Ingenieurfakultät Bau Geo Umwelt

Solution Compensation Spaces and Optimal Constraints Relaxation in Systems Design with Application to Vehicle Dynamics.

Marc Eric Vogt

Vollständiger Abdruck der von der Ingenieurfakultät Bau Geo Umwelt der Technischen Universität München zur Erlangung des akademischen Grades eines Doktor-Ingenieurs genehmigten Dissertation.

Vorsitzender: **Prof. Dr.-Ing. Kai-Uwe Bletzinger**

Prüfende der Dissertation:

1. **Prof. Dr. Fabian Duddeck**

2. **Prof. Dr. Markus Zimmermann**

3. **Prof. Dr. Karsten Urban**

Die Dissertation wurde am 26.07.2019 bei der Technischen Universität München eingereicht und durch die Ingenieurfakultät Bau Geo Umwelt am 28.11.2019 angenommen.

Preface

This thesis is based on the results obtained during my time as a PhD student. Between the years 2015 and 2019 I was involved in a joint research project between the Technical University of Munich (TUM) and the BMW Group as a member of both the TUM Graduate School and the BMW ProMotion Program in Munich. At the TUM I participated mainly in the activities of the research group of Prof. Dr.-Ing. habil. Fabian Duddeck, Associate Professorship of Computational Mechanics, while at BMW I worked in the development department for driving dynamics, preliminary design.

I would like to express my deep gratitude to my supervising professor Fabian Duddeck for giving me the opportunity to join his research group, providing me support whenever needed and enriching my work with a critical review and fruitful discussions.

I would like to express my deep appreciation to Dr. Martin Wahle for supervising me at BMW Group. I particularly appreciate that he gave me the opportunity to do research in an industrial environment and that he shared his extensive knowledge of driving dynamics and method development with me. Furthermore, I would like thank him for the numerous valuable discussions we have had and still have together.

I would also like to express my deep gratitude to Prof. Dr. Markus Zimmermann who shared his expertise in Solution Spaces, in which he has done extensive research in recent years. I particularly appreciate the huge amount of time and effort he invested into supporting my work. The numerous valuable discussions we had have contributed decisively to the ideas presented in this thesis.

I would like to thank Prof. Dr. Karsten Urban, Ulm University, for agreeing to be part of the committee.

Furthermore, I would like to thank Dr. Johannes Fender, for being my mentor during my time as a PhD student.

I would also like to thank all of my colleagues at the TUM as well as at BMW for supporting me with helpful discussions, technical support and encouragement. Special thanks to my colleagues, Marco Daub, Stefan Erschen, Christian Schulz, Jan-Dominik Korus, Mario Weinberger, Jens Wimmler and Amir Zare.

I would like to thank my family and friends for their valuable comments, and would like to express my deepest gratitude to my wife Anna Vogt and my two wonderful children, Mila and Neo, for their unwavering support.

I would also like to thank Anja Lindner for proof reading my thesis and her valuable comments.

Last but not least, I would like to emphasize the work done by the numerous interns and Master candidates, special thanks to Rilian Shao and Florian Cyril Stutz. Without their work this thesis would not have been possible.

Marc Eric Vogt
Munich, 2019

Abstract

In the early development phase of a complex product, many design parameters are subject to uncertainty due to lack-of-knowledge. This uncertainty can be dealt with using so-called Solution Spaces. Solution Spaces are sets of good designs that by definition reach all design goals. By considering sets of designs rather than a single design, uncontrollable variations of component properties that are typical in the early stages of systems design are allowed. Box-shaped Solution Spaces can be expressed as the Cartesian product of the design variables' permissible intervals. These intervals serve as decoupled target regions and can be interpreted as component requirements. Existing algorithms optimise the size of box-shaped Solution Spaces. Unfortunately, the sizes of the permissible intervals of crucial design variables are often not large enough to encompass all uncertainty and to ensure feasibility. In extreme cases not even a single feasible solution can be obtained.

A new approach called *Solution-Compensation Spaces* (B-corridors) is introduced in order to enlarge the permissible intervals. The design variables are divided into a set of early-decision (A-variables) and a set of late-decision variables (B-variables). Early-decision variables are associated with intervals on which they may assume any value in order to encompass uncertainty due to limited controllability. Late-decision variables are controllable and therefore associated with intervals on which they can be adjusted to any specific value. The Cartesian product of these intervals is called a Solution-Compensation Space. It has the property that for all values of early-decision variables within their permissible intervals there exists at least one set of late-decision variable values within their intervals such that the resulting design reaches all design goals.

In case the resulting interval sizes are still too small, *Optimal Constraint Relaxation* for Solution Spaces or Solution-Compensation Spaces can be applied. This newly introduced method is able to determine the minimal set of changes which needs to be applied to the constraints in order to make any arbitrarily strict Solution Space problem feasible. The set of proposed changes to the constraints is optimal with respect to a target function that weighs the relaxation of the different constraints and aims to relax each constraint as little as possible.

These approaches are applied to a chassis design problem. It is shown that by applying the Solution-Compensation Space approach, the permissible intervals of the early-design variables can be increased significantly. In case these intervals remain too small, we can obtain the desired interval size by applying Optimal Constraint Relaxation for Solution Spaces or Solution-Compensation Spaces.

