Ben-Gurion University of the Negev

The Faculty of Engineering Sciences

The Department of Mechanical Engineering

Model-Free Gain Scheduling for Linear Parameter Varying (LPV) Systems

Thesis submitted in partial fulfillment of the requirements

for the Master of Sciences degree

Gilad Shaul

Under the supervision of **Shai Arogeti**

March 2025

Ben-Gurion University of the Negev
The Faculty of Engineering Sciences
The Department of Mechanical Engineering

Model-Free Gain Scheduling for Linear Parameter Varying (LPV) Systems

Thesis submitted in partial fulfillment of the requirements
for the Degree of

Master of Sciences

in

Mechatronics Engineering

by

Gilad Shaul

Under the supervision of **Shai Arogeti**

Signature of student: _____ Date: 23/03/2025

Signature of supervisor: _____ Date: 23/03/2025

Signature of chairperson of the
committee for graduate studies: _____ Date: _____

March 2025

Abstract

This thesis introduces a novel data-driven framework for designing suboptimal controllers for Linear Parameter Varying (LPV) systems in a model-free context. Three types of controllers are considered, which are, gain scheduling, robust, and mixed. Traditionally, LPV systems are solved using Linear Matrix Inequalities (LMIs) when the system model is well-defined. However, obtaining accurate models for complex, real-world systems can be challenging or infeasible. The proposed approach addresses this limitation by leveraging evolutionary optimization algorithms, specifically Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), to identify suboptimal control gains directly from input-output data without relying on explicit system dynamics.

The core innovation lies in formulating the control problem as the minimization of a set of model-free Riccati equations, which is solved using GA and PSO. The key result is a set of suboptimal gains that form a convex set in the parameter space. This convex structure enables the application of gain scheduling. As the system parameters vary within known bounds, the controller gains are updated through interpolation within the convex set. This allows the controller to adapt to changing dynamics in real-time without re-optimization. Additionally, we modify the same suggested design infrastructure to generate robust controllers for cases where the LPV parameters are not given on the fly. Since both controller types are based on a similar design infrastructure, their combinations (i.e., mixed) are also possible for cases where LPV parameters are only partially given.

Comprehensive simulations and performance analysis on a vehicle lane-keeping system (LKS) case study validate the effectiveness of the proposed approach. The

GA and PSO-based gain-scheduled controllers consistently outperform the nominal fixed-gain LQR controller and the robust gains, exhibiting superior tracking and stability under parameter variations.

This study is significant in two ways. From a control theory point of view, it extends optimal control principles to model-free LPV systems, bridging the gap between the well-established domain of optimal control and the challenge of designing control systems from data. From a design perspective, it offers data-efficient and computationally feasible methods for implementing data-driven controllers in complex systems.

This thesis opens new avenues for research on data-driven adaptive control, evolutionary optimization for controller tuning, and the inference of approximate LPV models from data. It offers a promising alternative to LMI-based approaches for situations where explicit system dynamics are unavailable, paving the way for broader application of gain scheduling and robust control in real-world systems.

Acknowledgements

Completing this thesis has been made possible through the invaluable support and assistance of several individuals and organizations, to whom I express my sincere gratitude.

First and foremost, I am deeply indebted to my supervisor, Dr. Shai Arogeti, for his professional and knowledgeable guidance in the physical, computational, and numerical fields. His significant contribution to this research, from his laboratory computing equipment to countless hours of meetings and advice at any time, has been remarkable. I am incredibly grateful for his meticulous supervision while writing this scientific manuscript.

I also want to take a moment to thank my incredible team in the Army Reserve for their patience and support over the past year. You've been forced to endure my endless talk about ideas and concepts that probably made no sense to you, yet you still listened (or at least pretended to). I truly appreciate each of you and am grateful to be part of such a dedicated and resilient team.

Lastly, I want to express my deep gratitude to my amazing wife, Hagar. Words cannot fully capture how much I appreciate you. Over the past year, you have

single-handedly raised our firstborn, Yuval, brought our second child, Gili, into the world, and maintained our home while I spent nearly 200 days away serving in the Army Reserve. Throughout everything, you not only carried the weight of our family with unwavering strength and love, but you also found the energy to support me in completing this work. Your resilience, sacrifice, and constant love are more than I could ever deserve, and I am endlessly grateful for you.

Contents

List of Figures

List of Tables

List of Abbreviations

LPV	Linear Parameter Varying
LMIs	Linear Matrix Inequalities
GS	Gain Scheduling
GA	Genetic Algorithm
PSO	Particle Swarm Optimization
LKS	Lane Keeping System
LQR	Linear Quadratic Regulator
RADP	Robust Adaptive Dynamic Programming
CARE	Continuous time Algebraic Riccati Equation
EV	Estimation Value
LTI	Linear time-invariant
AWDV	Adaptive Weighted Delay Velocity
TVAC	Time Varying Accelerating Constant

List of Symbols

$x(t)$	State Vector
$u(t)$	Control Input
J	Cost Function
$V(x)$	Value Function
H	Hamiltonian
∇	Gradient
$\mathbb{R}^n$	n-Dimensional Euclidean Space of Real Numbers
A, B	System Matrices
Q, R	Weight Matrices
P	Solution Matrix of Riccati Equation
K	Control Gain
$\otimes$	Kronecker Product
N	Transformation matrix
$\| \bullet \|_F$	Frobenius norm
V_x	Vehicle's velocity in X axis
m	Vehicle's mass

F_{yr},F_{yf}	Force's equations on the front and rear tyres
C_{ar},C_{af}	Cornering stiffness for the front and rear tyres
l_r,l_f	The distance from the center of the vehicle's mass to the front and rear wheels
I_z	Vehicle's moment of inertia around Z axis
r	Turning radius
δ	Front tires turn angle
e_1	Lateral position error with respect to the road
e_2	Yaw angle error with respect to the road
$\dot{\psi}_{des}$	Desired raw rate
$rand$	Random Number
w	Inertia Coefficient
c_1, c_2	Acceleration Coefficients

Chapter 1

Introduction and Theoretical Background

1.1 Introduction

Achieving optimal performance in uncertain and dynamic environments is a significant challenge in modern control systems. Many real-world systems, particularly in the automotive and aerospace industries, are characterized by complex, nonlinear, and time-varying behaviors that are difficult to model accurately. Linear Parameter Varying (LPV) systems represent a robust framework for describing such systems, where the dynamics depend on varying parameters. Some data-driven algorithms, such as Robust Adaptive Dynamic Programming (RADP) [2], have been developed to manage these uncertainties. While these algorithms provide reasonable solutions for finding the optimal result in a fixed model-free system, they are ineffective if any

plant parameters change unexpectedly.

Traditionally, LPV systems can be analyzed and solved using a set of LMIs [3], usually for gain scheduling or robust implementation. Gain scheduling is a well-established approach for controlling LPV systems, where the controller gains are adjusted based on the current operating point. It has been widely applied in various domains, such as automotive systems [4]. However, an LMI design relies on the availability of an explicit mathematical model of the system, which may not always be feasible or practical to obtain for complex, real-world systems.

Recent studies highlight a significant limitation of conventional LPV gain scheduling, such as the need for an accurate plant model. In response, considerable progress has been made in developing model-free LPV gain scheduling methods, categorized into data-driven approaches and machine learning (ML) techniques. Data-driven approaches aim to optimize controller parameters from experimental data. For example, Miller and Sznaier (2022) introduced Quadratic Matrix Inequalities (QMIs) to capture all LPV plants consistent with measured data and to synthesize a gain-scheduling controller that stabilizes all such plants [5]. In ML methods, especially reinforcement learning (RL), the controller (or policy) is typically parameterized and improved through interaction with the system or offline data rather than derived from an explicit model. For example, Bao and Velni proposed a policy-gradient reinforcement learning scheme for LPV systems. Their method treats the nonlinear plant as LPV and uses an off-policy gradient algorithm to learn a state-feedback policy that implicitly adjusts gains as scheduling variables change[6]. However, these approaches have limitations regarding data efficiency, computational tractability, and the ability to provide stability guarantees.

This thesis addresses the challenge of designing gain-scheduled and robust controllers for model-free LPV systems by proposing a novel approach combining data-driven optimization and convex interpolation. The primary objective is to develop a framework to efficiently determine suboptimal control gains for systems with uncertain dynamics while ensuring smooth and stable operation under parameter variations.

The specific contributions of this thesis are as follows:

1. A data-driven methodology for solving Riccati equations, which, unlike ADP approaches, does not require an initial stabilizing controller.

2. A Comparison between PSO and GA as a means for solving Riccati equations.

3. A methodology for designing suboptimal gain scheduling and robust controllers for LPV systems in a model-free context. This methodology is based on a simultaneous minimizing of a set of Riccati equations, each representing a boundary point in the parameter space.

4. Implementation and comparative analysis of the proposed design methods for vehicle lane-keeping control.

The proposed algorithm minimizes the real-time computational burden by performing most of the calculations offline, with only the final computations executed online. The validation process involves computer simulations, data-driven gain optimization, and comparative performance analysis, demonstrating the applicability and potential benefits of the proposed approach in real-world control systems.

To evaluate the performance of the proposed gain-scheduled and robust controllers, they were compared to a nominal controller under parameter-varying scenarios. This

comparison allows a rigorous evaluation of the controllers' ability to maintain optimal performance and ensure system stability, even as the operating conditions change over time.

The remainder of this thesis is organized as follows. This chapter provides a comprehensive review of related work, particularly gain scheduling for LPV model-free systems. Chapter 2 explains the relevant theoretical background and the foundation on which the thesis is built, covering LPV systems, data-driven control, evolutionary optimization, and gain scheduling. Chapter 3 presents the algorithm development, detailing the problem formulation, and the optimization algorithms for gain scheduling and robust approaches. Chapter 4 describes the vehicle lane-keeping case study and presents the simulation results and performance analyses. Finally, Chapter 5 concludes the thesis, discussing the implications of the findings, limitations of the approach, and future research directions.

This thesis introduces a novel data-driven framework for designing gain-scheduled and robust controllers for LPV systems in a model-free context, using evolutionary optimization. The proposed approach represents a significant advancement in adaptive control, offering a promising alternative to traditional model-based methods and paving the way for broader application of gain scheduling in real-world systems with uncertain dynamics. Because the robust design suggested here is a direct derivative of the suggested gain scheduling design framework, the following is mainly focused on gain scheduling, to highlight relevant aspects of this research framework.

1.2 Related Work

LPV models have become an essential tool for representing nonlinear systems in a linear framework. By allowing the system matrices to depend on measurable scheduling parameters (such as speed, mass, or joint angles), LPV models enable the application of linear control design techniques over a wide range of operating conditions. In this context, LMIs have emerged as a popular method for analysis and controller synthesis because of their convexity properties, which guarantee the existence of a standard Lyapunov function and provide a systematic means to ensure robust stability and performance.

A particularly effective strategy within the LPV framework is gain scheduling. In gain scheduling, controllers are designed at the extreme vertices of a polytopic LPV model and then interpolated across the operating range. This approach ensures that the controller adapts in real-time to the varying dynamics of the system, thereby offering robustness and improved performance. The following sections review several recent contributions that combine LMI-based synthesis with gain scheduling for LPV systems in different application domains.

1.2.1 LMI-Based Controller Synthesis for LPV Systems

LMIs serve as the backbone for many modern robust control design techniques. They recast the controller design problem into a set of convex constraints. For LPV systems, this typically involves formulating LMI conditions that guarantee stability for all admissible parameter variations. For instance, in the polytopic LPV approach, the system is represented as a convex combination of a finite number of

LTI systems corresponding to the vertices of a polytope defined by the scheduling parameters. Controllers are designed for each vertex via LMI-based methods and then interpolated to yield a gain-scheduled controller that adapts smoothly across the parameter range. Tran et [7] present an LMI-based polytopic LQR cruise controller for an autonomous vehicle. Evren and Unel [8] develop a polytopic quasi-LPV model for a pan-tilt robotic system and synthesize an LQR controller via LMIs. Alcala et [9] take a further step by addressing both longitudinal and lateral control in autonomous guidance. They decompose the control design into two separate LMI-LQR problems—one for the kinematic model and one for the dynamic model. By integrating LMIs with a polytopic LPV framework, engineers can effectively address the challenges associated with system nonlinearities and parameter variations. The LMI-based synthesis ensures that the resulting controllers satisfy rigorous stability and performance criteria at each vertex of the polytopic representation. Moreover, gain scheduling through LMIs—implemented by interpolating vertex controllers—ensures that control gains continuously adapt to variations in scheduling parameters. This adaptive mechanism is particularly valuable in applications such as autonomous vehicles and robotic systems, where dynamic behavior can vary significantly across the operating envelope. Nevertheless, a specific model design is required to resolve the corresponding optimization problem.

1.2.2 Data-Driven Gain Scheduling for LPV Systems

As mentioned in the previous section, traditional gain scheduling methods rely on obtaining an accurate mathematical model of the plant. However, system identification is either expensive or infeasible in many practical scenarios—such as those

involving complex or poorly understood dynamics. To address these challenges, an increasing amount of research has turned to data-driven methods that directly design gain-scheduled controllers from measured data, eliminating the need for explicit model identification. A notable contribution in this area is the work by Miller and Sznaier, who propose a data-driven approach that synthesizes gain-scheduled controllers using Quadratic Matrix Inequalities (QMIs) [5]. Their method considers LPV plants with an affine parameter dependency and avoids the bilinearity that typically arises when dealing with varying parameter conditions. The approach provides both continuous-time and discrete-time formulations by formulating the control synthesis problem in terms of LMIs and performing vertex enumeration of the parameter polytope. In a complementary direction, Mejari et al. [10] focus on the simultaneous computation of robust invariant sets and gain-scheduled controllers for LPV systems. Their work leverages data-driven techniques to derive polytopic invariant sets directly from a single trajectory of input–state–scheduling data. Modares and Sadati propose a direct data-driven safe gain-scheduling framework that stabilizes the LPV system and enforces safety constraints [11]. While much of the work in data-driven gain scheduling has focused on theoretical developments for generic LPV systems, recent applications also demonstrate the practical impact of these ideas. Adıgüzel et [12] propose a gain scheduling attitude controller for quadrotor UAVs that is enhanced by a neural network (NN) supervisor. Xia et al. [13] propose a data-driven method for online gain scheduling in distributed secondary controllers for microgrids with time delays. The authors develop a time-delayed small-signal model for stability analysis and implement a constrained soft actor-critic (SAC) algorithm to learn optimal gain scheduling policies. All of the progress presented in this section has been made during the last two years, which

suggests a growing interest in data-driven control design approaches for LPV systems. Unlike the previous methods described above, the method presented here is accessible and intuitive. Being based on evolutionary optimization, it can explore non-convex solution spaces more freely, allowing for potential extensions to nonlinear and constrained design problems. In contrast to LMI-based methods, the complexity of evolutionary methods is more influenced by the population size rather than the number of unknowns, allowing for easier scalability.

1.3 Motivation for the Research

As discussed earlier, numerous applications are available for model-free gain scheduling LPV control. This has led to increased research on this topic in recent years. There are multiple approaches when dealing with gain scheduling in LPV systems, some model-based and some model-free. While the model-based approach is pretty established, the model-free approach has many areas left to be explored. The advantage of the data-driven design framework proposed here stems from the fact that it is based on a well-established theoretical basis (represented by polytopes) similar to that of its model-based counterpart.

In this study, we consider LPV systems with n varying parameters by constructing a polytope with 2^n vertices and formulating a model-free Riccati equation at each vertex. We define fitness functions from the Frobenius norm of the residuals of model-free Riccati equations. Using recording data of state and input variables, we employ evolutionary algorithms—PSO and GA—to find the matrices that minimize this fitness function. Subsequently, we further optimize an additional gain matrix

for each vertex by minimizing an augmented fitness function. From this result, we can formulate 2^n sub-optimal control gains corresponding to the vertices of a convex polytope, which can be scheduled across the entire operating range, all without the need for explicit system identification. Instead of solving often conservative convex optimization problems that can become computationally burdensome when the number of vertices increases, our method uses PSO and GA. These evolutionary algorithms can effectively explore nonconvex search spaces and find global optima in a less conservative manner, even when the fitness landscape is complex. By applying GA and PSO to solve the model-free Riccati equation, the method finds control gains directly from input–output data. This contrasts with traditional methods that use computationally intensive semi-definite programming formulations, and it provides flexibility across varying operating conditions. The use of PSO and GA allows us to easily adjust the fitness criteria or incorporate additional constraints without reformulating complex convex programs. This flexibility is particularly beneficial in practical applications where the design problem includes various design considerations.

Our comprehensive comparison study of the suggested gain-scheduling and robust design strategies—as well as differences observed for GA and PSO—yields insights into their complementary performance across different scenarios. This detailed evaluation aims to assist practitioners in selecting algorithms that meet their specific needs.

Chapter 2

Theoretical Background

This chapter provides a comprehensive study of the theoretical subjects that form the foundation for the research performed in this thesis. Key concepts and mathematical formulations are introduced, spanning topics such as optimal control theory, evolutionary optimization algorithms, and gain scheduling techniques for linear parameter varying systems. The aim is to equip the reader with the necessary background knowledge to fully understand and appreciate the novel contributions presented in subsequent chapters.

2.1 Linear Quadratic Regulator

The theory of optimal control concerns operating a dynamic system at a minimum cost. The case where the system dynamics are described by a set of linear differential equations and the cost is defined by a quadratic function is called the linear–quadratic (LQ) problem. One of the main results in the theory is that the

solution is provided by the linear–quadratic regulator (LQR), a feedback controller whose equations are given below. Although there are several methods from which the optimal controller can be developed, the method described in the following is of special importance as it will later be used to develop suboptimal controllers. Let the process model be,

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.1}$$

where $x(t) \in \mathbb{R}^n$ and $u(t) \in \mathbb{R}^m$. It is desirable to find u such that it minimizes the cost function

$$J = \int_0^\infty (x^T Q x + u^T R u) dt, \quad Q \geq 0, R > 0 \tag{2.2}$$

This is equivalent to minimizing a value function of the form,

$$V(x) = \int_t^\infty (x^T Q x + u^T R u) d\tau \tag{2.3}$$

2.1.1 Developing the LQR Optimal Controller from the Comparison Theorem

Assume two Riccati equations,

$$\begin{aligned} A^T P_1 + P_1 A - P_1 \overline{R} P_1 + \overline{Q}_1 = 0 \\ A^T P_2 + P_2 A - P_2 \overline{R} P_2 + \overline{Q}_2 = 0 \end{aligned} \tag{2.4}$$

where $\overline{Q}_2 \geq \overline{Q}_1 \geq 0$. Additionally, $\overline{R} \geq 0$ and $A - \overline{R} P_2$ is Hurwitz (i.e., all eigenvalues have a negative real part), then, $P_2 \geq P_1$. In other words, the equation with the smaller $\overline{Q}$ is the equation with the smaller P solution.

