_rate')
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    training_update = optimizer.minimize(objective, var_list=
   lyapunov_nn.lyapunov_function.parameters)

with tf.name_scope('sampling'):
    batch_size = tf.placeholder(tf.int32, [], 'batch_size')
    idx_range = tf.placeholder(tf.int32, shape=[], name='
   indices_to_sample')
    idx_batch = tf.random_uniform([batch_size, ], 0, idx_range, dtype=
   tf.int32, name='batch_sample')

#Initialization
#Use this cell to restore parameter checkpoints and try training again.

saver.restore(session, ckpt_path)
lyapunov_nn.update_values()
lyapunov_nn.update_safe_set()

test_classifier_loss = []
test_decrease_loss   = []
roa_estimate         = np.copy(lyapunov_nn.safe_set)

grid              = lyapunov_nn.discretization
c_max             = [lyapunov_nn.feed_dict[lyapunov_nn.c_max], ]
safe_set_fraction = [lyapunov_nn.safe_set.sum() / grid.nindex, ]

```

```
Training
# Training hyperparameters
outer_iters = 20
inner_iters = 10
horizon     = 100
test_size   = int(1e4)

feed_dict = {
    tf_states:            np.zeros((1, grid.ndim)), # placeholder
    safe_level:           1.,
    lagrange_multiplier: 1000,
    #
    level_multiplier:     1.3,
    learning_rate:        5e-3,
    batch_size:           int(1e3),
}

print('Current metrics ...')
c = lyapunov_nn.feed_dict[lyapunov_nn.c_max]
num_safe = lyapunov_nn.safe_set.sum()
print('Safe level (c_k): {}'.format(c))
print('Safe set size: {} ({:.2f}% of grid, {:.2f}% of ROA)\n'.format(
   int(num_safe), 100 * num_safe / grid.nindex, 100 * num_safe / roa.
   sum()))
print('')
time.sleep(0.5)

for _ in range(outer_iters):
    print('Iteration (k): {}'.format(len(c_max)))
    time.sleep(0.5)

    # Identify the "gap" states, i.e., those between V(c_k) and V(a *
   c_k) for a > 1
    c = lyapunov_nn.feed_dict[lyapunov_nn.c_max]
    idx_small = lyapunov_nn.values.ravel() <= c
    idx_big = lyapunov_nn.values.ravel() <= feed_dict[level_multiplier]
     * c
    idx_gap = np.logical_and(idx_big, ~idx_small)

    # Forward-simulate "gap" states to determine which ones we can add
   to our ROA estimate
    gap_states = grid.all_points[idx_gap]
    for _ in range(horizon):
        gap_states = tf_future_states.eval({tf_states: gap_states})
    gap_future_values = tf_values_nn.eval({tf_states: gap_states})
    roa_estimate[idx_gap] |= (gap_future_values <= c).ravel()

    # Identify the class labels for our current ROA estimate and the
   expanded level set
    target_idx = np.logical_or(idx_big, roa_estimate)
    target_set = grid.all_points[target_idx]
    target_labels = roa_estimate[target_idx].astype(OPTIONS.np_dtype).
   reshape([-1, 1])
    feed_dict[idx_range] = target_set.shape[0]

    # Test set
    idx_test = idx_batch.eval({batch_size: test_size, idx_range:
   target_set.shape[0]})
    test_set = target_set[idx_test]
    test_labels = target_labels[idx_test]

    # SGD for classification
    for _ in tqdm(range(inner_iters)):
        # Training step
        idx_batch_eval = idx_batch.eval(feed_dict)
        feed_dict[tf_states] = target_set[idx_batch_eval]
        feed_dict[roa_labels] = target_labels[idx_batch_eval]
        feed_dict[class_weights], class_counts = balanced_class_weights
```

```
(feed_dict[roa_labels].astype(bool))
            session.run(training_update, feed_dict=feed_dict)

            # Record losses on test set
            feed_dict[tf_states] = test_set
            feed_dict[roa_labels] = test_labels
            feed_dict[class_weights], class_counts = balanced_class_weights
(feed_dict[roa_labels].astype(bool))
            results = session.run([classifier_loss, decrease_loss],
feed_dict)
            test_classifier_loss.append(results[0].mean())
            test_decrease_loss.append(results[1].mean())