Assume $u(t) = -Kx(t)$ is a state feedback stabilizing controller (not necessarily optimal). We are interested in K that minimizes the cost function (2.2). The closed-loop with the state feedback controller is, $\dot{x}(t) = (A - BK)x(t)$, $x(0) = x_0$, and by solving the state equations, $x(t) = e^{(A-BK)t}x_0$, $x(t)$ and $u(t)$ can be substituted in (2.2). Then the cost with the state feedback controller,

$$\begin{aligned} J &= \int_0^\infty \left(x^T(t)Qx(t) + u^T(t)Ru(t)\right) dt \\ &= \int_0^\infty \left(x^T(t)Qx(t) + x^T(t)K^TRKx(t)\right) dt \\ &= \int_0^\infty \left(x^T(t)(Q + K^TRK)x(t)\right) dt \\ &= x_0^T \left(\int_0^\infty \left(e^{(A-BK)^Tt}(Q + K^TRK)e^{(A-BK)t}\right) dt\right) x_0 \end{aligned} \tag{2.5}$$

For the stabilizing controller $u = -Kx$ we can define,

$$P = \int_0^\infty \left(e^{(A-BK)^Tt}(Q + K^TRK)e^{(A-BK)t}\right) dt \tag{2.6}$$

then,

$$J = x_0^T P x_0 \tag{2.7}$$

This result clearly shows that P is a matrix representing the cost corresponding to K. Hence, the optimal controller is the one with the smallest $P > 0$. From (2.7) it can be easily shown that (2.6), for a stabilizing K, is the solution of the following Lyapunov equation,

$$(A - BK)^T P + P(A - BK) = -Q - K^TRK \tag{2.8}$$

This Lyapunov equation can be considered as a value equation (it computes the value of K). Recall that we are interested in the controller K with the minimal

$P > 0$. Rearrange the Lyapunov equation to get the following,

$$(A - BK)^T P + P(A - BK) + Q + K^T RK = 0 \tag{2.9}$$

By completing the square of (2.9), the equation can be reformulated as,

$$A^T P + PA - PBR^{-1}B^T P + (R^{-1}B^T P - K)^T R(R^{-1}B^T P - K) + Q = 0 \tag{2.10}$$

From the comparison theorem, the gain needed for a minimal P is $K = R^{-1}B^T P$. From substituting $K = R^{-1}B^T P$ in (2.10) we get the Riccati equation from which P can be solved.

$$PA + A^T P - PBR^{-1}B^T P + Q = 0 \tag{2.11}$$

2.2 Kronecker Product

In mathematics, the Kronecker product, denoted by $\otimes$, operates on two matrices of arbitrary size, resulting in a block matrix. It is (in a sense) a generalization of the tensor product (denoted by the same symbol) from vectors to matrices. The Kronecker product is to be distinguished from the usual matrix multiplication, which is an entirely different operation. For two matrices $A \in \mathbb{R}^{\mathsf{m}\times\mathsf{n}}$ and $B \in \mathbb{R}^{\mathsf{p}\times\mathsf{q}}$ the

Kronecker product is

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix} \tag{2.12}$$

The Kronecker product has many useful properties [14]. An important for our development is

$$\mathsf{vec}(ABC) = (C^T \otimes A)\mathsf{vec}(B) \tag{2.13}$$

With this, one transforms a set of matrix equations into a set of scalar equations. To do so, we define two essential operators.

2.2.1 Vec

A transformation action that represents a matrix as a vector. For a matrix $A \in \mathbb{R}^{\mathsf{m}\times\mathsf{n}}$ represent by,

$$[A] = \begin{bmatrix} | & | & | & | \\ v_1 & v_2 & \cdots & v_n \\ | & | & | & | \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \tag{2.14}$$

the $\mathsf{vec}(A) \in \mathbb{R}^{(\mathsf{m}+\mathsf{n})\times 1}$ is defined as,

$$\mathsf{vec}(A) = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} a_{11} \\ \vdots \\ a_{m1} \\ a_{12} \\ \vdots \\ a_{m2} \\ \vdots \\ a_{1n} \\ \vdots \\ a_{mn} \end{bmatrix} \tag{2.15}$$

2.2.2 Svec

This is similar to the vec operator presented at 2.2.1, but this operator is for symmetric matrix ($A = A^T$) while $A \in \mathbb{R}^{\mathsf{n}\times\mathsf{n}}$. The $\mathsf{svec}(A) \in \mathbb{R}^{(\mathsf{n}(\mathsf{n}+1)/2)\times 1}$ operator transforms the entries of A to a vector that does not show repeated values from the

matrix symmetries. The following is an example,

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \rightarrow \mathsf{svec}(A) \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{22} \\ a_{23} \\ a_{33} \end{bmatrix} \tag{2.16}$$

2.2.3 Vec to Svec transformation

Here we define a matrix $N \in \mathbb{R}^{\mathsf{n}^2 \mathsf{x} (\mathsf{n}^2+\mathsf{n})/2}$, to transform between Vec and Svec matrix representations. It is defined by,

$$\mathsf{vec}(A) = N\mathsf{svec}(A) \tag{2.17}$$

For example, if $[A] \in \mathbb{R}^{4\times4}$, then $[N] \in \mathbb{R}^{16\times10}$ is given as follow,

$$N = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.18}$$

2.3 Frobenius norm

The Frobenius norm is a matrix norm of a $A \in \mathbb{R}^{\mathsf{mxn}}$ defined as the square root of the sum of the absolute squares of its elements [15]

$$\| A \|_F \equiv \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} | a_{ij} |^2} \tag{2.19}$$

The Frobenius norm can be considered as a generalization of the well-known vector Euclidean norm, and it is also equal to the square root of the matrix trace of AA^H, where A^H is the conjugate transpose,

$$\| A \|_F \equiv \sqrt{Tr(AA^H)} \tag{2.20}$$

2.4 Lyapunov Stability of a Linear System

Lyapunov stability theory is commonly used to show the stability of linear and nonlinear systems. It is based on a Lyapunov function $V(x)$, often quadratic of the form $V(x) = x^T Px$, and exploration of its behavior along the system's trajectories. Consider an autonomous system described by

$$\dot{X} = f(X) \tag{2.21}$$

Suppose that there exists a scalar function $V(x)$ which for some real number $\varepsilon > 0$, satisfies the following properties for all x in the region $\| x \| \leq \varepsilon$:

- $V(x)$ is positive definite function i.e. $V(x) > 0; x \neq 0$ and $V(0) = 0$.
- $V(x)$ has continuous partial derivatives with respect to all components of x.

The equilibrium state $x_e = 0$ of the system given (2.21) is asymptotically stable if $\dot{V}(x) < 0 \; \forall \; x \neq 0$ i.e., $\dot{V}(x)$ is a negative definite function.

2.5 Quadratic Stability of a Linear System

Quadratic stability is a specific form of Lyapunov stability. It provides a robust framework for assessing the stability of systems subject to uncertainties or parameter variations. The concept is based on a quadratic Lyapunov function, which serves as a global measure of system stability.

2.5.1 Definition and Context

A system is said to be quadratically stable if a positive definite matrix P exists, such that a quadratic Lyapunov function, $V(x) = x^T P x$, satisfies the stability conditions across the entire state space. For a linear time-invariant (LTI) system described by:

$$\dot{x}(t) = Ax(t) \tag{2.22}$$

The quadratic Lyapunov function satisfies:

$$\dot{V}(x) = \frac{d}{dt}(x^T P x) = x^T (A^T P + PA) x < 0, \quad \forall x \neq 0 \tag{2.23}$$

Where $A^T P + PA < 0$ (a negative definite matrix) ensures asymptotic stability. Quadratic stability benefits systems with uncertain or time-varying parameters, such as in robust control design or gain scheduling. Instead of proving stability for every possible variation in system parameters, quadratic stability ensures a single Lyapunov function suffices for all admissible uncertainties. For example, consider a family of systems defined by:

$$\dot{x}(t) = A(\triangle)x(t) \tag{2.24}$$

where $A(\triangle)$ represents the system matrix dependent on a parameter $\triangle$ within a bounded set. If there exists a positive definite matrix P such that:

$$A^T(\triangle)P + PA(\triangle) < 0, \quad \forall\triangle \tag{2.25}$$

the system is quadratically stable. This guarantees stability for all configurations of the parameter $\triangle$.

2.6 Genetic Algorithm

The genetic Algorithm (GA) is a powerful tool in computer science. It belongs to the evolutionary algorithm category, which harnesses the principles of natural selection to tackle a wide range of constrained and unconstrained optimization problems. The GA operates by iteratively modifying a population of individual solutions. In each iteration, the algorithm selects individuals from the current population to serve as parents and uses them to generate the children for the next generation. This process

of successive generations leads the population to evolve towards an optimal solution. [16]

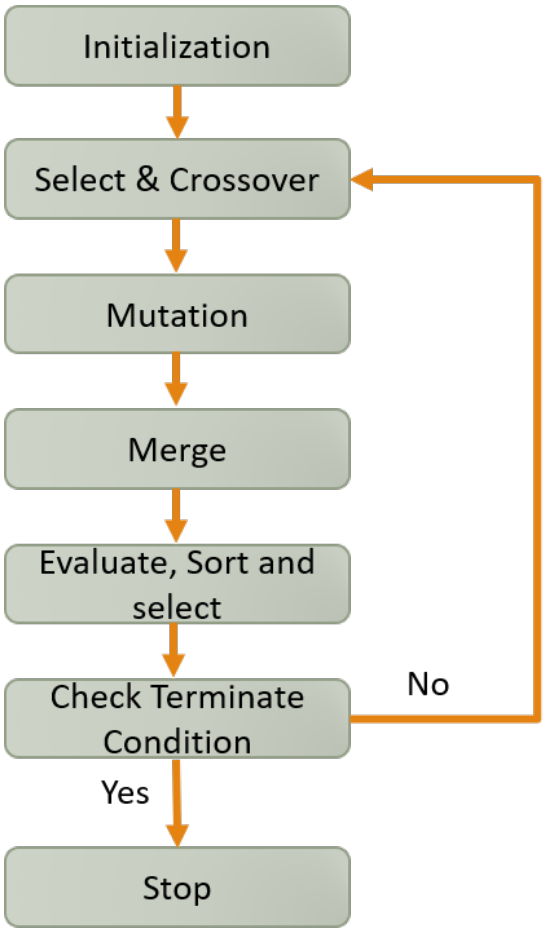


Figure 2.1: GA Flow Chart

During the Initialization step, the genetic algorithm starts by generating a random initial population. In the Select and Crossover step, the algorithm selects a group of individuals from the current population who exhibit the best fitness value and uses them to generate a new population, commonly referred to as children. In the Mutation and Merge steps, the algorithm mutates the population based on a defined mutation rate and then merges between all the populations available. In the

Evaluate and Sort step, the algorithm evaluates and sorts the population based on each fitness value and terminates the rest. In the next step, the algorithm checks if the population meets the termination condition. If it does, the algorithm stops. Otherwise, it returns to the Select and Crossover step.

2.6.1 Uniform Crossover

There are several techniques to perform a crossover [17]. Here, the technique called "Uniform Crossover" will be explain. In uniform crossover, we have two parents and two offspring represented as follows

$$\begin{aligned} x_1 &= (x_{11}, x_{12}, ..., x_{1n}), \quad y_{1i} = (y_{11}, y_{12}, ..., y_{1n}) \\ x_2 &= (x_{21}, x_{22}, ..., x_{2n}), \quad y_{2i} = (y_{21}, y_{22}, ..., y_{2n}) \end{aligned} \tag{2.26}$$

Where x_1, x_2 representing the parents and y_1, y_2 representing the offspring. Giving a random vector of binary numbers called α so $\alpha_i \in \{0, 1\}$

$$\alpha = (\alpha_1, \alpha_2, ..., \alpha_n) \tag{2.27}$$

So, the value of the offspring will be calculated like this

$$\begin{aligned} y_{1i} &= \alpha_i x_{1i} + (1 - \alpha_i) x_{2i} \\ y_{2i} &= (1 - \alpha_i) x_{1i} + \alpha_i x_{2i} \end{aligned} \tag{2.28}$$

2.6.2 Mutation

In Genetic Algorithms (GA), the mutation process introduces diversity into the sampled population by using mutation operators that prevent the population of chromosomes from becoming too similar. This slows the convergence to the global optimum, but reduces the chance of convergence to local minima. In a classical GA algorithm, individuals are represented as binary vectors. Assume that we have a binary representation of an individual from the population $x = (x_1, x_2, ..., x_n)$ while $x_i \in [0, 1]$ with a random index $j \in [1, 2, ..., n]$, so the mutated individual $x' = (x'_1, x'_2, ..., x'_n)$ will be calculated as follows

$$x'_i = \begin{cases} x_i - 1 & i \neq j \\ x_i & i = j \end{cases} \tag{2.29}$$

The mutation rate can be changed to meet the solution requirements. In this thesis, individuals are represented by real numbers. Hence, the mutation strategy used here is different and belongs to a group of methods called Differential Evolution (DE). In this strategy, one randomly chooses three distinct individuals x_1, x_2, x_3 from the population and generates a mutant vector as:

$$mutant\_vector = x_1 + F(x_2 - x_3) \tag{2.30}$$

F is a scaling factor called the "differential weight." This parameter controls how aggressively the algorithm explores the solution space by scaling the difference between two randomly chosen population vectors. If F is larger, the mutation will produce more significant changes in the mutant vector, promoting greater diversity

and potentially escaping local optima. However, if F is too large, the mutation step introduces substantial perturbations to the population vectors and can cause instability in the search process or prolong convergence. If F is too small, the changes are mild, and the algorithm can converge prematurely. Thus, choosing an appropriate value for F is crucial to balancing exploration and exploitation in the Differential Evolution optimization process. This is one of the most basic and widely used mutation strategies in Differential Evolution. It is known for its simplicity and decent exploratory capabilities.

2.6.3 Merge, Sorting and Selection

Assuming we have the parents' population, represented in a vector of $nx1$, and the offspring population, represented in a vector of $mx1$. After combining these two populations, we get a vector of size $(n + m)x1$. But to continue running the algorithm, sorting must be performed. First, a calculation of each fitness value is performed. Then, the merge population vector is sorted in increasing order from the individual with the lowest fitness value at the top to the individual with the highest fitness value at the bottom. Then, it comes to the selection stage, where all the population in the vector with an index higher than the initial population size is exterminated.

2.7 Particle Swarm Optimization

Particle swarm optimization, a population-based stochastic optimization algorithm, stands out for its inspiration from the collective behavior of animals, such as flocks of

birds or schools of fish. Unlike other algorithms, PSO scatters particles throughout the solution space, allowing them to update their position and velocities based on data from other particles. The swarm in PSO is not confined to a specific area but constantly explores the entire solution space for the optimal solution. Moreover, particles in PSO exhibit a unique ability to maintain stable movement in the search space while adapting to environmental changes by altering their movement mode[18].

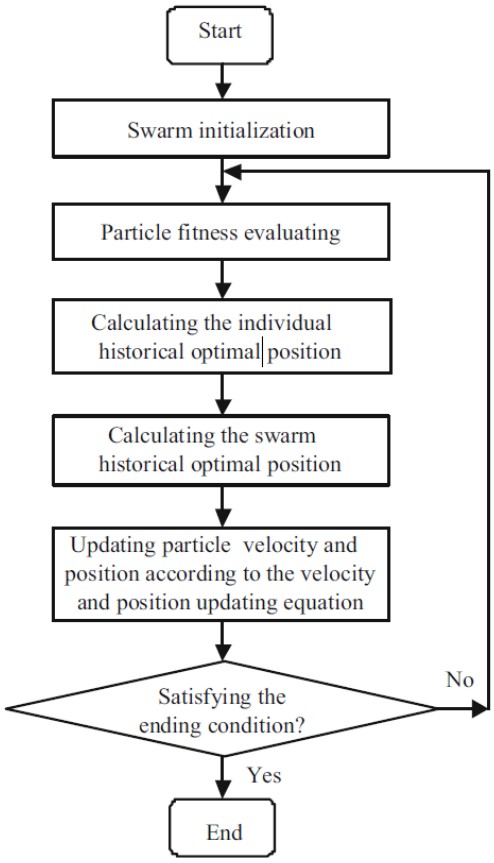


Figure 2.2: PSO Algorithm Flow Chart

Like the GA, the PSO also starts with a random initial population. The difference is that in the PSO, each particle data is represented with three vectors, the position $\vec{x_i}(t) \in X$, the velocity $\vec{v_i}(t)$ and the particle's personal best position $\vec{p_i}(t)$, the

velocity describes the movement of particle i, in terms of direction, distance and step size. In the PSO, each particle is not alone, it communicates with the other particles in the swarm, learning from them to find the best solution. So, in addition to the three vectors, the swarm as a whole remembers the position of the particle with the best fitness value in all the iterations and updates it along the way $\vec{g_i}(t)$. After the initialization, each particle is evaluated using the fitness equation, then updating each particle's personal best and the swarm global best. The mathematical model describing the updating of a particle in PSO is as follows

$$\begin{cases} \vec{v_i}(t+1) = w\vec{v_i}(t) + c_1\big(\vec{p_i}(t) - \vec{x_i}(t)\big) + c_2\big(\vec{g}(t) - \vec{x_i}(t)\big) \\ \vec{x_i}(t+1) = \vec{x_i}(t) + \vec{v_i}(t+1) \end{cases} \tag{2.31}$$

After that, check for the ending conditions and, if needed, repeat. The weight given to each part in updating velocity can be changed during the algorithm's runtime. At the beginning of the run, the particle's velocity will get a more significant weight than the weights of the global and personal best to explore as much of the solution space as possible. Still, as the algorithm progresses, the ratio will change so that all particles converge to the global minimum.

2.7.1 Coefficients Adaptation

As mentioned before, during the run of the PSO algorithm, we want to adapt the inertia coefficient ω and acceleration coefficients c_1 and c_2. Two techniques will be used to achieve this. The first is for the adaptation of the acceleration coefficient, also known as PSO-TVAC, while the second is for the adaptation of the inertia coefficient, also known as PSO-AWDV (Adaptive Weighted Delay Velocity) [19]. As

mentioned before, at the beginning of the run, we wish to explore as much of the solution space as possible, so we put more weight on each particle's personal best, and as the algorithm progresses, the weight transfers to the global best to help the algorithm converge to its global best. To do so, we use the following

$$\begin{aligned} c_1 &= (c_{1i} - c_{1f})x\frac{k_{max}-k}{k_{max}} + c_{1f} \\ c_2 &= (c_{2i} - c_{2f})x\frac{k_{max}-k}{k_{max}} + c_{2f} \end{aligned} \tag{2.32}$$

where c_{1i} and c_{1f} indicate, respectively, the initial and final values of the acceleration factor c_1. Likewise, the initial and final values of the acceleration factor c_2 are represented by c_{2i} and c_{2f}, respectively. The coefficients k and k_{max} denote, respectively, the current and maximal iterations in the algorithm. To get the effect we want, we define $c_{2i} = c_{1f}$ and $c_{1i} = c_{2f}$.

In contrast, the inertia weight of velocity is adaptively regulated following the evolutionary state in the optimization, and it is calculated as follows

$$w = 1 - \frac{a}{1 + e^{b \cdot E(k)}} \tag{2.33}$$

where a and b are two parameters that can be designed to adjust the search performance of PSO, $E(k)$ is the estimation value (EV) of the evolutionary state at the kth iteration, and can be computed as follows,

$$E(k) = \frac{|f_{max}(k)| - |f_{min}(k)|}{|f_{max}(k)|} \tag{2.34}$$

with $f(k)$ representing fitness. In the PSO context, the EV refers to a measure of the particle swarm's condition or status at any point during the optimization

process. This state adapts the particles' velocity updating function, which can help them navigate the search space more effectively. Specifically, the evolutionary state of the particle swarm is quantitatively evaluated using an estimation method based on the evolutionary factor, such as the difference between the particles' maximal and minimal fitness values. This factor helps determine the particles' dispersed or concentrated in the search space. For instance, a high difference indicates that the particles are widely scattered, representing an early stage of the optimization process. Conversely, a low difference suggests that the particles converge towards potential solutions. This evolutionary state is crucial for determining the inertia weight in adaptive PSO algorithms, influencing how much past velocities affect the current velocity. By adopting the inertia weight based on the evolutionary state, the algorithm can avoid premature convergence to local optima and improve search space exploration. This adaptation allows the algorithm to balance exploration and exploitation dynamically based on how the swarm evolves over iterations. Choosing the parameters a and b in the context of adaptive PSO is critical for regulating the inertia weight dynamically, directly affecting the optimization process. The parameters are chosen to adjust the search performance of the PSO algorithm.

Parameter a primarily affects the upper limit of the inertia weight. A higher a value generally allows the inertia weight to reach a lower value, promoting more exploration over exploitation, especially in the later stages of the search. This can help avoid local minima. A lower a can be considered when the problem requires more rapid convergence, usually in simpler landscapes.

Parameter b influences the rate of change of the inertia weight. A higher b results in a more sensitive change in inertia weight concerning the evolutionary state $E(k)$, allowing quick adaptation to changes in the search landscape. A smaller b makes

the inertia weight change more gradually, which could help balance exploration and exploitation throughout the optimization process. Using this technique, the update of the particle velocity is as follows

$$\vec{v_i}(t+1) = w\vec{v_i}(t) + (1-w)\vec{v_i}(t-1) + c_1 r_1\big(\vec{p_i}(t) - \vec{x_i}(t)\big) + c_2 r_2\big(\vec{g}(t) - \vec{x_i}(t)\big) \quad (2.35)$$

2.8 Convex Set

A convex set is a fundamental concept in optimization and control theory, particularly when dealing with optimization problems. A set $\mathbf{C}$ in a vector space is called convex if, for any two points within the set, the line segment connecting those two points also lies entirely within the set. Formally, a set $\mathbf{C} \subseteq \mathbb{R}$ is convex if for any $x_1, x_2 \in \mathbf{C}$ and for any scalar $\lambda \in [0, 1]$, the point given by the convex combination of x_1 and x_2:

$$\lambda x_1 + (1-\lambda)x_2 \quad (2.36)$$

Also belongs to $\mathbf{C}$. This property can be visualized in two-dimensional space: if you take any two points in a convex set and draw a straight line between them, that line will never leave the set. In contrast, non-convex sets might contain points where the line between them exits the set at some point. Convex sets play a crucial role in optimization because optimization problems over convex sets (known as convex optimization problems) are more accessible and have desirable properties. For instance, if a cost function is convex and the feasible region is a convex set, any local minimum will also be a global minimum. This makes convex optimization particularly useful in control systems, where stability and efficiency are paramount.

2.9 Gain Scheduling and Robust Control

Gain scheduling is a widely used control technique for systems whose dynamics vary with operating conditions. In many practical systems, such as vehicles, the relationship between inputs and outputs is not fixed but depends on variables like speed, load, or environmental factors. As a result, traditional linear control strategies may fail to perform optimally across all operating conditions. Gain scheduling addresses this issue by adjusting the controller parameters based on real-time measurements of these conditions. In gain scheduling, the control system is designed as a collection of linear controllers, each tailored for a specific range of operating conditions. These conditions, known as scheduling variables, are typically system states or inputs that significantly affect the dynamics. For instance, a vehicle control system might use velocity and steering angle as scheduling variables. The control law gains are "scheduled" based on these variables, ensuring optimal performance across a wide range of operating scenarios. The typical steps involved in gain scheduling include[20]:

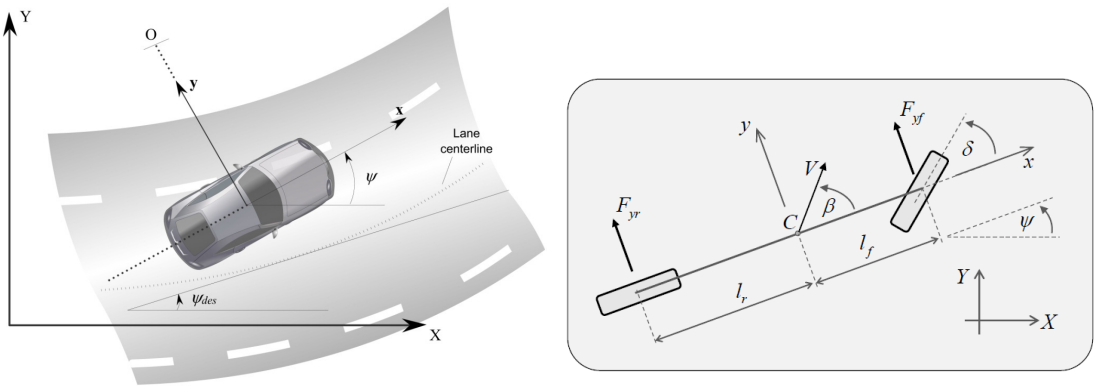
1. Identifying scheduling variables that influence system behavior.
2. Designing local controllers (typically linear) for each operating condition.
3. Interpolating between controllers to smooth the transition from one set of gains to another as operating conditions change.

Robust control, on the other hand, is designed to maintain system stability and performance despite model changes. Unlike the gain scheduling approach, knowledge of the value of the scheduling variables is not needed in real time. This makes the algorithm simpler but generally more conservative. There is a wide variety

of robust control methods. Here, we remain within the same framework used for developing the gain scheduling control and suggest the technique to maintain the desired performance even when the system deviates from a nominal model. In both presented methods (gain scheduling and robust) model variations are bounded by a polytope.

2.10 Lane Keeping System

LKS is a technology designed to assist drivers in maintaining the car's lane by correcting the steering. The system is developed based on a dynamic vehicle model with 3 DOF. This model is referred to as the "Bicycle Model" (see the figure below). Each pair of wheels is combined into one wheel located in the center of the wheel's axis. The control system aims to minimize both position and heading errors (e_1 and e_2), both measured with respect to the center of the desired lane.



(a) (b)

Figure 2.3: (a) Lateral Vehicle Dynamics, (b) Vehicle Bicycle Model [1]

The position error dynamic is described as follows,

$$\ddot{e_1} = (\ddot{y} + \nu_x \dot{\psi}) - \frac{V_x^2}{r} \tag{2.37}$$

and the dynamic of the orientation error is developed by,

$$e_2 = \psi - \psi_{des} \tag{2.38}$$

with the use of equations (2.37) & (2.38) the state-space vector can be assembled (the full details can be found in [1]),

$$x = \begin{bmatrix} e_1 & \dot{e_1} & e_2 & \dot{e_2} \end{bmatrix}^T, u = \begin{bmatrix} \delta \end{bmatrix} \tag{2.39}$$

The state-space matrices are as follows,

$$\begin{aligned} A &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-2C_{af}-2C_{ar}}{mV_x} & \frac{2C_{af}+2C_{ar}}{m} & \frac{-2C_{af}l_f+2C_{ar}l_r}{mV_x} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-2C_{af}l_f+2C_{ar}l_r}{I_zV_x} & \frac{2C_{af}l_f-2C_{ar}l_r}{I_z} & \frac{-2C_{af}l_f^2-2C_{ar}l_r^2}{I_zV_x} \end{bmatrix}, \\ B_1 &= \begin{bmatrix} 0 \\ \frac{2C_{af}}{m} \\ 0 \\ \frac{2C_{af}l_f}{I_z} \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 \\ \frac{-2C_{af}l_f+2C_{ar}l_r}{mV_x} - V_x \\ 0 \\ \frac{-2C_{af}l_f^2-2C_{ar}l_r^2}{I_zV_x} \end{bmatrix} \end{aligned} \tag{2.40}$$

While the system's state-space equation is,

$$\dot{x} = Ax + B_1\delta + B_2\dot{\psi}_{des}, \quad u = [\delta] \tag{2.41}$$

In the development of the algorithm in this thesis, we assume that the road (i.e., desired lane) is a straight line; therefore, $\dot{\psi}_{des} = 0$.

Chapter 3

Algorithm Development

This chapter outlines the systematic development of the algorithms that form the foundation of this research. The primary goal is to create a data-driven, sub-optimal controller capable of operating within situations of model variations. To achieve this, we first analyze the feasibility of calculating optimal gains by GA and PSO in well-defined systems (through known model matrices). Subsequently, we develop methods to derive (by GA and PSO) optimal gains in model-free systems using experimental data. Finally, we extend this methodology to compute sub-optimal gains for systems characterized by varying parameters (such as velocity and mass in the lane-keeping application). The algorithm's effectiveness is initially validated using the system introduced in Section 2.10, followed by an evaluation of two additional systems to ensure general applicability.

3.1 Control of Linear Parameter-Varying Systems

LPV systems constitute a fundamental class of dynamical systems whose state-space representation exhibits linear structure with coefficients that depend on time-varying parameters. Assume an LPV of the following structure,

$$\dot{x}(t) = A(\theta)x(t) + Bu(t) \tag{3.1}$$

where $x(t) \in \mathbb{R}^n$ represents the state vector, $u(t) \in \mathbb{R}^m$ denotes the control input, and the vector $\theta = [\theta_1 \ \cdots \ \theta_k]^T \in \mathbb{R}^k$ represents a set of parameters with a value that is unknown (a priory) for the designer. For all $\theta_i, \ i = [1...k]$ the min and max bounds are given, i.e., $\theta_{min} < \theta_i < \theta_{max}$. We assume that $A(\theta)$ depends linearly on θ, in a way that can be represented as a convex combination of matrices representing all combinations of the extreme values of all θ_i, i.e., $A(\theta) = \sum_1^{2^k} \alpha_i A_i, \ \sum_1^{2^k} \alpha_i = 1$. Hence, $A(\theta)$ represents a polytope with 2^k vertices.

LPV systems serve as a powerful modeling paradigm that bridges the gap between linear time-invariant (LTI) systems and nonlinear systems, offering a structured approach to control synthesis with formal stability guarantees. This framework has proven particularly valuable for representing complex nonlinear dynamics in applications ranging from aerospace systems and vehicle dynamics to chemical processes, where system behavior undergoes significant variations across operating conditions while maintaining an underlying linear structure parameterized by exogenous variables.

Two types of controllers can be designed for the LPV system,

1. Robust controller – a unified (single) controller that guarantees performance over the entire parameter (θ_i) space.

2. Gain scheduling (GS) controller – a composite controller based on interpolating controllers attributed to extreme values of model parameters (θ_i).

The advantage of the robust controller is that the actual values of θ_i are not required. Nevertheless, if the actual values θ_i are given 'on the fly', the robust controller can be considered conservative (in the sense that the provided information can obtain better performance).

The alternative to the robust controller is the gain-scheduling controller which requires online knowledge of θ_i to generate an updated value for the controller based on interpolation. The gain-scheduling controller is a composition of individual controllers, each suboptimal concerning a vertex of the $A(\theta)$ polytope. To guarantee the stability of the composite controllers, all individual controllers are designed to be quadratically stable with respect to a shared Lyapunov function, $V_i = x^T P x$.

3.1.1 Gain Scheduling by State Feedback Controllers

As mentioned, gain scheduling is a method to control an LPV system. The control system is designed as a collection of linear controllers, each tailored for a specific range of operating conditions. These conditions, known as scheduling variables, are typically system states or inputs significantly affecting the dynamics. Let,

$$\dot{x}(t) = A(\theta)x(t) + Bu(t), \quad A(\theta) = \sum_{1}^{p} \alpha_i A_i, \quad \sum_{1}^{p} \alpha_i = 1, \quad p = 2^k \tag{3.2}$$

So the individual model corresponds to a vertex is $\dot{x} = A_i x + Bu$.

Let $u = -K_i x$ be a state feedback controller that can be shown to stabilize the individual model by the same quadratic Lyapunov function $V_i = x^T P x$. From the time derivative of the Lyapunov function and the combination of (3.2) with the state feedback controller

$$\begin{aligned} \dot{V}_i &= \dot{x}^T P x + x^T P \dot{x} \\ &= x^T (A_i - BK_i)^T P x + x^T P (A_i - BK_i) x \\ &= x^T [(A_i - BK_i)^T P + P(A_i - BK_i)] x \end{aligned} \tag{3.3}$$

From the Lyapunov stability theory, $\dot{V}_i = -x^T \overline{Q}_i x < 0$, $\overline{Q}_i > 0$.

After designing a stabilizing controller $u = -K_i x$ for any $\dot{x} = A_i x + Bu$. The GS controller can be composed based on the knowledge of α_i. Let the GS controller be

$$u(t) = Kx(t), \quad K = \sum_{1}^{p} \alpha_i K_i, \quad p = 2^k \tag{3.4}$$

Then the closed-loop system will be

$$\dot{x}(t) = \left(\sum_{1}^{p} \alpha_i A_i \right) x + Bu = \left(\sum_{1}^{p} \alpha_i A_i \right) x - B \left(\sum_{1}^{p} \alpha_i K_i \right) x = \sum_{1}^{p} \alpha_i (A_i - BK_i) x \tag{3.5}$$

Theorem 3.1 *The GS state feedback controller is quadratically stable.*

Proof. Consider the quadratic Lyapunov function, $V = x^T P x$, and (3.3), the time

derivative is,

$$
\begin{aligned}
\dot{V} &= \dot{x}^T P x + x^T P \dot{x} \\
&= x^T \textstyle\sum_1^p \alpha_i (A_i - BK_i)^T P x + x^T P \sum_1^p \alpha_i A_i - BK_i) x \\
&= \textstyle\sum_1^p \alpha_i x^T [(A_i - BK_i)^T P + P(A_i - BK_i)] x \\
&= - \textstyle\sum_1^p \alpha_i x^T \overline{Q}_i x < 0
\end{aligned}
\tag{3.6}
$$

From (3.3) we have $\overline{Q}_i > 0$, then the feedback controller is quadratically stable. □

3.1.2 Robust controller for an LPV system

In the context presented here, robust controller synthesis for LPV systems addresses the challenge of maintaining stability and performance in the presence of parametric variations. Consider a system of the form

$$
\dot{x}(t) = A(\theta)x(t) + Bu(t), \quad A(\theta) = \sum_1^p \alpha_i A_i, \quad \sum_1^p \alpha_i = 1, \quad p = 2^k \tag{3.7}
$$

The individual model corresponds to a vertex is $\dot{x} = A_i x + Bu$. Let $u = -Kx$ be a state feedback controller that can be shown to stabilize the individual models by the same quadratic Lyapunov function $V_i = x^T P x$. From the time derivative of the Lyapunov function and the state feedback controller (with the unified gain K for all individual models)

$$
\begin{aligned}
\dot{V}_i &= \dot{x}^T P x + x^T P \dot{x} \\
&= x^T (A_i - BK)^T P x + x^T P (A_i - BK) x \\
&= x^T [(A_i - BK)^T P + P(A_i - BK)] x
\end{aligned}
\tag{3.8}
$$

From the Lyapunov stability theory, $\dot{V}_i = -x^T \overline{Q}_i x < 0$, $\overline{Q}_i > 0$.

Since $B = \sum_1^p \alpha_i B$, $\sum_1^p \alpha_i = 1$ the closed-loop system can be represent as,

$$\dot{x}(t) = \left(\sum_1^p \alpha_i A_i\right) x + Bu = \left(\sum_1^p \alpha_i A_i\right) x - \left(\sum_1^p \alpha_i B\right) Kx = \sum_1^p \alpha_i (A_i - BK)x \tag{3.9}$$

Theorem 3.2 *The robust state feedback controller is quadratically stable.*

Proof. The procedure here is similar to that of the GS controller proof but for an identical K in all individual sub-models. Consider the quadratic Lyapunov function, $V = x^T P x$, the time derivative is,

$$\begin{aligned}
\dot{V} &= \dot{x}^T P x + x^T P \dot{x} \\
&= x^T \textstyle\sum_1^p \alpha_i (A_i - BK)^T P x + x^T P \sum_1^p \alpha_i (A_i - BK) x \\
&= \textstyle\sum_1^p \alpha_i x^T [(A_i - BK)^T P + P(A_i - BK)] x \\
&= -\textstyle\sum_1^p \alpha_i x^T \overline{Q}_i x < 0 \ , \quad \overline{Q}_i > 0
\end{aligned} \tag{3.10}$$

Which guarantees the stability of the LPV system. □

3.2 Sub-Optimal Model Free Control

Sub-optimal control refers to applying control strategies that, while not necessarily achieving the absolute optimal solution, provide satisfactory and practical performance under given constraints. For systems where the dynamics are not fully understood or where computational resources are limited, sub-optimal control becomes a pragmatic approach to achieving control objectives. These scenarios are particularly prevalent in complex systems like autonomous vehicles, where real-time

decision-making is crucial, and system dynamics can be unpredictable. Recall the LPV model,

$$\dot{x}(t) = A(\theta)x(t) + Bu(t), \quad A(\theta) = \sum_{1}^{p} \alpha_i A_i, \quad \sum_{1}^{p} \alpha_i = 1, \quad p = 2^k \tag{3.11}$$

So the individual model corresponds to a vertex is $\dot{x} = A_i x + Bu$. We are interested in suboptimal controllers that share an identical Lyapunov function for all A_i, to show stability. For LQR controllers, it is common to use the $P = P^T$, which is the solution of the Riccati equation, to form a quadratic Lyapunov function, which may also be referred to as the value function. The optimal controller is the one with the minimal P matrix. It is generally impossible to have an identical optimal P matrix for different individual system models. Therefore, we slightly modify the expression for the optimal gain, to be

$$K = R^{-1}B^T P + L \tag{3.12}$$

This gain is optimal for $L = 0$, and sub-optimal for $L \neq 0$. Obviously, we wish a minimal L to be as near as possible to optimality. The degree of freedom that we now have in K will allow the GS and robust controllers described earlier, e.g., we may design $K_i = R^{-1}B^T P + L_i$ for different individual models with identical P. multiple controllers with identical P. To solve for the sub-optimal P, the sub-optimal gain (3.12) is substituted in (2.10), to yield the following sub-optimal Lyapunov equation,

$$\begin{aligned} A^T P + PA - PBR^{-1}B^T P + (R^{-1}B^T P - K)^T R(R^{-1}B^T P - K) + Q = 0 \\ A^T P + PA - PBR^{-1}B^T P + Q + L^T RL = 0 \\ A^T P + PA - PBR^{-1}B^T P + Q = -L^T RL \end{aligned} \tag{3.13}$$

This result and the comparison theorem reveal the influence of L on sub-optimality. It is clear that sub-optimal controllers should be designed for minimal $L^T RL$. This can be achieved by minimizing the Frobenius Norm of $A^T P + PA - PBR^{-1}B^T P + Q$, which for the sub-optimal controller is different from zero.