        # Update Lyapunov values and ROA estimate, based on new parameter
values
        lyapunov_nn.update_values()
        lyapunov_nn.update_safe_set()
        roa_estimate |= lyapunov_nn.safe_set

        c_max.append(lyapunov_nn.feed_dict[lyapunov_nn.c_max])
        safe_set_fraction.append(lyapunov_nn.safe_set.sum() / grid.nindex)
        print('Current safe level (c_k): {}'.format(c_max[-1]))
        print('Safe set size: {} ({:.2f}% of grid, {:.2f}% of ROA)\n'.
format(int(lyapunov_nn.safe_set.sum()),

          100 * safe_set_fraction[-1],

          100 * safe_set_fraction[-1] * roa.size / roa.sum()))

#Results
#
fig = plt.figure(figsize=(8, 3), dpi=OPTIONS.dpi, frameon=False)
fig.subplots_adjust(wspace=0.35)
plot_limits = np.column_stack((- ([pressure_max, massflow_max]), ([
    pressure_max, omega_max])))

ax = plt.subplot(121)
alpha = 1
colors = [None] * 3
colors[0] = (0, 158/255, 115/255)        # ROA - bluish-green
colors[1] = (230/255, 159/255, 0)        # NN  - orange
colors[2] = (0, 114/255, 178/255)        # LQR - blue


# True ROA
z = roa.reshape(grid.num_points)
ax.contour(z.T, origin='lower', extent=plot_limits.ravel(), colors=(
    colors[0],), linewidths=1)
ax.imshow(z.T, origin='lower', extent=plot_limits.ravel(), cmap=
    binary_cmap(colors[0]), alpha=alpha)

# # Neural network
z = lyapunov_nn.safe_set.reshape(grid.num_points)
ax.contour(z.T, origin='lower', extent=plot_limits.ravel(), colors=(
    colors[1],), linewidths=1)
ax.imshow(z.T, origin='lower', extent=plot_limits.ravel(), cmap=
    binary_cmap(colors[1]), alpha=alpha)

# # LQR
z = lyapunov_lqr.safe_set.reshape(grid.num_points)
ax.contour(z.T, origin='lower', extent=plot_limits.ravel(), colors=(
    colors[2],), linewidths=1)
ax.imshow(z.T, origin='lower', extent=plot_limits.ravel(), cmap=
    binary_cmap(colors[2]), alpha=alpha)

# Plot some trajectories
N_traj = 20
skip = int(grid.num_points[0] / N_traj)
```

```python
sub_idx = np.arange(grid.nindex).reshape(grid.num_points)
sub_idx = sub_idx[::skip, ::skip].ravel()
sub_trajectories = trajectories[sub_idx, :, :]
sub_states = grid.all_points[sub_idx]
for n in range(sub_trajectories.shape[0]):
    x = sub_trajectories[n, 0, :] * (pressure_max)
    y = sub_trajectories[n, 1, :] * (massflow_max)
    ax.plot(x, y, 'k--', linewidth=0.25)
sub_states = grid.all_points[sub_idx]
dx_dt = (tf_future_states.eval({tf_states: sub_states}) - sub_states) / dt
dx_dt = dx_dt / np.linalg.norm(dx_dt, ord=2, axis=1, keepdims=True)
ax.quiver(sub_states[:, 0] * (pressure_max), sub_states[:, 1] * (massflow_max), dx_dt[:, 0], dx_dt[:, 1],
          scale=None, pivot='mid', headwidth=3, headlength=6, color='k')

ax.set_aspect(pressure_max / massflow_max / 1.2)
ax.set_xlim(plot_limits[0])
ax.set_ylim(plot_limits[1])
ax.set_xlabel(r'plenum pressure coefficient  ')
ax.set_ylabel(r'mass flow coefficient')


proxy = [plt.Rectangle((0,0), 1, 1, fc=c) for c in colors]
legend = ax.legend(proxy, [r'$\mathcal{S}_\pi$', r'NN', r'LQR'], loc='upper right')
legend.get_frame().set_alpha(1.)


# Plot safe growth over the iterations
ax = plt.subplot(222)
ax.plot(c_max, '.-', color=colors[1])
ax.set_ylabel(r'safe level $c_k$')
ax.set_ylim([0, None])
plt.setp(ax.get_xticklabels(), visible=False)
roa_fraction = roa.sum() / roa.size
ax = plt.subplot(224)
ax.plot(np.array(safe_set_fraction) / roa_fraction, '.-', color=colors
    [1])
ax.set_ylabel(r'fraction of $S_\pi$')
ax.set_ylim([0, 1])
ax.set_xlabel(r'safe level set update iteration $k$')
plt.show()
```

Listing B.2: Learning a Lyapunov Function for Compressor Surge