3.2.1 Stability of Sub-Optimal Controllers

The optimal LQR controller is guaranteed to be stable for a stabilizable and detectable set $S(A, B, \sqrt{Q})$. To obtain stability of suboptimal controllers (3.12), recall (2.8) and (2.10), which yielded (3.13). Hence, the existence of (3.13) is a sufficient condition for the existence of,

$$\begin{gathered} (A - BK)^T P + P(A - BK) = -\overline{Q} \leq 0 \\ \overline{Q} = -Q - K^T RK \end{gathered} \tag{3.14}$$

for the gain $K = R^{-1}B^T P + L$. Now by taking a Lyapunov function of the form $V = x^T Px$, its time derivative is $\dot{V} = -x^T \overline{Q}x \leq 0$ with $\overline{Q} = -Q - K^T RK$. This result, with the condition for detectability of $(A, \sqrt{Q})$, guarantees the asymptotic stability of the closed-loop (note that this always exists for $Q > 0$). To summarize, the following two conditions guarantee the stability of the suboptimal controller,

$$\begin{gathered} A^T P + PA - PBR^{-1}B^T P + Q \leq 0 \\ P > 0 \end{gathered} \tag{3.15}$$

The first condition guarantees the existence of (3.13) for some L, while the second is required for P to form a Lyapunov function.

3.2.2 Gain Scheduling and Robust Controllers Design Equations

For p model vertices, the gain scheduling control problem requires solving,

$$\begin{aligned} A_1^T P + PA_1 - PBR^{-1}B^T P + Q &= -L_1^T R L_1 \\ A_2^T P + PA_2 - PBR^{-1}B^T P + Q &= -L_2^T R L_2 \\ \vdots \qquad\qquad \vdots \qquad\qquad &\quad \vdots \\ A_p^T P + PA_p - PBR^{-1}B^T P + Q &= -L_p^T R L_p \end{aligned} \tag{3.16}$$

for the following unknown matrices, $P, L_1, L_2, \cdots, L_p$. Note that all equations are solved with identical P but different L. Then, P gains are obtained for the P vertices by,

$$\begin{aligned} K_1 &= R^{-1}B^T P + L_1 \\ K_2 &= R^{-1}B^T P + L_2 \\ &\vdots \\ K_p &= R^{-1}B^T P + L_p \end{aligned} \tag{3.17}$$

The robust design, on the other hand, requires a unified controller for all p vertices. This is obtained by solving all Riccati equations for an identical L, which gives,

$$\begin{aligned} A_1^T P + PA_1 - PBR^{-1}B^T P + Q &= -L^T R L \\ A_2^T P + PA_2 - PBR^{-1}B^T P + Q &= -L^T R L \\ \vdots \qquad\qquad \vdots \qquad\qquad &\quad \vdots \\ A_p^T P + PA_p - PBR^{-1}B^T P + Q &= -L^T R L \end{aligned} \tag{3.18}$$

to be solved for the unknown matrices P and L. The sub-optimal robust controller is,

$$K = R^{-1}B^T P + L \tag{3.19}$$

3.2.3 Mixed GS Robust control problem

If LPV parameters are not given in real-time, robust control is the obvious choice. But robust controllers tend to be more conservative (i.e., "less optimal") as they are forced to be solved for an identical L in (3.18). Assume a case where some LPV model parameters are given in real-time and some are not. For this case, a mixed GS-robust controller can be designed. This design will provide robustness with respect to the unknown parameters, and be scheduled with respect to the parameters given in real time.

As an example of this concept, consider a four-vertex LPV system of the form,

$$\dot{x}(t) = A(\theta)x + Bu, \qquad \theta = [\theta_1, \theta_2]^T \tag{3.20}$$

Where

$$\begin{aligned} A_1 &= A(\theta_{1,min}, \theta_{2,min}) \quad A_3 = A(\theta_{1,max}, \theta_{2,min}) \\ A_2 &= A(\theta_{1,min}, \theta_{2,max}) \quad A_4 = A(\theta_{1,max}, \theta_{2,max}) \end{aligned} \tag{3.21}$$

Assume, for example, that θ_2 is not available for GS (i.e., unknown in real time).

Then, for this case, solve,

$$
\begin{aligned}
(\theta_{1,min}, \theta_2) \Leftrightarrow & \begin{cases} A_1^T P + PA_1 - PBR^{-1}B^T P + Q = -L_1^T R L_1 \\ A_2^T P + PA_2 - PBR^{-1}B^T P + Q = -L_1^T R L_1 \end{cases} \\
(\theta_{1,max}, \theta_2) \Leftrightarrow & \begin{cases} A_3^T P + PA_3 - PBR^{-1}B^T P + Q = -L_2^T R L_2 \\ A_4^T P + PA_4 - PBR^{-1}B^T P + Q = -L_2^T R L_2 \end{cases}
\end{aligned}
\tag{3.22}
$$

for the unknown matrices, P, L_1, L_2. The two obtained controllers are,

$$
\begin{aligned}
K_1 &= R^{-1}B^T P + L_1 \\
K_2 &= R^{-1}B^T P + L_2
\end{aligned}
\tag{3.23}
$$

which are to be scheduled with respect to θ_1.

3.2.4 Suboptimal Control in LPV Model Free Systems

Given the LPV model matrices A_i and B, we have shown that the sub-optimal controller can be developed from a set of Riccati equations of the form (3.13). However, this becomes a challenge when the system matrices are unknown. Here, we suggest a data-driven representation of (3.13). This representation is not based on the model matrices A and B, which are replaced by data, including the state variables x and the corresponding input variables u. Consider a system with the state space model,

$$
\dot{x} = Ax + Bu \tag{3.24}
$$

Let $V = x^T P x$, then the time derivative of V with respect to the system's trajectories is,

$$\dot{V} = x^T(A^T P + PA)x + 2x^T PBu \tag{3.25}$$

Form (3.13), we can write,

$$PA + A^T P = PBR^{-1}B^T P - Q - L^T RL \tag{3.26}$$

and with $\overline{Q} = Q + L^T RL$ and (3.25), we have,

$$\begin{aligned} \dot{V} &= x^T(PBR^{-1}B^T P - \overline{Q})x + 2x^T PBu \\ &= x^T(P_B^T R^{-1} P_B - \overline{Q})x + 2x^T P_B^T u \\ &= x^T(P_B^T R^{-1} P_B - \overline{Q})x + 2u^T P_B x \end{aligned} \tag{3.27}$$

where we have defined $P_B = B^T P$.

Integrating on (3.27) from t_i to $t_i + T$,

$$\begin{aligned} &V(t_i + T) - V(t_i) \\ &= x^T(t_i + T)Px(t_i + T) - x(t_i)Px(t_i) \\ &= \int_{t_i}^{t_i+T} x^T(P_B^T R^{-1} P_B - \overline{Q})xdt + 2\int_{t_i}^{t_i+T} u^T P_B x dt \end{aligned} \tag{3.28}$$

Rearrangement for a numerical solution using Kronecker product shown in subsection 2.2

$$\begin{aligned} &x^T(t_i + T)Px(t_i + T) - x(t_i)Px(t_i) \\ &= \left[x^T(t_i + T) \otimes x^T(t_i + T) - x^T(t_i) \otimes x^T(t_i)\right] N\mathsf{svec}(P) \end{aligned} \tag{3.29}$$

$$\int_{t_i}^{t_i+T} x^T(P_B^T R^{-1} P_B)xdt = \left[\int_{t_i}^{t_i+T} x^T \otimes x^T d\tau\right] \mathsf{vec}(P_B^T R^{-1} P_B) \tag{3.30}$$

$$\int_{t_i}^{t_i+T} x^T(\overline{Q})xdt = \left[\int_{t_i}^{t_i+T} x^T \otimes x^T d\tau\right] \mathsf{vec}(\overline{Q}) \tag{3.31}$$

$$\int_{t_i}^{t_i+T} u^T(P_B)xdt = 2\left[\int_{t_i}^{t_i+T} x^T \otimes u^T d\tau\right] \mathsf{vec}(P_B) \tag{3.32}$$

Further, we define matrices $\delta_{xx} \in \mathbb{R}^{l\mathbf{x}n^2}$, $I_{xx} \in \mathbb{R}^{l\mathbf{x}n^2}$ and $I_{xu} \in \mathbb{R}^{l\mathbf{x}mn}$, such that

$$\delta_{xx} = \left[x \otimes x|_{t_1}^{t_1+T},\ x \otimes x|_{t_2}^{t_2+T},\ \cdots,\ x \otimes x|_{t_l}^{t_l+T}\right]^T \tag{3.33}$$

$$I_{xx} = \left[\int_{t_1}^{t_1+T} x \otimes xd\tau,\ \int_{t_2}^{t_2+T} x \otimes xd\tau,\ \cdots,\ \int_{t_l}^{t_l+T} x \otimes xd\tau\right]^T \tag{3.34}$$

$$I_{xu} = \left[\int_{t_1}^{t_1+T} x \otimes ud\tau,\ \int_{t_2}^{t_2+T} x \otimes ud\tau,\ \cdots,\ \int_{t_l}^{t_l+T} x \otimes ud\tau\right]^T \tag{3.35}$$

where $0 \leq t_1 < t_2 < \cdots < t_l$ (i.e., the integration is performed in multiple time intervals). This enables the formulation of multiple independent equations of the form (3.28).

Using (3.29, 3.30, 3.31, 3.32, 3.33, 3.34, 3.35) in (3.28), we get,

$$\delta_{xx}N\mathsf{svec}(P) - 2I_{xu}\mathsf{vec}(P_B) - I_{xx}\mathsf{vec}(P_B^T R^{-1} P_B) + I_{xx}\mathsf{vec}(\overline{Q}) = 0 \tag{3.36}$$

and with, $\overline{Q} = Q + L^T RL$, the following data-driven alternative of the Riccati equation is achieved,

$$\begin{aligned} \delta_{xx}N\mathsf{svec}(P) - 2I_{xu}\mathsf{vec}(P_B) - I_{xx}\mathsf{vec}(P_B^T R^{-1} P_B) + I_{xx}\mathsf{vec}(Q) \\ = -I_{xx}\mathsf{vec}(L^T RL) \end{aligned} \tag{3.37}$$

Using data, we have $\delta_{xx}, I_{xx}, I_{xu}$ (with their definition above), and from (3.37) one can solves for (P, P_B, L) in a model-free context that replaces (3.13). The number of

unknown scalars in (P, P_B, L) determines the minimal number of time sections (t_i to $t_i + T$) required for the solution. To obtain independent equations, the system needs to be persistently excited (meaning, the data is sufficiently reached to represent the dynamics). Persistent excitation is a standard condition necessary for system identification. Generally, to obtain persistent excitation the system input u includes a component of random noise.

Lastly, we reformulate the stability criteria in (3.15) to a model-free variant, given as,

$$\begin{gathered} \delta_{xx} N \mathsf{svec}(P) - 2I_{xu}\mathsf{vec}(P_B) - I_{xx}\mathsf{vec}(P_B^T R^{-1} P_B) + I_{xx}\mathsf{vec}(Q) \leq 0 \\ P > 0 \end{gathered} \tag{3.38}$$

3.3 Model-Based Solution of Riccati Equation by PSO and GA

In this section, we explore the solution of the Riccati equation using Particle Swarm Optimization (PSO) and Genetic Algorithms (GA). Traditional methods for solving the Riccati equation rely on analytical or numerical approaches that do not necessarily suit the sub-optimal design framework developed here. Sub-optimal controllers can be solved by LMIs, for known model matrices. By leveraging PSO and GA, to solve Riccati equations, we suggest an alternative approach. The main purpose of this section is to determine whether these algorithms can effectively solve a well-defined (i.e., model-based) Riccati equation, allowing us to apply them later to solve the model-free case. For development and examination, the algorithms are demon-

strated using the lane-keeping control design problem, discussed in the sub-section 2.10.

Consider a model-based Riccati equation of the form,

$$PA + A^T P - PBR^{-1}B^T P + Q = 0 \tag{3.39}$$

P is the unknown n by n symmetric matrix and A, B, Q, R are known real coefficient matrices, with Q and R symmetric. Both GA and PSO require a fitness function to guide the optimization process by minimizing an objective value. In this case, the fitness function is based on the Frobenius norm of the Riccati equation residual. The goal is to iteratively adjust the decision variables to minimize the norm of the residual. This approach allows us to first validate the effectiveness of these algorithms on a well-defined Riccati equation before extending their application to the model-free case. We first define the fitness as,

$$Fitness = PA + A^T P - PBR^{-1}B^T P + Q \tag{3.40}$$

and the fitness value as its Frobenius norm (described in section 2.3),

$$Fitness\_Value = \| \ Fitness \ \|_F \tag{3.41}$$

For the example, the matrices A and B are calculated from (2.40), using the model parameters in (3.46). The matrices Q and R are predetermined and chosen by the user. Q is a diagonal matrix and is described as follows

$$Q = diag[q_1 \quad q_2 \quad q_3 \quad q_4] \tag{3.42}$$

where each q_i is chosen based on the relative importance of the state to overall performance. Those considerations chose the Q values:

- q_1 (Lateral position error): This should be relatively high since keeping the vehicle near the center of the lane is a primary objective.
- q_2 (Rate of lateral error): Medium weight to penalize rapid lateral movement, which can feel uncomfortable to passengers.
- q_3 (Yaw angle error): High weight since maintaining proper alignment with the road is critical for stability and safety.
- q_4 (Rate of yaw error): Medium weight to prevent oscillatory steering behavior.

The R value is related to the steering inputs. It controls how aggressively the controller will use steering inputs. Too small will cause aggressive steering that might feel jerky to passengers. Too large will cause sluggish lane-keeping performance. Based on those considerations, we chose Q and R to be

$$\begin{gathered} Q = diag[5 \quad 1 \quad 100 \quad 1] \\ R = 2 \end{gathered} \tag{3.43}$$

3.3.1 PSO in well-defined system

In this subsection, we apply Particle Swarm Optimization (PSO), presented in subsection 2.7, to solve the Riccati equation in a well-defined system with known dynamics, (3.40). PSO, a population-based optimization technique inspired by the collective behavior of swarms, iteratively adjusts candidate solutions to minimize

the value function, (3.41). By evaluating multiple potential solutions and updating them based on individual and collective experiences, PSO efficiently navigates the search space to find an optimal or near-optimal solution. This initial evaluation is a benchmark to determine whether PSO can accurately solve the Riccati equation before extending its use to model-free scenarios. By combining (3.40) and (3.41), along with the PSO explanation presented in subsection 2.7, the following algorithm has been developed.

Algorithm 3.1: Standard PSO algorithm for a well define system

```
Input: PSO Parameters, System Parameters
Output: Optimal Gain from the solution matrix P
1: Initialization of the PSO
2: Determine the system matrix
3: Determine the fitness value function (3.41) as a function of P
4: for iteration = 1 to Number_Of_Iterations do
    for i = 1 to Number_Of_Particles do
        fitness = Fitness_Value(particles[i])
        if fitness < Personal_Best_Value[i] then
            Personal_Best[i] = particles[i]
            Personal_Best_Value[i] = fitness
        end
        if fitness < Global_Best_Value then
            Global_Best = particles[i]
            Global_Best_Value = fitness
        end
    end
    for i = 1 to number of particles do
        Update the velocity and position of the particle.
    end
end
5: Calculate the Optimal Gain from the solution of the algorithm P
```

After multiple executions of the algorithm, we observed that PSO does not consistently converge to a single solution, resulting in varying outcomes across different

runs. Additionally, the obtained solutions do not always ensure system stability. See Table 3.1.

Run No.	G_Best	k_1	k_2	k_3	k_4	Stability
1	4.1154	1.7854	0.8165	3.3961	1.0147	Yes
2	4.4653	1.8084	0.8206	4.3026	1.0784	Yes
3	1.2882	-1.5740	-0.0170	-7.4035	-0.4565	No

Table 3.1: Algorithm solution for PSO well-defined

When using a metaheuristic like PSO to minimize (or solve) a Riccati equation, the risk is that we rely on a stochastic optimization process that may or may not converge to a globally optimal (and stabilizing) solution. There are a few common factors that can contribute to these inconsistent or unstable results:

1. **Random Initialization:** PSO starts with a swarm of candidate solutions that are randomly initialized. Different random seeds can lead to different initial positions and velocities, which, in turn, can lead to different final solutions (some of which may not stabilize the system).

2. **Non-Convex Objective Landscape:** The cost function for the Riccati problem may be non-convex, containing multiple local minima. Depending on the parameterization and constraints, PSO might converge to a local minimum that does not guarantee system stability.

3. **Inadequate Parameter Settings:** PSO has hyperparameters such as inertia weight, cognitive and social acceleration coefficients, swarm size, and maximum iterations. Poorly chosen parameters can cause premature convergence or failure to find stabilizing solutions.

4. **Lack of Stability Criterion:** If the optimization minimizes some cost without explicitly enforcing stability constraints, the solution might be "optimal" concerning that cost but still destabilizing in practice.

We can take several steps to address these issues to enhance the algorithm's performance. First, we can improve PSO by incorporating Adaptive Weight Dynamic Variation (AWDV), as discussed in 2.7.1, and fine-tuning the a and b parameters to prevent convergence to local minima. Second, when evaluating stability in a well-defined system, we can assess the stability of each particle by computing the associated gain. Next, we determine the eigenvalues of the closed-loop system—if all eigenvalues have negative real parts, the system is stable. A penalty is applied to the fitness value of particles that do not meet this criterion to enforce stability during optimization, ensuring that the updated fitness function prioritizes stable solutions.

$$Fitness\_Value = \| Fitness \|_F + Penalty \tag{3.44}$$

Applying those changes, the modified and improved algorithm is

Algorithm 3.2: Stabilize PSO algorithm for a well define system

```
Input: PSO Parameters, System Parameters
Output: Optimal Gain from the solution matrix P
1: Initialization of the PSO
2: Determine the system matrix
3: Determine the fitness value function (3.41) as a function of P
4: for iteration = 1 to Number_Of_Iterations do
    for i = 1 to Number_Of_Particles do
        fitness = Fitness_Value(particles[i])
        if Stability_Check = false then
            fitness = fitness + Penalty
        end
        if fitness < Personal_Best_Value[i] then
            Personal_Best[i] = particles[i]
            Personal_Best_Value[i] = fitness
        end
        if fitness < Global_Best_Value then
            Global_Best = particles[i]
            Global_Best_Value = fitness
        end
        Update the velocity and position of the particle.
    end
    Adjusting the algorithm weights
end
5: Calculate the Optimal Gain from the solution of the algorithm P
```

Using the stability check provides us with a stabilizing solution, and the problem of inconsistencies was solved by adjusting the weights.

Run No.	Global best P	k_1	k_2	k_3	k_4	Stability
1	2.7025e-05	1.5811	0.5312	8.1351	0.5359	Yes
2	5.3387e-07	1.5811	0.5312	8.1351	0.5359	Yes
Optimal	9.3124e-13	1.5811	0.5312	8.1351	0.5359	Yes

Table 3.2: Algorithm solution for PSO well-defined

Table 3.2 indicates that the solution generated by the improved algorithm closely aligns with the optimal solution. In conclusion, the algorithm presented in Algorithm 3.2 effectively addresses the shortcomings of the previous Algorithm 3.1 by introducing a stability penalty and applying the AWDV method. These refinements have significantly improved efficiency, accuracy, and stability, offering a more robust solution to earlier challenges concerning consistency and reliability. Furthermore, the results of this section illustrate the strong potential of PSO for tackling model-free systems, demonstrating its viability and effectiveness in data-driven control strategies. These advancements lay a solid foundation for future development and broader application in the field.

3.3.2 GA in well-defined system

In this subsection, we apply Genetic Algorithms (GA), presented in subsection 2.6, to solve the Riccati equation in a well-defined system with known dynamics. GA is an evolutionary optimization technique inspired by natural selection, where candidate solutions evolve over successive generations through selection, crossover, and mutation. By formulating the optimization problem to minimize the value function of the Riccati equation, GA explores a diverse set of solutions while gradually converging toward an optimal one. This evaluation is a benchmark to determine whether GA can reliably solve the Riccati equation before extending its application

to model-free systems. For the development of the algorithm, we used the same fitness function as in the well-defined PSO approach, as defined in (3.40), (3.43), and (3.41). Similar to the PSO algorithm, GA also exhibited issues with the stability of the converged solution. To address this, we evaluated the stability of each individual in the population and applied penalties to those that did not meet the stability criteria. This modification helped steer the optimization process toward solutions that minimize the fitness function and ensure system stability.

Algorithm 3.3: Stabilize GA algorithm for a well define system

```
Input: GA Parameters, System Parameters
Output: Optimal Gain from the solution matrix P
1: Create an initial population and replace, not stabilize, individuals
2: Determine the system matrix
3: Determine the fitness value function (3.41) as a function of P
4: Calculate the initial fitness value of each individual and sort the values
5: for generation = 1 to Max_generations do
    for i = 1 to Population_Size do
        Create a mutant vector
        Create a crossover vector
        fitness_new = Fitness_Value(Crossover_Vector)
        if Stability_Check = false then
            fitness_new = fitness_new + Penalty
        end
        Combine_population = [Population, New_Pop]
        Sort_population = sort(Combine_Population, 'ascend')
        Population = Sort_population(1:pop_size)
        best_P = Population(1)
    end
end
6: Calculate the Optimal Gain from the solution of the algorithm P
```

After multiple executions of the algorithm, we observed that large population size is required for GA to converge to the optimal solution consistently. Insufficient population diversity can lead to premature convergence or suboptimal solutions.

Table 3.3 presents the results supporting this observation.

Run No.	Global best P	k_1	k_2	k_3	k_4	Pop Size	Iter No.
1	0.6373	1.5667	0.5881	6.5538	0.5037	500	500
2	0.0444	1.5815	0.5369	8.0256	0.5346	5000	500
Nominal	0.003604	1.5811	0.5312	8.1351	0.5359	5000	50

Table 3.3: Algorithm solution for GA well-defined

As shown in Table 3.3, the obtained values are close to the optimal solution but do not match it exactly. In our efforts to refine the algorithm, we identified that our current sorting method was limiting population diversity. By consistently retaining only the best values from the offspring, we risked reducing genetic variation, leading to generations dominated by descendants of the same parents after multiple iterations. To mitigate this issue, we drew inspiration from the PSO algorithm and modified the sorting and selection process. Instead of purely sorting and selecting the top-performing individuals, we now compare each offspring to its respective parent and retain the one with the better fitness value. This approach helps maintain diversity across generations and improves convergence toward the optimal solution. The revised algorithm is as follows.

Algorithm 3.4: Stabilize GA algorithm for a well-defined system, PSO sort

```
Input: GA Parameters, System Parameters
Output: Optimal Gain from the solution matrix P
1: Create an initial population and replace, not stabilize, individuals
2: Determine the system matrix
3: Determine the fitness value function (3.41) as a function of P
4: Calculate the initial fitness value of each individual and sort the values
5: for generation = 1 to Max_generations do
    for i = 1 to Population_Size do
        Create a mutant vector
        Create a crossover vector
        fitness_new = Fitness_Value(Crossover_Vector)
        if Stability_Check = false then
            fitness_new = fitness_new + Penalty
        end
        if fitness_new < fitness(i) then
            Population(i,:) = Crossover_Vector
            fitness(i) = fitness_new
        end
        best_fitness = min(fitness)
        best_P = population(best_fitness)
    end
end
6: Calculate the Optimal Gain from the solution of the algorithm P
```

This improved algorithm has yielded more reliable and optimized solutions, requiring fewer converging iterations. The revised approach enhances accuracy and efficiency by preserving population diversity and preventing premature convergence. A comparison of the results before and after these modifications is presented in Table 3.4.

Algorithm	G_best	k_1	k_2	k_3	k_4	Pop Size	Iter No.
3.3	0.6373	1.5667	0.5881	6.5538	0.5037	500	500
3.4	0.000003	1.5811	0.5312	8.1351	0.5359	500	400
Optimal	0.003604	1.5811	0.5312	8.1351	0.5359	5000	50

Table 3.4: GA algorithm solutions comparison

In conclusion, we successfully addressed the limitations identified in Algorithm 3.3 by introducing an alternative sorting approach that streamlines the selection process and enhances overall search efficiency. This refined algorithm demonstrates notable gains in efficiency, stability, and accuracy, making it a more reliable choice for diverse problem settings. Furthermore, the results presented in this section illustrate the strong potential of GA for tackling model-free systems, underscoring its viability in data-driven control applications. These findings lay a solid foundation for further development and broader exploration of GA-based methodologies, especially in domains where explicit system models are challenging to obtain, a topic explored in the remainder of this work.

3.4 Data Collection

This section outlines the approach used to generate the dataset of x and u parameters. As previously discussed, the algorithm's development consists of two phases. The first phase evaluates its potential by computing the optimal solution, while the second phase focuses on deriving the suboptimal solution using the proposed methodology.

3.4.1 Simulation Framework for LPV System Data Generation and Parametric Validation

To generate the required state and input data for numerical validation, we established a comprehensive simulation framework utilizing computer representations of a linear parameter-varying (LPV) system, i.e., the lane-keeping dynamics. A state-space realization was constructed with initial condition vector $x_0 = [0.15, 0.05, -0.1, 0.05]^T$ and simulated over a fixed time horizon of $t_{span} = 100$ time steps. To ensure robust evaluation across varying operating conditions, we systematically perturbed critical parameters including mass ($m \in [1250, 1573]\ kg$) and longitudinal velocity ($V_x \in [10, 40]\ m/s$), with corresponding moment of inertia values computed through the relation

$$I_z = 0.5m(l_f^2 + l_r^2) \tag{3.45}$$

We use ode function integration for each parametric configuration to obtain state trajectories, followed by algebraic computation of the corresponding input vectors using $u = -kx + \epsilon$, where ϵ is a random noise added for persistent excitation. This methodical approach yielded a comprehensive dataset of state-input pairs (x_k, u_k) across diverse operating regions, facilitating subsequent analysis of the algorithm's performance under both nominal and perturbed conditions. The coefficient values

for these systems were obtained from [1].

$$
\begin{aligned}
m &= [1250, 1573]\ [kg] \\
l_r &= 1.58\ [m] \\
l_f &= 1.1\ [m] \\
C_{a1} &= 80000\ [N/rad] \\
C_{a2} &= 80000\ [N/rad] \\
V_x &= [10, 40]\ [m/s]
\end{aligned}
\tag{3.46}
$$

3.5 Data-Driven Model-Free Solution of Riccati Equation

We now extend these methods to a model-free scenario by building on the success of applying Particle Swarm Optimization (PSO) and Genetic Algorithms (GA) to a well-defined Riccati equation. In this case, the system dynamics are unknown, meaning the Riccati equation cannot be directly formulated using predefined matrices. Instead, we rely on data-driven optimization to approximate the optimal solution. By leveraging the adaptability of PSO and GA, we aim to find a suitable control policy without requiring explicit system models. The development of this model-free Riccati equation is presented in subsection 3.2.4. This section explores how these algorithms solve the model-free Riccati equation, assessing their efficiency, stability, and accuracy in an environment where traditional analytical approaches are infeasible. To establish the fitness function, we employed the model-free representation

of the Riccati equation, (3.36) with $L = 0$.

$$Fitness = \delta_{xx} N \mathsf{svec}(P) - 2I_{xu}\mathsf{vec}(P_B) - I_{xx}\mathsf{vec}(P_B^T R^{-1} P_B) + I_{xx}\mathsf{vec}(Q) \quad (3.47)$$

The values of δ_{xx}, I_{xx}, and I_{xu} were computed using the data collected in the simulation section 3.2.4. The weighting matrices R and Q are predefined in (3.43). To identify the optimal solution, we must determine the values of the two unknown parameters, $[svec(P), P_B]$, that minimize the fitness value defined by (3.44). From the previous section, we learned that the algorithm needs a stability check to eliminate unstable solutions; unlike in the model-free method, we can't compute the eigenvalues of the closed-loop system because the system's matrices are unknown. Instead, we use the method presented in 3.2.1 and (3.38).

3.5.1 PSO Model-Free Solution of Riccati Equation

In this subsection, we apply Particle Swarm Optimization (PSO) to solve the Riccati equation by the developed model-free tools, for the case where the system mode is unknown. Unlike the well-defined case presented in section 3.3.1, where the Riccati equation could be directly formulated using known matrices, the model-free approach relies solely on data-driven optimization. PSO is particularly well-suited for this scenario due to its ability to explore high-dimensional search spaces efficiently. The algorithm aims to find the optimal parameters $[svec(P), P_B]$ by minimizing the fitness function, ensuring stability and performance in the absence of a predefined model knowledge.

Before executing the algorithm, we must address the challenges encountered in

the well-defined PSO approach—namely, stability verification and solution inconsistency. Using subsection 3.2.1 to address the stability, we use the stability condition presented in (3.38). Additionally, to mitigate the issue of inconsistent solutions, we fine-tune the algorithm's weight parameters, improving convergence reliability in this model-free setting. The algorithm for PSO with stability check for a model-free system is represented in the Algorithm 3.5.

Algorithm 3.5: Stabilize PSO algorithm for a model-free system

```
Input: PSO Parameters, Dataset of x and u
Output: Optimal Gain from the solution matrix P
1: Initialization of the PSO
2: Calculate δ_xx, I_xx, and I_xu from the dataset
3: Determine the fitness value function (3.41) as a function of [P, P_B]
4: for iteration = 1 to Number_Of_Iterations do
    for i = 1 to Number_Of_Particles do
        fitness = Fitness_Value(particles_P{i},particles_PB{i})
        if Stability_Check = false then
            fitness = fitness + Penalty
        end
        if fitness < Personal_Best_Value then
            Personal_Best_P{i} = particles_P{i}
            Personal_Best_PB{i} = particles_PB{i}
            Personal_Best_Value{i} = fitness
        end
        if fitness < Global_Best_Value then
            Global_Best_P = particles_P{i}
            Global_Best_PB = particles_PB{i}
            Global_Best_Value = fitness
        end
        Update the velocity and position of the particle.
    end
    Adjusting the algorithm weights
end
5: Calculate the Optimal Gain from the solution of the algorithm P
```

The algorithm was tested on LKS design problem presented in Section 2.10. The results of running the algorithm several times are presented in Table 3.5

Run No.	Fitness value	Stability
1	0.0062	Yes
2	0.0065	Yes
3	0.0157	Yes
Optimal	0.000197	Yes

Table 3.5: Algorithm solution for PSO model-free

The algorithm's solutions indicate a return to inconsistencies. To address this, we introduced two new approaches to improve the consistency. The first adjustment involves modifying how the fitness function is evaluated. Consider we have n particles of P and n particles of P_B. Instead of computing the fitness function once using $P(i)$ and $P_B(i)$ directly, we compute the fitness function twice: first, using the current particle of $P(i)$ with the global best of P_B, and second, using the current particle of $P_B(i)$ with the international best of P, while $i = [1....n]$. This ensures that the global best of each parameter is determined independently from its twin, allowing the algorithm to find the most optimal values across all particles rather than being limited to specific pairings. The second adjustment came from some code debugging: the particle values were not converging toward the global best. Instead of decreasing over iterations, particle differences increased, leading to divergence rather than convergence. See Fig. 3.1. This unexpected behavior suggests that the update mechanism requires further refinement to ensure particles move toward an optimal and stable solution.

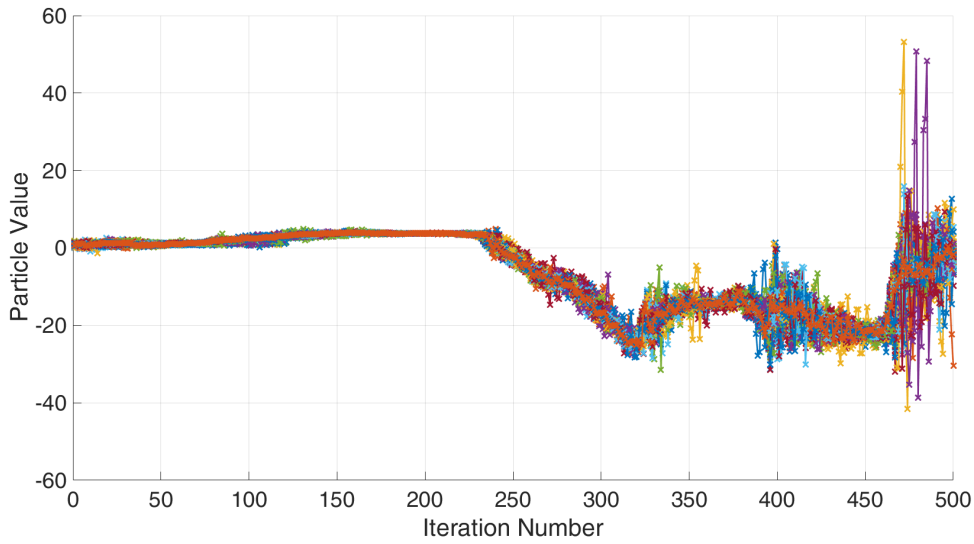


Figure 3.1: Particles values per iteration

If we examine the PSO equation for updating a particle, (2.31). We observe that the particle's value depends on its velocity, and if the velocity becomes too large, it significantly impacts the particle's value. So, we examine the values of the velocities. See Fig. 3.2.

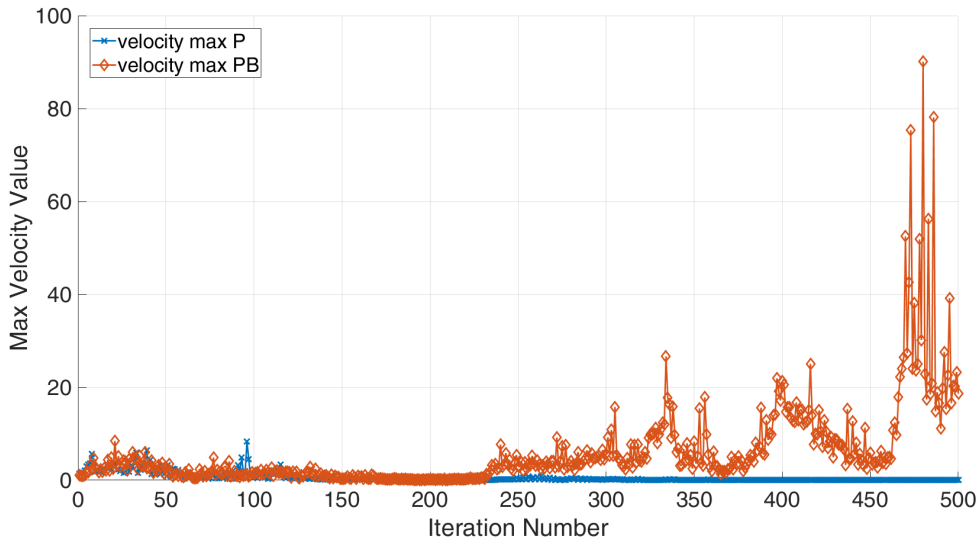


Figure 3.2: Velocities max value as function of the iteration number

The purpose of the velocity in PSO is to guide the particle toward the best solution. However, if the velocity became too large compared to the particle's value, it will not fulfill its purpose and will toss the particle from side to side. To address this issue, we examine the PSO hyperparameters. PSO performance depends strongly on parameters w (inertia), c_1, c_2 (cognitive and social coefficients). We already have some adaptive parameters in our algorithm from the AWDV [19], while the inertia adaptation corresponds to the population's values by calculating the EV value, (2.33). The c_1, c_2 adaptation is very straightforward and doesn't have any connection to the population's values, (2.32). To prevent excessively high velocities, we will adjust the values of c_1 and c_2 based on the diversity of the population. Many papers, such as [21], discuss the idea of adaptation based on population diversity, but we didn't find any that use this to adapt the coefficients. The purpose of this method is to manage the size and direction of velocities based on the diversity of the population. When the population shows high diversity, the global best solution is given greater

weight, helping to pull all particles towards a common target. Conversely, when the population diversity is low, each particle's personal best solution receives more weight, allowing for better solution space exploration. To do so, we will first compute the centroid of all particles and then measure the average distance of each particle to that centroid. To measure the population diversity, we will measure how far particles are spread out from their centroid.

$$diversity = \frac{1}{N}\sum_{i=1}^{N} \|P_i - \bar{P}\| \tag{3.48}$$

while P_i is the i-th particle (in vector form), $\bar{P}$ is the swarm centroid, and N is the number of particles. The adaptation factor, α, is the ratio between the diversity to the maximum diversity.

$$\alpha = \frac{diversity}{max\_diversity} \tag{3.49}$$

The maximum diversity is chosen based on previous runs of the algorithm,

$$\begin{aligned} c_1 &= c_{1min} + (c_{1max} - c_{1min})(1 - \alpha) \\ c_2 &= c_{2min} + (c_{2max} - c_{2min})(\alpha) \end{aligned} \tag{3.50}$$

Based on this approach, if the swarm is very spread out ($\alpha \approx 1$), then $c_1 \approx c_{1min}$ and $c_2 \approx c_{2max}$, the particles will converge to the global best. If the swarm is clustered ($\alpha \approx 0$), then $c_1 \approx c_{1max}$ and $c_2 \approx c_{2min}$, each particle will be more influenced by its personal best solution. We implement this modification into Algorithm 3.5.

Algorithm 3.6: PSO algorithm for a model-free system

```
Input: PSO Parameters, Dataset of x and u
Output: Optimal Gain from the solution matrix P
1: Initialization of the PSO
2: Calculate δ_xx, I_xx, and I_xu from the dataset
3: Determine the fitness value function (3.41) as a function of [P, P_B]
4: for iteration = 1 to Number_Of_Iterations do
    for i = 1 to Number_Of_Particles do
        fitness_P = Fitness_Value(particles_P{i},Global_Best_PB)
        fitness_PB = Fitness_Value(Global_Best_P,particles_PB{i})
        if Stability_Check = false then
            fitness_P = fitness_P + Penalty
        end
        if fitness_P < Personal_Best_Value_P{i} then
            Personal_Best_P{i} = particles_P{i}
            Personal_Best_Value_P{i} = fitness_P
        end
        if fitness_PB < Personal_Best_Value_PB{i} then
            Personal_Best_PB{i} = particles_PB{i}
            Personal_Best_Value_PB{i} = fitness_PB
        end
        if fitness_P < Global_Best_Value_P then
            Global_Best_P = particles_P{i}
            Global_Best_Value_P = fitness_P
        end
        if fitness_PB < Global_Best_Value_PB then
            Global_Best_PB = particles_PB{i}
            Global_Best_Value_PB = fitness_PB
        end
        Update the velocity and position of the particle.
    end
    Adjusting the algorithm weights
end
5: Calculate the Optimal Gain from the solution of the algorithm P
```

Algorithm Type	Fitness value P	Fitness value P_B	Stability
PSO	0.000732	0.000786	Yes
Optimal	0.000197	0.000197	Yes

Table 3.6: Algorithm solution for PSO model-free compared to optimal

The results presented in Table 3.6 indicate the average outcomes from ten runs of the algorithm. The results demonstrate that the PSO algorithm successfully solves the model-free Riccati equation, confirming its potential as an effective optimization tool in data-driven control settings. However, the initial implementation faced challenges with the inconsistent convergence. By introducing the adaptation based on population diversity, we significantly improved the algorithm's performance, ensuring that particles moved toward the global best solution in a more structured and stable manner. This modification dramatically enhanced solution reliability, reduced variance across runs, and improved overall convergence efficiency. These findings highlight the effectiveness of PSO for solving complex, model-free control problems and provide a strong foundation for further exploration in LPV model-free systems.

3.5.2 GA Model-Free Solution of Riccati Equation

In this subsection, we extend the application of Genetic Algorithms (GA) to solve the model-free Riccati equation, where the system dynamics are unknown. Unlike traditional approaches that rely on explicit system models, GA offers a data-driven optimization framework capable of handling complex, high-dimensional search spaces. However, applying GA in a model-free setting presents new challenges, particularly in ensuring stability and solution consistency. To address these issues, we adapt

the algorithm to incorporate stability constrains presented in (3.38). Building on insights gained from the development of the PSO algorithm in Section 3.5.1, we adopt a similar approach for GA by utilizing the fitness function defined in 3.47.

Algorithm 3.7: GA algorithm for a model-free system

```
Input: GA Parameters, Dataset of x and u
Output: Optimal Gain from the solution matrix P
1: Create an initial population and replace, not stabilize, individuals
2: Calculate δ_xx, I_xx, and I_xu from the dataset
3: Determine the fitness value function (3.41) as a function of [P, P_B]
4: Calculate the initial fitness value of each individual and sort the values
5: for generation = 1 to Max_generations do
    for i = 1 to Population_Size do
        Create a mutant vector for P and a mutant vector for PB
        Create a crossover vector for P and a crossover vector for PB
        fitness_new_P = Fitness_Value(Crossover_Vector_P, Best_PB)
        fitness_new_PB = Fitness_Value(Best_P, Crossover_Vector_PB)
        if Stability_Check = false then
            fitness_new_P = fitness_new_P + Penalty
        end
        if fitness_new_P < fitness_P(i) then
            Population_P(i,:) = Crossover_Vector_P
            fitness_P(i) = fitness_new_P
        end
        if fitness_new_PB < fitness_PB(i) then
            Population_PB(i,:) = Crossover_Vector_PB
            fitness_PB(i) = fitness_new_PB
        end
        best_fitness_P = min(fitness_P)
        best_fitness_PB = min(fitness_PB)
        best_P = population_P(best_fitness_P)
        best_PB = population_PB(best_fitness_PB)
    end
end
6: Calculate the Optimal Gain from the solution of the algorithm P
```

In the GA, unlike the PSO, we didn't need to make many adaptations; the solution

we got from the algorithm was consistent and stabilized the system.

Algorithm Type	Fitness value P	Fitness value P_B	Stability
GA	0.00127	0.00127	Yes
Optimal	0.000197	0.000197	Yes

Table 3.7: Algorithm solution for GA model-free compare to optimal

The results presented in Table 3.7 indicate the average outcomes from ten runs of the algorithm. The results demonstrate that GA successfully solved the model-free Riccati equation without requiring additional adaptations beyond the initial modifications. Unlike PSO, where significant adjustments were necessary to improve stability and consistency, GA naturally converged to a stable solution every run. The algorithm consistently produced reliable results, confirming its robustness in handling model-free optimization problems. These findings highlight GA's potential as an effective tool for solving Linear Parameter-Varying (LPV) model-free Riccati equations, providing a strong foundation for future research in data-driven control and optimization.

3.6 LPV Data-Driven Model-Free Solution of Riccai Equation

In this section, we extend the application of PSO and GA to solve the LPV model-free design problem expressed by a set of Riccati equations. Unlike the previous model-free case, where the system remained constant, this problem introduces additional complexity as the system parameters may change over time. Traditional methods, like LMI, require model knowledge that is assumed unavailable here, mak-

ing data-driven optimization approaches like PSO and GA more suitable. We aim to evaluate whether these algorithms can adapt to a concept of varying model parameters while maintaining stability and solution consistency. We will discuss the modifications required to handle system variations, analyze algorithm performance, and assess their potential for solving model-free LPV systems.

3.6.1 Data-Driven LPV Algorithm

Consider a model-free LPV system characterized by n time-varying parameters, each constrained within two prescribed boundaries, which form 2^n vertex solutions. Those parameters construct a polytope with 2^n vertices defining the parameter space.

3.6.1.1 GS Sub-Optimal Controller for Data-Driven LPV

A model-free Riccati equation can be derived for each vertex to represent a corresponding sub-optimal controller; see sub-section 3.2.2. Based on (3.37) and (3.16), we get 2^n fitness function, one for each vertex:

$$\begin{aligned} F_i(P, P_B, L_i) = \delta_{xxi} N \mathsf{svec}(P) - 2I_{xui}\mathsf{vec}(P_B) - I_{xxi}\mathsf{vec}(P_B^T R^{-1} P_B) \\ + I_{xxi}\mathsf{vec}(Q) + I_{xxi}\mathsf{vec}(L_i^T R L_i) \end{aligned} \tag{3.51}$$

The global fitness function can be written as:

$$V(P, P_B, L_i) = \sum_{i=1}^{2^n} \parallel F_i(P, P_B, L_i) \parallel_F^2 \tag{3.52}$$

Attempting to solve it this way may introduce significant noise from the L term. Therefore, our first step is to find the shared P and P_B that minimize the 2^n equa-

tions without considering the L_i term. After this, we can incorporate the L_i term (as an unknown) and solve (for L_i) each equation individually. The fitness equation can be written as:

$$\begin{aligned} F_i(P, P_B) = \delta_{xxi} N \mathsf{svec}(P) - 2I_{xui}\mathsf{vec}(P_B) - I_{xxi}\mathsf{vec}(P_B^T R^{-1} P_B) \\ + I_{xxi}\mathsf{vec}(Q) \end{aligned} \tag{3.53}$$

Applying the stability constraints denoted in (3.38). From (3.53) the global fitness function will be:

$$V(P, P_B) = \sum_{i=1}^{2^n} \parallel F_i(P, P_B) \parallel_F^2 \tag{3.54}$$

After determining the robust values of P and P_B that minimize all 2^n equations, we rerun the algorithm to find the L terms (separately for each vertex) to adjust the boundaries of our solution space.

$$\begin{aligned} F_i(L_i) = \delta_{xxi} N \mathsf{svec}(P) - 2I_{xui}\mathsf{vec}(P_B) - I_{xxi}\mathsf{vec}(P_B^T R^{-1} P_B) \\ + I_{xxi}\mathsf{vec}(Q) + I_{xxi}\mathsf{vec}(L_i^T R L_i) \end{aligned} \tag{3.55}$$

So the value function of each L_i is,

$$V_i(L_i) = \parallel F_i(Li) \parallel_F^2 \tag{3.56}$$

We use those equations to develop the model-free PSO and GA algorithm for the sub-optimal control problems. In an LPV system with n varying parameters, we have a 2^n vertex solutions. Based on (3.17), each vertex gain K_i is computed from

$$K_i = R^{-1} P_B + L_i \tag{3.57}$$

Based on subsection 3.1.1 we can ensure those gains are stabilizing and part of a convex combination. This notation of convex combinations offers a powerful method for performing the GS using interpolation.

3.6.1.2 Robust Controller for Data-Driven LPV

In robust control, like in GS, each vertex has a model-free Riccati equation, see sub-section 3.2.2, the difference is in the common L (i.e., now all vertices are solved for an identical L). Based on (3.37) and (3.18), we get 2^n fitness functions, one for each vertex,

$$\begin{aligned} F_i(P, P_B, L) = \delta_{xxi} N \mathsf{svec}(P) - 2I_{xui}\mathsf{vec}(P_B) - I_{xxi}\mathsf{vec}(P_B^T R^{-1} P_B) \\ + I_{xxi}\mathsf{vec}(Q) + I_{xxi}\mathsf{vec}(L^T R L) \end{aligned} \tag{3.58}$$

The global fitness function can be written as,

$$V(P, P_B, L) = \sum_{i=1}^{2^n} \parallel F_i(P, P_B, L) \parallel_F^2 \tag{3.59}$$

Based on (3.19) the robust gain is computed using the following

$$K = R^{-1} B^T P + L \tag{3.60}$$

3.6.2 LKS control by Model-Free LPV design

For a standard LPV convex combination, we require that the system matrices depend affinely on the scheduling parameters. Our suggested theoretical framework asusme only variation of the model matrix A. For the LKS control problem (2.40), we only

consider the case of converging to a straight road, hence the tracking error dynamics,

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-2C_{af}-2C_{ar}}{mV_x} & \frac{2C_{af}+2C_{ar}}{m} & \frac{-2C_{af}l_f+2C_{ar}l_r}{mV_x} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-2C_{af}l_f+2C_{ar}l_r}{I_zV_x} & \frac{2C_{af}l_f-2C_{ar}l_r}{I_z} & \frac{-2C_{af}l_f^2-2C_{ar}l_r^2}{I_zV_x} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2C_{af}}{m} \\ 0 \\ \frac{2C_{af}l_f}{I_z} \end{bmatrix} \delta \tag{3.61}$$

With $x_1 = e_1$ as the position tracking-error, $x_2 = e_2$ the orientation tracking-error and δ as the steering command. Since m and I_z are physically coupled, we assume, $I_z \cong 0.5m(l_f^2 + l_r^2)$, using this, the model becomes,

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-2C_{af}-2C_{ar}}{mV_x} & \frac{2C_{af}+2C_{ar}}{m} & \frac{-2C_{af}l_f+2C_{ar}l_r}{mV_x} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-4C_{af}l_f+4C_{ar}l_r}{V_xm(l_f^2+l_r^2)} & \frac{4C_{af}l_f-4C_{ar}l_r}{m(l_f^2+l_r^2)} & \frac{-4C_{af}l_f^2-4C_{ar}l_r^2}{V_xm(l_f^2+l_r^2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 2C_{af} \\ 0 \\ \frac{2C_{af}l_f}{(l_f^2+l_r^2)} \end{bmatrix} \frac{\delta}{m} \tag{3.62}$$

Where the varying parameters are the velocity and mass, the matrices have terms like $1/m$ and $1/V_x$, which means the dependence on m and V_x is nonlinear. To solve this issue, we define new scheduling parameters

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}, \quad \theta_1 = \frac{1}{mV_x}, \quad \theta_2 = \frac{1}{m} \tag{3.63}$$

that form the LPV model. To remove the dependency of the model matrix B in m, we define $u = \delta/m$. Now u can be designed as a GS or robust controller. After designing u we apply, $\delta = mu$.

3.6.3 Sub-Optimal PSO Framework for Linear Parameter-Varying Model-Free Systems

This section uses the PSO algorithm to compute gains with two approaches, Robust and GS controllers. Like in the previous section of the PSO algorithm, we encounter the same obstacles, such as inconsistency in the solutions and stability. Numerous attempts were made to improve the algorithm, including various types of PSO, constant adjustments, feasibility checks, and boundary identification using the gain equation, but none were successful. The infinite solution space and random initial population led to different incorrect solutions. However, we noticed that the algorithm will converge to identical values if the initial population is the same across runs. After many attempts, we realized the need to choose a better initial population while restricting the solution space. If we examine each system individually, we can utilize the algorithm developed in 3.5.1 to determine the optimal solution for each system, using this data to establish the initial population and the boundaries of the solution space. We now have each vertex optimal value of P and P_B. Next, we determine the maximum difference between the variables in each vector and the average value of those variables. We add and subtract this difference from the average to establish the boundaries of our solution space.

$$\begin{aligned} Threshold\_P &= max\_P - min\_P \\ Low\_P &= Average\_P - Threshold\_P \\ High\_P &= Average\_P + Threshold\_P \end{aligned} \tag{3.64}$$

We perform the same for P_B. Then, the initial population is selected from the average value to fit within these boundaries.

$$\begin{aligned} Particle\_P\{i\} &= Low\_P + (High\_P - Low\_P) \cdot \mathbf{rand}(\mathbf{size}(P)) \\ Particle\_P_B\{i\} &= Low\_P_B + (High\_P_B - Low\_P_B) \cdot \mathbf{rand}(\mathbf{size}(P_B)) \end{aligned} \tag{3.65}$$

We also want to ensure that the values of L do not push us outside the boundaries, so we apply the threshold of k to set upper and lower limits for L.

$$\begin{aligned} L\_Upper\_Limit &= threshold\_k \\ L\_Lower\_Limit &= -threshold\_k \end{aligned} \tag{3.66}$$

For the GS suboptimal LPV system, we provided the following algorithm.

Algorithm 3.8: GS PSO algorithm for LPV model-free system

```
Input: PSO Parameters, Dataset of x and u
Output: Four Suboptimal Gains
1: Create an initial population using the solution of Algorithm (3.6) for
   each vertex
2: Calculate δ_xx, I_xx, and I_xu from the dataset
3: Determine four fitness function (3.53) as a function of [P, P_B]
4: Determine the fitness value function (3.54)
5: Determine four fitness function (3.55) as a function of [L_i]
6: Determine four fitness value functions (3.56) with L
7: for iteration = 1 to Number_Of_Iterations do
    for i = 1 to Number_Of_Particles do
        fitness_P(i) = Fitness_Value(particles_P{i},G_Best_PB)
        fitness_PB(i) = Fitness_Value(G_Best_P,particles_PB{i})
        Stability_Check(particles_P{i})
        Boundaries_Check(particles_P{i},particles_PB{i})
        Personal_Best_Update(particles_P{i},particles_PB{i})
        Global_Best_Update(particles_P{i},particles_PB{i})
        Particle_Update(particles_P{i},particles_PB{i})
    end
    Adjusting the algorithm weights
end
8: for iteration = 1 to Number_Of_Iterations do
    for i = 1 to Number_Of_Particles do
        for j = 1 to Number_Of_Vertices do
            fitness_L_j(i) =
              Fitness_Value_L_j(G_Best_P,G_Best_PB,particles_L_j{i})
            Boundaries_Check(G_Best_P,G_Best_PB,particles_L_j{i})
            Personal_Best_Update(particles_L_j{i})
            Global_Best_Update(particles_L_j{i})
            Particle_Update(particles_L_j{i})
        end
    end
    Adjusting the algorithm weights
end
9: Calculate the sub-optimal Gains from the solution of the algorithm
```

After executing the algorithm ten times, we have obtained the following solution.

G_best P	G_best P_B	G_best L_1	G_best L_2	G_best L_3	G_best L_4
0.0020	0.0020	0.0030	0.0030	0.0030	0.0030

Table 3.8: Algorithm global best solutions for model-free varying PSO

The results presented in Table 3.8 indicate the average outcomes from ten runs of the algorithm. In conclusion, we can say that the PSO algorithm can find a stable and consistent solution for a suboptimal problem for the LPV system using the GS approach. The suboptimal gains represent

Vertex Gain	k_1	k_2	k_3	k_4
PSO k_1	1.5811	0.6783	8.8292	0.4593
optimal k_1	1.5811	0.5917	9.3969	0.4738
PSO k_2	1.5811	0.3405	6.3804	0.5845
optimal k_2	1.5811	0.4355	6.4273	0.5850
PSO k_3	1.5811	0.7425	8.3018	0.4849
optimal k_3	1.5811	0.5756	9.4216	0.4749
PSO k_4	1.5811	0.3933	7.1417	0.5567
optimal k_4	1.5811	0.4339	6.382	0.5670

Table 3.9: Algorithm gains for model-free LPV PSO

Where k_1 is the gain for the high mass and high velocity, k_2 is the gain for the high mass and low velocity, k_3 is the gain for low mass and high velocity, and k_4 is the gain for low mass and low velocity. The solution obtained from the Algorithm 3.8 satisfies the suboptimal solution.

For the robust suboptimal LPV system, we provided the following algorithm.

Algorithm 3.9: Robust PSO algorithm for LPV model-free system

Input: PSO Parameters, Dataset of x and u

Output: Four Suboptimal Gains

```
Create an initial population using the solution of Algorithm (3.6) for
 each vertex
Calculate δ_xx, I_xx, and I_xu from the dataset
Determine four fitness function (3.58) as a function of [P, P_B, L]
Determine the fitness value function "Fitness_Value" from (3.59)
for iteration = 1 to Number_Of_Iterations do
    for i = 1 to Number_Of_Particles do
        fitness_P(i) = Fitness_Value(particles_P{i},G_Best_P_B,G_Best_L)
        fitness_P_B(i) = Fitness_Value(G_Best_P,particles_P_B{i},G_Best_L)
        fitness_L(i) = Fitness_Value(G_Best_P,G_Best_P_B,particles_L{i})
        Stability_Check(particles_P{i})
        Boundaries_Check(particles_P{i},particles_P_B{i},particles_L{i})
        Personal_Best_Update(particles_P{i},particles_P_B{i},particles_L{i})
        Global_Best_Update(particles_P{i},particles_P_B{i},particles_L{i})
        Particle_Update(particles_P{i},particles_P_B{i},particles_L{i})
    end
    Adjusting the algorithm weights
end
Calculate the sub-optimal Gains from the solution of the algorithm
```

After executing the algorithm multiple times, we obtained the following solution.

PSO	k_1	k_2	k_3	k_4
Robust Gain	1.5811	0.6050	10.3647	0.5239

Table 3.10: Algorithm Robust gains for model-free LPV PSO

The results presented in Table 3.10 indicate the average outcomes from ten runs of the algorithm.

3.6.4 Sub-Optimal GA Optimization Framework for Linear Parameter-Varying Model-Free Systems

After demonstrating in section 3.5.2 that a GA can effectively solve a Riccati equation for a model-free fixed system, we now need to verify if it can also address a Riccati equation for a model-free LPV design. Taking notes from the development of the PSO algorithm 3.6.3, we will use the boundaries for the selection of the initial population, (3.64, 3.65).

Algorithm 3.10: GS GA for LPV model-free system

Input: GA Parameters, Dataset of x and u
Output: Four Suboptimal Gains

```
1: Create an initial population using the solution of Algorithm (3.7) for
   each vertex
2: Calculate δ_xx, I_xx, and I_xu from the dataset
3: Determine four fitness function (3.53) as a function of [P, P_B]
4: Determine the fitness value function (3.54)
5: Determine four fitness function (3.55) as a function of [L_i]
6: Determine four fitness value functions (3.56) with L
7: for generation = 1 to Max_generations do
    for j = 1 to Population_Size do
        Create a mutant vector for P and a mutant vector for P_B
        Create a crossover vector for P and a crossover vector for P_B
        fitness new P = Fitness_Value(Crossover Vector P, Best P_B)
        fitness new P_B = Fitness_Value(Best P, Crossover Vector P_B)
        Stability_Check(Crossover Vector P)
        Boundaries_Check(Crossover Vector P,Crossover Vector P_B)
        P Population Update(Population_P(j), Crossover_Vector_P)
        P_B Population Update(Population_P_B(j), Crossover_Vector_P_B)
        best_fitness = (min(fitness_P),min(fitness_P_B))
        best_P = population_P(best_fitness_P)
        best_P_B = population_P_B(best_fitness_P_B)
    end
    Adjusting the algorithm weights
end
8: for generation = 1 to Max_generations do
    for j = 1 to Population_Size do
        Create four mutant vectors for L_i
        Create four crossover vectors for L_i
        fitness new L_i(j) =
         Fitness_Value_L_i(best_P,best_P_B,Crossover_Vector_L_i)
        Boundaries_Check(Best_P,Best_P_B,Crossover_Vector_L_i)
        L_i Population Update(Population_L_i(i), Crossover_Vector_L_i)
        Particle_Update(particles_L_i{i})
    end
    Adjusting the algorithm weights
end
9: Calculate the Sub-Optimal Gains from the solution of the algorithm
```

After running the Algorithm multiple times, we observed that while the solution remained stable, the results were inconsistent. Upon debugging the code, we found that the fitness value did not change significantly during the run. Suppose we consider how GA functions, we realize that changes in the population are primarily influenced by two main factors: mutation and crossover. The mutation rate influences mutation, while the crossover rate influences crossover. Values that worked well in the previous section of the GA do not perform the same when the solution space is narrow. While looking at the PSO-AWDV [19], we thought to take this approach and adapt the parameters during the algorithm run. We base our adaptation on the JADE approach [22]. By that we track success rates over a memory window and introduce random perturbations to prevent stagnation. The mutation rate receives random perturbations and changes in each generation based on the following:

$$M_{rate} = M_{rate} + 0.1r_1 \tag{3.67}$$

The mutation vector depends on the global best and is computed as follows:

$$Mutation\_vector = G\_Best + M_{rate}(Parent_1 - Parent_2) \tag{3.68}$$

The crossover rate dynamically adjusts based on the Algorithm's success rate, which is determined by the proportion of offspring that outperform their parents. If the success rate falls below 20%, this indicates an excessive number of failed trials. Consequently, the Algorithm decreases the crossover rate to promote more conservative solutions, ensuring that new offspring closely resemble their parents. Conversely, if the success rate exceeds 20%, it signifies sufficient successful trials, prompting the Algorithm to increase the crossover rate to enhance exploration.

Algorithm 3.11: GS adaptive GA for a varying model-free system

```
Input: GA Parameters, Dataset of x and u
Output: Four Suboptimal Gains
1: Create an initial population using the solution of Algorithm (3.7) for
   each vertex
2: Calculate δ_xx, I_xx, and I_xu from the dataset
3: Determine four fitness function (3.53) as a function of [P, P_B]
4: Determine the fitness value function (3.54)
5: Determine four fitness function (3.55) as a function of [L_i]
6: Determine four fitness value functions (3.56) with L
7: for generation = 1 to Max_generations do
    Adapt parameters based on success rate
    for j = 1 to Population_Size do
        Sort the population and perform parent selection
        Create a mutant vector for P and a mutant vector for B
        Create a crossover vector for P and a crossover vector for B
        fitness new P = Riccati Fun(Crossover Vector P, Best B)
        fitness new B = Riccati Fun(Best P, Crossover Vector B)
        Boundaries_Check(Crossover Vector P,Crossover Vector B)
        P Population Update(Population_P(j), Crossover_Vector_P)
        B Population Update(Population_B(j), Crossover_Vector_B)
    end
    Memory update
    Update best solution
end
8: Repeat algorithm with L term
9: Calculate the Sub-Optimal Gains from the solution of the algorithm P,
   B, and L_i
```

Ten independent runs were conducted to assess the performance of the proposed adaptive genetic Algorithm 3.11. The results of these runs are summarized in Table 3.11. The table presents the fitness function value for each Algorithm execution, highlighting the best-obtained values for parameters P, P_B, and L_1 to L_4. From Table 3.11, it is evident that the Algorithm consistently converges to a narrow range of values across multiple runs, demonstrating its reliability in optimizing the gains.

Run No.	G_best P	G_best P_B	G_best L_1	G_best L_2	G_best L_3	G_best L_4
1	0.0020	0.0020	0.0027	0.0010	0.0023	0.0012
2	0.0020	0.0020	0.0028	0.0010	0.0023	0.0012
3	0.0020	0.0020	0.0028	0.0010	0.0023	0.0012
4	0.0020	0.0020	0.0028	0.0010	0.0024	0.0012
5	0.0020	0.0020	0.0028	0.0010	0.0023	0.0012
6	0.0023	0.0023	0.0033	0.0015	0.003	0.0013
7	0.0020	0.0020	0.0028	0.0010	0.0023	0.0012
8	0.0022	0.0022	0.0026	0.0017	0.0026	0.0015
9	0.0021	0.0021	0.0029	0.0010	0.0023	0.0012
10	0.0020	0.0020	0.0028	0.0010	0.0023	0.0015
Average	0.00206	0.00206	0.00287	0.00116	0.00244	0.00129

Table 3.11: Algorithm global best solutions for model-free varying GA

The minimal variation in results suggests robustness in the search process, even in a model-free LPV setup. To further evaluate the effectiveness of the genetic

Vertex Gain	k_1	k_2	k_3	k_4
GA K_1	1.5811	0.5982	8.9879	0.4151
optimal K_1	1.5811	0.5917	9.3969	0.4738
GA K_2	1.5811	0.3483	5.4795	0.5034
optimal K_2	1.5811	0.4355	6.4273	0.5850
GA K_3	1.5811	0.5888	8.8847	0.4550
optimal K_3	1.5811	0.5756	9.4216	0.4749
GA K_4	1.5811	0.3482	6.2201	0.5048
optimal K_4	1.5811	0.4339	6.382	0.5670

Table 3.12: Algorithm gains for model-free LPV GA

Algorithm, Table 3.12 compares the gains obtained using the GA approach and the optimal gains at each vertex. The comparison in the table indicates that the GA-derived gains are relatively close to the optimal values, validating the effectiveness of the proposed approach in determining suboptimal solutions.

For the robust suboptimal LPV system, we provided the following algorithm.

Algorithm 3.12: Robust GA for LPV model-free system

Input: GA Parameters, Dataset of x and u
Output: Robust Suboptimal Gains
1: Create an initial population using the solution of Algorithm (3.7) for each vertex
2: Calculate δ_{xx}, I_{xx}, and I_{xu} from the dataset
3: Determine four fitness function (3.58) as a function of $[P, P_B, L]$
4: Determine the fitness value function "Fitness_Value" from (3.59)
5: Calculate the initial fitness value of each individual and sort the values
6: **for** *generation = 1* **to** *Max_generations* **do**
    Adapt parameters based on success rate
    for *j = 1* **to** *Population_Size* **do**
        Sort the population and perform parent selection
        Create a mutant vector for P, P_B, L
        Create a crossover vector for P, P_B, L
        fitness new P = Fitness_Value(Crossover Vector P, Best P_B, Best L)
        fitness new P_B = Fitness_Value(Best P, Crossover Vector P_B, Best L)
        fitness new L = Fitness_Value(Best P, Best P_B, Crossover Vector L)
        Boundaries_Check(Crossover Vector P,Crossover Vector P_B, Crossover Vector L)
        P Population Update(Population_P(j), Crossover_Vector_P)
        P_B Population Update(Population_P_B(j), Crossover_Vector_P_B)
        L Population Update(Population_L(j), Crossover_Vector_L)
    end
    Memory update
    Update best solution
end
7: Calculate the Sub-Optimal Gain from the solution of the algorithm

After executing the algorithm multiple times, we obtained the following solution.

GA	k_1	k_2	k_3	k_4
Robust Gain	1.5811	0.5068	10.1782	0.5946

Table 3.13: Algorithm Robust gain for model-free LPV GA

The results presented in Table 3.13 indicate the average outcomes from ten runs of the algorithm.

Chapter 4

Performance Analysis

In this research, we sought to apply a data-driven gain scheduling control theory in a model-free LPV system. We used non-standard algorithms, such as PSO and GA, to base our theory. This chapter evaluates the performance of GA and PSO algorithms developed in 3.6.4 and 3.6.3, respectively. The comparative analysis methodology examines the performance of three distinct control approaches: a nominal LQR controller against two adaptive gain-scheduled controllers (derived from PSO and GA) and two robust controllers (also derived from PSO and GA optimization techniques). The experimental framework employs a structured evaluation protocol centered on a vehicle lateral dynamics model 3.6.2. The robustness evaluation of the control algorithms was conducted across a comprehensive parameter space, systematically varying two critical vehicle dynamics parameters. The longitudinal velocity V_x was modulated within the $10-40$ $[m/s]$ range, representing typical operational velocities encountered in highway driving scenarios. Concurrently, the vehicle mass parameter m varied between $1250-1573$ $[kg]$, accounting for different loading conditions signif-

icantly affecting the vehicle's inertial characteristics. As mentioned, the system dynamics were reformulated using scheduling parameters $\theta_1 = 1/(mV_x)$ and $\theta_2 = 1/m$ to address the inherent nonlinearities in the state-space representation. The nominal controller gains were computed based on θ_1 and θ_2 at the geometric center of the parameter space, specifically at ($V_x = 21.64$ $[m/s]$) and ($m = 1404$ $[kg]$). These parameters enabled the development of an LPV model that captures the velocity and mass-dependent characteristics of the system.

4.1 Performance Evaluation Framework

To evaluate the performance, the following value function has been used,

$$V = \int_0^{t_{span}} (x^T Q x + u^T R u) d\tau \tag{4.1}$$

The experimental evaluation protocol implemented a multi-controller simulation framework wherein all three control strategies (nominal LQR, PSO-based LPV, and GA-based LPV) were simulated concurrently within the same computational environment to ensure consistent comparison conditions. Multiple independent simulation trials were conducted using distinct initial state vectors to systematically evaluate controller robustness and performance consistency across varying starting conditions.

Sim No.	V_x	m	x_1	x_2	x_3	x_4
1	10	1250	-3.0	2.6	-0.45	-1.1
2	10	1573	-2.75	-2.75	-3.53	2.92
3	40	1250	4.31	-2.53	-0.53	0.85
4	40	1573	2.65	-2.30	-2.54	-2.53
5	15	1350	-1.56	0.19	-0.54	-1.67
6	15	1450	0.89	-2.88	-1.66	-1.06
7	35	1350	-3.73	-3.62	0.86	-2.63
8	25	1410	-0.35	2.28	-2.40	0.11

Table 4.1: Initial condition by simulation number

4.2 Performance Analysis

This section presents a comparative analysis of the obtained control gains for the vehicle lane-keeping system, where the GA and PSO methods were evaluated against the numerically computed nominal solution. The primary objective of this analysis is to assess the performance of the adaptive control gains obtained via convex interpolation within the gain set, particularly in how well they approximate the nominal solution under varying vehicle mass and velocity conditions, after the execution of the simulations with the initial conditions presented in table 4.1.

Sim No.	Nominal	GS PSO	GS GA	Robust PSO	Robust GA
1	35.1889	34.3355	34.5431	36.6216	35.9111
2	213.6245	206.4388	207.5734	223.1682	217.7047
3	47.4927	47.4315	46.9871	46.9451	47.1674
4	147.7055	147.0666	145.3395	144.4476	146.5599
5	13.3272	13.2103	13.3068	13.6561	13.4753
6	41.3618	40.5041	40.5978	43.4801	42.1094
7	44.9854	45.0751	44.7438	44.7203	44.7263
8	105.7238	106.6763	106.3131	107.0388	106.1432

Table 4.2: Value function simulation comparison

Table 4.2 compares the value function values across different simulations.

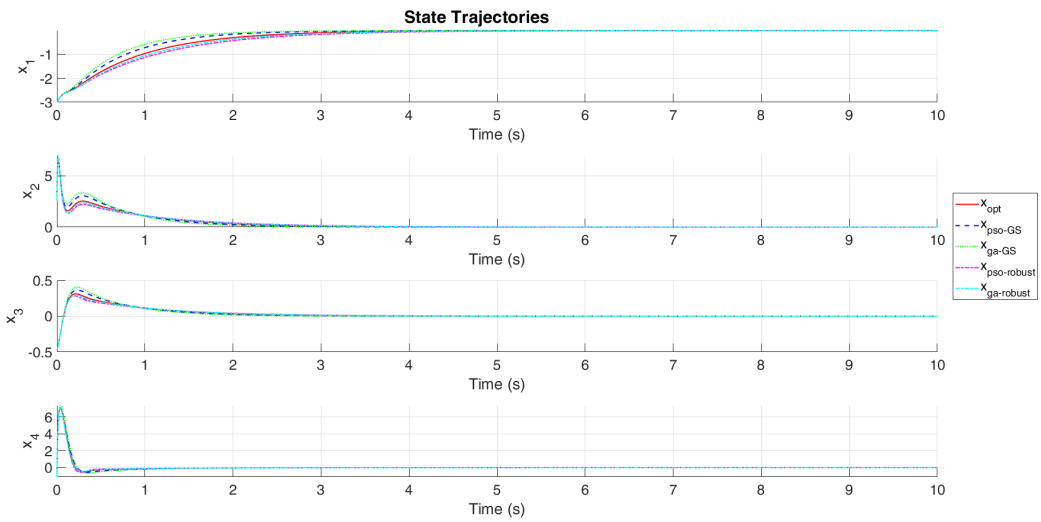


Figure 4.1: Comparison between the state trajectories per gain

To calculate the percentage of the improvement between the nominal gain to the PSO and GA gains, we will use the following:

$$\begin{aligned} Improve\_GA &= \frac{Nominal - GA}{Nominal} \cdot 100 \\ Improve\_PSO &= \frac{Nominal - PSO}{Nominal} \cdot 100 \end{aligned} \tag{4.2}$$

Sim No.	GS GA[%]	GS PSO[%]	Robust GA[%]	Robust PSO[%]
1	1.84	2.43	-2.05	-4.07
2	2.83	3.36	-1.91	-4.47
3	1.06	0.13	0.68	1.15
4	1.60	0.43	0.78	2.21
5	0.15	0.88	-1.11	-2.47
6	1.85	2.07	-1.81	-5.12
7	0.54	-0.20	0.58	0.59
8	-0.56	-0.90	-0.40	-1.24

Table 4.3: Percentage improvement of each algorithm compared to the nominal

This table shows the comparison between GA and PSO to the nominal. The higher the value, the higher proficiency against the nominal. A negative value shows that the nominal outperformed the other method in the current simulation.

Sim No.	GA vs PSO GS[%]	GA vs PSO Robust[%]
1	-0.59	2.02
2	-0.53	2.56
3	0.94	-0.47
4	1.17	-1.43
5	-0.72	1.36
6	-0.23	3.31
7	0.74	-0.01
8	0.34	0.85

Table 4.4: Percentage comparison between the algorithm of each algorithm compared to the nominal

This table shows the comparison between PSO and GA. Negative values mean PSO outperformed GA, while positive values mean GA outperformed PSO.

	Percentage Difference		Improvement Gap	
Sim No.	GS PSO−Rob PSO [%]	GS GA−Rob GA [%]	GS PSO−Rob PSO [%]	GS GA−Rob GA [%]
1	-6.24	-3.81	6.50	3.89
2	-7.50	-4.65	7.83	4.74
3	1.04	-0.38	-1.02	0.38
4	1.81	-0.83	-1.77	0.83
5	-3.26	-1.25	3.35	1.26
6	-6.84	-3.59	7.20	3.65
7	0.79	0.04	-0.79	-0.04
8	-0.34	0.16	0.34	-0.16
Avg	-2.57	-1.79	2.71	1.82

Table 4.5: Percentage comparison between Gain Scheduled and Robust algorithms

This table clearly shows two different comparisons between the GS and Robust algorithms:

1. **Percentage Difference:** This shows the direct percentage difference between the GS and Robust values, calculated as

$$\frac{(GS - Robust)}{Robust} \cdot 100\% \tag{4.3}$$

   Negative values indicate that the GS value is smaller than the Robust value (which is generally better since we're minimizing the cost function).

2. **Improvement Gap:** This shows the difference in improvement percentages (compared to nominal) between GS and Robust versions, calculated as

$$(GS\_improvement - Robust\_improvement) \tag{4.4}$$

   Positive values indicate that the GS version showed more significant improvement compared to the nominal controller than the Robust version did.

4.2.1 Comparison Between the Nominal Gain to GS GA and PSO Gains

We can compare the computed gains based on the cost calculated from the value function (4.1).

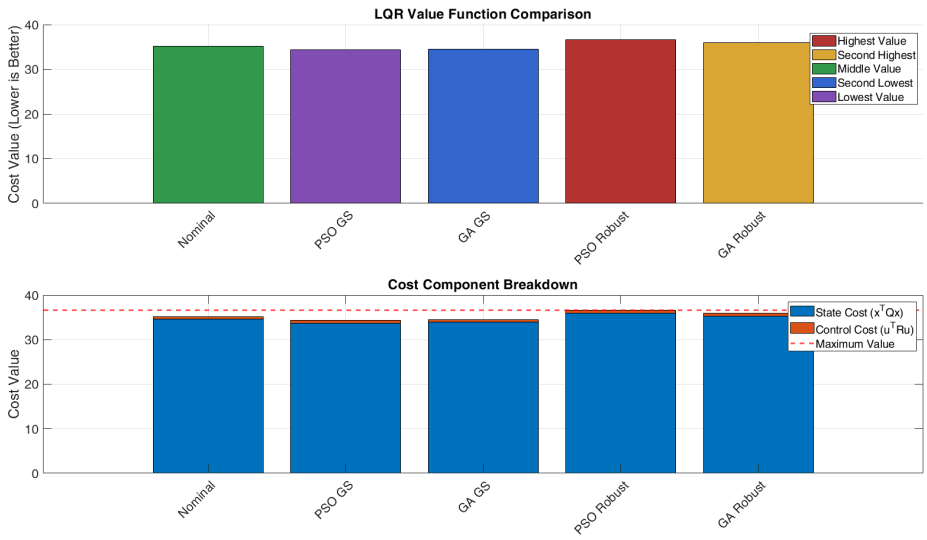


Figure 4.2: Comparison between the cost of the value function

4.2.1.1 Performance Analysis Framework

This comprehensive analysis evaluates five distinct control approaches for a vehicle lane-keeping system:

- Nominal LQR controller
- Gain Scheduled GA-based controller (GS GA)
- Gain Scheduled PSO-based controller (GS PSO)
- Robust GA-based controller
- Robust PSO-based controller

The evaluation framework employs a vehicle lateral dynamics model under varying operating conditions across the parameter space, with longitudinal velocity (V_x)

ranging from $10 - 40$ $[m/s]$ and vehicle mass (m) varying between $1250 - 1573[kg]$. Performance is quantified using the standard quadratic cost function (4.1).

4.2.1.2 Gain Scheduled Controllers versus Nominal Controller

The empirical results demonstrate that both gain scheduled controllers generally outperform the nominal controller, particularly in non-nominal operating conditions:

- In extreme parameter scenarios (simulations 1-4), both GS controllers show significant improvements over the nominal controller, with improvement percentages reaching up to 3.36% for GS PSO and 2.83% for GS GA.
- At intermediate parameter values (simulations 5-6), both adaptive controllers maintain their advantage, with GS PSO showing slightly superior performance (2.07% improvement) compared to GS GA (1.85% improvement).
- At high velocities (simulation 7), GS GA demonstrates moderate improvement (0.54%) while GS PSO slightly underperforms (-0.20%) compared to the nominal controller.
- Near the nominal design point (simulation 8), both adaptive controllers marginally underperform compared to the nominal controller, with performance decrements of -0.56% for GS GA and -0.90% for GS PSO.

Overall, GS demonstrates an average performance improvement across all test scenarios.

4.2.1.3 Robust Controllers versus Nominal Controller

The robust controller variants show a different performance pattern:

- The robust controllers underperform in low-velocity scenarios (simulations 1-2, 5-6), with performance degradation reaching -5.12% for Robust PSO and -2.05% for Robust GA.
- However, they demonstrate advantages in high-velocity scenarios (simulations 3-4, 7), with Robust PSO achieving improvements of up to 2.21%.
- Near the nominal point (simulation 8), both robust controllers slightly underperform compared to the nominal controller.

This pattern suggests that robust controllers don't have a significant advantage over the nominal controller.

4.2.1.4 Comparison Between GA and PSO Implementations

Comparing the GA and PSO implementations reveals interesting patterns:

- For gain scheduled controllers, PSO outperforms GA in low-velocity scenarios (simulations 1-2, 5-6), with performance advantages of up to 0.72%.
- Conversely, GA outperforms PSO in high-velocity scenarios (simulations 3-4, 7-8), with advantages reaching 1.17%.
- For robust controllers, the pattern is largely reversed—GA generally outperforms PSO in low-velocity scenarios (simulations 1-2, 5), while PSO shows advantages in high-velocity scenarios (simulations 3-4).

This suggests complementary strengths between the algorithms, with PSO's particle convergence mechanism potentially providing advantages in certain regions of the parameter space, while GA's evolutionary mechanisms excel in others.

4.2.1.5 Gain Scheduled versus Robust Implementations

The direct comparison between gain scheduled and robust implementations reveals a clear performance distinction. Gain scheduled controllers consistently outperform their robust counterparts in low-velocity scenarios (simulations 1-2, 5-6), with improvement gaps of up to 7.83% for PSO and 4.74% for GA. In high-velocity scenarios (simulations 3-4, 7), robust controllers demonstrate competitive or slightly superior performance. On average, gain scheduled implementations show 2.71% better improvement for PSO and 1.82% better improvement for GA compared to their robust counterparts.

This indicates that the gain scheduled approach provides superior overall adaptability across the parameter space, while robust controllers may offer advantages in specific high-velocity operating conditions.

4.2.1.6 Performance Evaluation - Conclusions and Insights

The presented analysis yielded several insights:

1. **Adaptive Superiority:** Both gain scheduled controllers demonstrate consistent performance advantages over the nominal controller in non-nominal operating conditions, validating the efficacy of data-driven gain scheduling in model-free LPV systems.

2. **Algorithm-Specific Strengths:** GA and PSO exhibit complementary strengths across the parameter space, with PSO typically excelling in low-velocity scenarios and GA showing advantages in high-velocity conditions.

3. **Implementation Trade-offs:** Gain scheduled implementations provide superior overall performance compared to robust implementations, particularly in low-velocity scenarios, while robust controllers may offer advantages in specific high-velocity operating conditions.

4. **Practical Design:** The results suggest that hybrid approaches combining the strengths of both algorithms and implementation strategies could potentially yield further performance improvements across the full operating envelope.

These findings validate the hypothesis that data-driven gain scheduling methods can effectively approximate optimal control solutions across a continuous parameter space without explicit system identification, with meta-heuristic techniques providing a computationally tractable approach to this complex optimization problem.

Chapter 5

Discussion and Conclusions

This thesis has presented a novel data-driven framework for determining suboptimal gain scheduling and robust controllers for Linear Parameter Varying (LPV) systems in model-free contexts. The primary achievement has been the successful development of methodologies that eliminate the need for explicit system identification while maintaining robust performance across varying operating conditions. The core contributions of this research can be summarized as follows:

1. Model-free representation of the Riccati equations, enabling optimal control principles to be applied in scenarios where system dynamics are unknown or difficult to model accurately. Compared to other data-driven methods, such as ADP (adaptive dynamic programming), our method does not require an initial stabilizing controller.

2. Successful application of evolutionary optimization algorithms—specifically Genetic Algorithms (GA) and Particle Swarm Optimization (PSO)—to solve

the model-free Riccati equation both for fixed and parameter-varying systems.

3. Data-driven approach to solving control design problems for LPV systems using robust and gain-scheduling controllers.

4. Comparative analysis of the performance of GA and PSO in solving model-free control problems across various scenarios, specifically conducted on the lane-keeping control problem example.

5.1 Discussion of Results

5.1.1 Computational Efficiency and Solution Quality

The experimental results presented in Chapter 4 revealed several significant findings regarding the performance of the proposed approach. Both gain-scheduled and robust controllers successfully converged to stable solutions for the model-free Riccati equation, although with different characteristics across the parameter space.

The gain-scheduled controllers demonstrated noticeable improvements over the nominal LQR controller, particularly in non-nominal operating conditions, with improvements of up to 3.36% for GS PSO and 2.83% for GS GA in extreme parameter scenarios. This confirms the effectiveness of the data-driven gain scheduling approach in adapting to varying system dynamics without explicit model knowledge.

The robust controllers (which are generally simpler to implement than the GS controllers) exhibited a different performance pattern, showing competitive or slightly superior performance in high-velocity scenarios but underperforming in low-velocity conditions. This highlights the complementary nature of these control strategies and

suggests potential benefits in developing hybrid (or mixed) approaches that leverage the strengths of both methodologies (as a first step in this approach, the reader is referred to subsection 3.2.3).

5.1.2 Algorithm-Specific Performance Characteristics

A notable finding from the comparative analysis is the complementary performance patterns exhibited by GA and PSO implementations. For gain-scheduled controllers, PSO consistently outperformed GA in low-velocity scenarios, with performance advantages of up to 0.72%, while GA demonstrated superior performance in high-velocity conditions, with advantages reaching 1.17%.

Interestingly, this pattern was reversed mainly for robust controllers, with GA generally outperforming PSO in low-velocity scenarios and PSO showing advantages in high-velocity conditions. This suggests that the inherent search mechanisms of these algorithms interact differently with the solution landscape depending on the control strategy and operating conditions.

The contrasting performance characteristics of these algorithms across different regions of the parameter space highlight the potential value of algorithm selection based on specific application requirements or operating conditions. Furthermore, these findings suggest that hybrid algorithms combining elements of both GA and PSO might offer improved performance across the entire operating envelope.

5.1.3 Gain-Scheduled versus Robust Control Strategies

A key contribution of this research is the systematic comparison between gain-scheduled and robust control implementations for model-free LPV systems. The results clearly demonstrate that gain-scheduled controllers consistently outperform their robust counterparts in low-velocity scenarios, with improvement gaps of up to 7.83% for PSO and 4.74% for GA.

In high-velocity scenarios, robust controllers demonstrated competitive or slightly superior performance, suggesting they may be optimized for these operating conditions at the expense of performance in low-velocity regimes. On average, gain-scheduled implementations showed 2.71% better improvement for PSO and 1.82% better improvement for GA compared to their robust counterparts.

This performance distinction highlights the fundamental trade-off between these control strategies: gain-scheduled controllers offer superior adaptability across the parameter space by continuously adjusting control gains based on current operating conditions. In contrast, robust controllers prioritize stability and performance in specific regions of the parameter space without requiring real-time parameter measurements.

5.2 Limitations and Challenges

Despite the promising results, several limitations and challenges were encountered during this research.

5.2.1 Algorithm Convergence and Stability

While GA and PSO are generally effective for optimization, their convergence speed and solution quality can be sensitive to parameter tuning (e.g., population size, mutation rates, inertia coefficients). The proper selection of these hyperparameters often requires empirical experimentation. Both GA and PSO exhibited convergence issues in their standard implementations, requiring significant modifications to ensure reliable and stable solutions. The stability constraints introduced in Algorithms 3.6 and 3.10 were essential for directing the search toward stabilizing solutions, but they also increased the computational complexity of the optimization process.

The challenge of inconsistent solutions in PSO was particularly notable, requiring incorporating adaptive parameters based on population diversity. While these modifications successfully addressed the convergence issues, they introduced additional hyperparameters that required careful tuning, potentially limiting the generalizability of the approach.

5.2.2 Data Dependency

The data-driven nature of the approach necessitates sufficient exploration of the system's operating range to ensure that the resulting controllers perform well across the entire parameter space. The quality and quantity of the training data directly impact the optimized controllers' performance. In practical applications, collecting comprehensive data that covers the entire operating envelope may be challenging, particularly for safety-critical systems.

5.3 Future Research Directions

The findings and limitations of this research suggest several promising avenues for future investigation:

5.3.1 Hybrid Optimization Approaches

Given the complementary strengths of GA and PSO, exploring hybrid optimization approaches that combine the global search capabilities of GA with the efficient convergence properties of PSO could potentially enhance both the quality and efficiency of the optimization process. Such hybrid algorithms could adaptively switch between different search strategies based on the characteristics of the solution space.

5.3.2 Extension to Nonlinear Systems

While this thesis focused on LPV systems, extending the methodology to nonlinear systems represents a natural progression, as heuristic search methods, as presented here, do not require linear models for their operations. Nevertheless, it will require a different formulation of stability and performance criteria.

5.3.3 Incremental Learning and Adaptation

The current approach performs optimization offline, with the resulting gains deployed through interpolation during online operation. Exploring incremental learning techniques that continuously refine the control gains based on real-time data

could enhance the system's adaptability to unforeseen operating conditions or time-varying dynamics.

5.3.4 Application to More Complex Systems

It would further demonstrate its practical utility by validating the approach on more complex, real-world systems beyond the vehicle lane-keeping case study. Potential applications include aircraft control, robotic manipulation, process control, and energy systems, where system dynamics are often complex and time-varying.

5.4 Concluding Remarks

This thesis has introduced a novel data-driven framework for suboptimal controller design for systems with LPV models, bridging the gap between classical optimal control theory and the practical reality of systems with uncertain or complex dynamics. By leveraging evolutionary optimization algorithms to identify suboptimal control gains directly from input-output data, the approach provides a practical solution for controlling systems where explicit modeling is challenging or not feasible.

The method presented is accessible and intuitive, distinguishing it from other methods for solving a data-driven model-free LPV problem. This evolutionary optimization approach can explore non-convex solution spaces more freely, potentially finding better-performing controllers that more conservative formulations might miss. The computation overload is more affected by the data processing effectiveness than the number of unknown parameters.

The empirical validation using a vehicle lane-keeping system demonstrated that both gain-scheduled and robust controllers consistently outperform the nominal controller in non-nominal operating conditions, confirming the practical utility of such designs. The superiority of these controllers, with performance improvements of up to 3.36%, validates the hypothesis that these data-driven methods can effectively approximate optimal control solutions across a continuous parameter space without explicit system identification. The comparative analysis between gain-scheduled and robust control strategies revealed essential insights into their relative strengths across different operating conditions. In general, GS showed stronger performance at the expense of more implementation complexity and the requirement of real-time information on parameter variation.

In conclusion, this research represents a significant step toward bridging the gap between theoretical optimal control and practical implementation in complex, real-world systems. By eliminating the need for explicit system identification while maintaining robust performance across varying operating conditions, the proposed approach offers a promising alternative to traditional model-based control design methods.

Bibliography

[1] Rajamani, R. *Vehicle Dynamics and Control*. Mechanical Engineering Series. Springer US, 2012. ISBN 9781461414339. URL `https://books.google.co.il/books?id=cZJFDox4KuUC`.

[2] Jiang, Y. and Jiang, Z.P. *Robust Adaptive Dynamic Programming*. IEEE Press Series on Systems Science and Engineering. Wiley, 2017. ISBN 9781119132646.

[3] Samadzadeh, S and Mazinan, AH. Lmi-based lpv control strategy considering uav systems. *Spatial Information Research*, 27(4):425–431, 2019.

[4] Quan, Ying Shuai, Kim, Jin Sung, and Chung, Chung Choo. Linear parameter varying models-based gain-scheduling control for lane keeping system with parameter reduction. *IEEE Transactions on Intelligent Transportation Systems*, 23(11):20746–20756, 2022. doi: 10.1109/TITS.2022.3188838.

[5] Miller, Jared and Sznaier, Mario. Data-driven gain scheduling control of linear parameter-varying systems using quadratic matrix inequalities. *IEEE Control Systems Letters*, 7:835–840, 2022.

[6] Bao, Yajie and Velni, J Mohammadpour. An overview of data-driven mod-

eling and learning-based control design methods for nonlinear systems in lpv framework. In *Proc. of the 5th IFAC Workshop on Linear Parameter Varying Systems*, 2022.

[7] Tran, Gia Quoc Bao, Pham, Thanh-Phong, Sename, Olivier, and Gáspár, Péter. Design of an lmi-based polytopic lqr cruise controller for an autonomous vehicle towards riding comfort. *Periodica Polytechnica Transportation Engineering*, 51 (1):1–7, 2023.

[8] Evren, Sanem and Unel, Mustafa. Stabilization of a pan-tilt system using a polytopic quasi-lpv model and lqr control. In *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 636–641. IEEE, 2016.

[9] Alcalá, Eugenio, Puig, Vicenç, Quevedo, Joseba, and Escobet, Teresa. Gain scheduling lpv control scheme for the autonomous guidance problem using a dynamic modelling approach. *arXiv preprint arXiv:1712.00390*, 2017.

[10] Mejari, Manas, Gupta, Ankit, and Piga, Dario. Data-driven computation of robust invariant sets and gain-scheduled controllers for linear parameter-varying systems. *IEEE Control Systems Letters*, 7:3355–3360, 2023.

[11] Modares, Amir, Sadati, Nasser, and Modares, Hamidreza. Data-driven safe gain-scheduling control. *Asian Journal of Control*, 25(6):4171–4182, 2023.

[12] Adıgüzel, Fatih, Kurtuluş, Kadircan, and Türker, Türker. A gain scheduling attitude controller with nn supervisor for quadrotor uavs. *International Journal of Control, Automation and Systems*, 22(12):3777–3791, 2024.

[13] Xia, Yang, Xu, Yan, Wang, Yu, Yao, Weitao, Mondal, Suman, Dasgupta, Souvik, Gupta, Amit K, and Gupta, Gaurav M. A data-driven method for online gain scheduling of distributed secondary controller in time-delayed microgrids. *IEEE Transactions on Power Systems*, 39(3):5036–5049, 2023.

[14] Broxson, Bobbi Jo. The kronecker product, 2006.

[15] Weisstein, Eric W. Frobenius norm. `https://mathworld.wolfram.com/FrobeniusNorm.html`, 2024.

[16] MathWorks. What is the genetic algorithm: an overview. `https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html`, 2023.

[17] Kora, Padmavathi and Yadlapalli, Priyanka. Crossover operators in genetic algorithms: A review. *International Journal of Computer Applications*, 162 (10), 2017.

[18] Wang, Dongshu, Tan, Dapei, and Liu, Lei. Particle swarm optimization algorithm: an overview. *Soft computing*, 22:387–408, 2018.

[19] Xu, Lin, Song, Baoye, and Cao, Maoyong. An improved particle swarm optimization algorithm with adaptive weighted delay velocity. *Systems Science & Control Engineering*, 9(1):188–197, 2021.

[20] Rugh, Wilson J. and Shamma, Jeff S. Research on gain scheduling. *Automatica*, 36(10):1401–1425, 2000. ISSN 0005-1098. doi: https://doi.org/10.1016/S0005-1098(00)00058-3. URL `https://www.sciencedirect.com/science/article/pii/S0005109800000583`.

[21] Xu, Jiucheng, Xu, Shihui, Zhang, Lei, Zhou, Changshun, and Han, Ziqin. A particle swarm optimization algorithm based on diversity-driven fusion of opposing phase selection strategies. *Complex & Intelligent Systems*, 9(6):6611–6643, 2023.

[22] Zhang, Jingqiao and Sanderson, Arthur C. Jade: adaptive differential evolution with optional external archive. *IEEE Transactions on evolutionary computation*, 13(5):945–958, 2009.

תוכן עניינים

הגברים משתנים במערכת לא ידועה עבור מערכות לינאריות עם פרמטרים משתנים

גלעד שאול

עבודת גמר לתואר מוסמך למדעים

אוניברסיטת בן־גוריון בנגב

2025

תקציר

עבודת תזה זו מציגה מסגרת חדשנית מבוססת נתונים לקביעת בקרת תזמון הגברים תת־אופטימלית של מערכות לינאריות עם פרמטרים משתנים LPV בהקשר נטול מודל. באופן מסורתי, מערכות LPV נפתרות באמצעות אי־שוויונים מטריציאליים לינאריים LMIs כאשר מודל המערכת מוגדר היטב. עם זאת, השגת מודלים מדויקים למערכות מורכבות מהעולם האמיתי עשויה להיות מאתגרת או בלתי אפשרית. הגישה המוצעת מתמודדת עם מגבלה זו באמצעות ניצול אלגוריתמי אופטימיזציה אבולוציוניים, בפרט אלגוריתמים גנטיים GA ואופטימיזציית נחיל חלקיקים PSO, לזיהוי הגברי בקרה תת־אופטימליים ישירות מנתוני קלט־פלט מבלי להסתמך על דינמיקת מערכת מפורשת.

החידוש המרכזי טמון בניסוח בעיית הבקרה כמזעור של גרסה נטולת מודל של משוואת ריקטי, אשר נפתרת באמצעות GA ו־PSO הממצא המרכזי הוא קבוצה של

הגברים תת־אופטימליים היוצרים קבוצה קמורה במרחב הפרמטרים. מבנה קמור זה מאפשר את יישום תזמון ההגברים. כאשר פרמטרי המערכת משתנים בגבולות ידועים, הגברי הבקר מתעדכנים באמצעות אינטרפולציה בתוך הקבוצה הקמורה. זה מאפשר לבקר להסתגל לדינמיקה המשתנה בזמן אמת ללא צורך באופטימיזציה מחדש.

סימולציות מקיפות וניתוחי ביצועים על מקרה בוחן של מערכת שמירת נתיב ברכב LKS מאמתים את היעילות של הגישה המוצעת. הבקרים מבוססי GA ו־PSO עם תזמון הגברים עולים בביצועיהם על בקר LQR קבוע נומינלי ועל הגברים רובוסטיים, ומציגים מעקב ויציבות עדיפים תחת שינויי פרמטרים.

עבודה זו משמעותית בשני אופנים. מבחינה תיאורטית, היא מרחיבה עקרונות בקרה אופטימלית למערכות LPV נטולות מודל, מגשרת על הפער בין התחום המבוסס של בקרה אופטימלית לבין המציאות המאתגרת של מערכות עם אי־ודאויות במודל. מבחינה מעשית, היא מספקת גישה יעילה מבחינת נתונים וישימה מבחינה חישובית ליישום בקרה אדפטיבית במערכות מורכבות על ידי העברת האופטימיזציה לשלב הלא־מקוון ושימוש באינטרפולציה לעדכוני הגברים בזמן אמת.

תזה זו פותחת אפיקים חדשים למחקר בנושא בקרה אדפטיבית מבוססת נתונים, אופטימיזציה אבולוציונית לכוונון בקרים, והסקת מודלים מקורבים של LPV מנתונים. היא מציעה אלטרנטיבה מבטיחה לגישות מבוססות LMI למצבים בהם דינמיקת מערכת מפורשת אינה זמינה, וסוללת את הדרך ליישום רחב יותר של בקרת תזמון הגברים במערכות מהעולם האמיתי.

אוניברסיטת בן־גוריון בנגב
הפקולטה למדעי ההנדסה
המחלקה להנדסת מכונות

הגברים משתנים במערכת לא ידועה עבור מערכות לינאריות עם פרמטרים משתנים

חיבור זה מהווה חלק מהדרישות לקבלת התואר מגיסטר בהנדסה (M.Sc)

גלעד שאול

בהנחיית **שי ארוגטי**

מרץ 2025