Marc Eric Vogt

Zusammenfassung

In der frühen Entwicklungsphase eines komplexen Produkts sind viele Designparameter aufgrund von fehlendem Wissen mit Unbestimmtheiten behaftet. Diese Unbestimmtheiten können mit sogenannten Lösungsräumen behandelt werden. Lösungsräume stellen eine Menge an guten Designs dar, welche per Definition alle Designziele erreichen. Die Betrachtung von vielen Designs ermöglicht es, nicht-kontrollierbare Variationen der Bauteileigenschaften, wie sie in den frühen Phasen des Systementwurfs typisch sind, abzufangen. Boxförmige Lösungsräume können als das kartesische Produkt aus zulässigen Intervallen für Designvariablen ausgedrückt werden. Diese Intervalle dienen als entkoppelte Zielregionen und können als Komponentenanforderungen interpretiert werden. Bestehende Algorithmen optimieren die Größe von boxförmigen Lösungsräumen. Leider ist die Größe der zulässigen Intervalle für entscheidende Auslegungsvariablen oft nicht groß genug, um alle Unbestimmtheiten abzudecken und die Machbarkeit zu gewährleisten. Im Extremfall kann nicht einmal eine einzige umsetzbare Lösung gefunden werden.

Deshalb wird eine neue Methode eingeführt, die sogenannten B-Korridore (*Solution-Compensation Spaces*), um die zulässigen Intervalle zu vergrößern. Dafür werden die Designvariablen in einen Satz von Typ-A- und Typ-B-Variablen unterteilt. Typ-A-Variablen sind zulässigen Intervallen zugeordnet, in denen sie aufgrund ihrer begrenzten Kontrollierbarkeit jeden beliebigen Wert annehmen dürfen. Typ-B-Variablen sind kontrollierbar und damit Intervallen zugeordnet, in denen sie auf einen bestimmten Wert eingestellt werden können. Diese Intervalle werden als B-Korridore bezeichnet. Dieser hat die Eigenschaft, dass für alle Werte der Typ-A-Variablen aus ihren zulässigen Intervallen mindestens ein Satz von Typ-B-Variablen aus ihren Intervallen existiert, so dass das resultierende Design alle Designziele erreicht.

Falls die resultierenden Intervallgrößen immer noch nicht groß genug sind, kann die Methode der Optimalen Relaxation der Randbedingungen (*Optimal Constraint Relaxations*) angewendet werden. Diese neu eingeführte Methode ist in der Lage, einen minimalen Satz von Änderungen an den Anforderungen zu bestimmen, um jedes beliebig strenge Lösungsraumproblem umsetzbar zu machen. Der Satz von Änderungen ist optimal in Bezug auf eine Zielfunktion, die die Relaxation der verschiedenen Anforderungen gewichtet und darauf abzielt, jede Anforderung so wenig wie möglich zu entspannen.

Diese Methoden werden auf ein Fahrwerkdesignproblem angewendet. Es wird gezeigt, dass durch die Anwendung des B-Korridoransatzes die zulässige Intervallgröße für die Typ-A-Variablen deutlich erhöht werden kann. Falls diese Intervalle immer noch zu klein sind, bringen wir sie auf die gewünschte Länge, indem wir die Methode der Optimalen Relaxation der Randbedingungen anwenden.

Contents

Chapter 1

Introduction

In this chapter, we discuss the motivation behind this thesis, which is the aim to increase flexibility and robustness during the early stages of complex product design, such as the design of passenger vehicles. We briefly explain fundamental concepts such as the decomposition of technical systems, the V-model approach, target cascading as well as the term robustness, as it is used in this thesis. The aim of the thesis is outlined and the methods used to reach that aim are listed. At the end of the chapter, an overview of the structure of this thesis is given.

1.1. Context and motivation

The performance of a vehicle with respect to driving dynamics depends on the properties of several interacting components, such as tyres, suspensions, and dampers. A major challenge is that many design variables influence different vehicle characteristics at the same time. The complexity grows steadily as both, the number of derivatives and the cost-driven necessity for high communality between the vehicle derivatives increase. Especially in the early development phase, many constraints remain subject to large uncertainties due to the lack of knowledge about the final state of the complex vehicle architecture [82]. Hence, lack-of-knowledge or epistemic uncertainty in technical systems needs to be considered. Classical design approaches like incremental and iterative development optimise a single design in order to meet all requirements [43]. Unfortunately, it is not guaranteed for these methods to converge and the use of it is "expensive, inefficient and vulnerable to uncertainty and variability" [55]. An effective way to cope with these problems as well as to treat the uncertainty are set-based design approaches. These seek permissible sets of designs. One example is seeking intervals for the design variables rather than a single optimal solution, e.g. [29, 57, 62]. Considering sets rather than one single design allows for variations of component properties later in the development.

This thesis focuses on the computation of a certain type of set-based designs, so-called Solution Spaces. The idea to compute box-shaped Solution Spaces was already introduced in 1983 by Swaney & Grossmann [68]. They used the method to analyse flexibility in chemical process design. Since the complete Solution Space can have any arbitrary non-linear, non-convex, non-connected shape and is often high-dimensional, it is difficult to compute and describe it. Hence, box-shaped Solution Spaces can be computed instead. As shown in figure 1.1, box-shaped Solution Spaces decouple the design variables and hence serve as independent target regions, which can be interpreted as component requirements. This enables a simultaneous and distributed, i.e. at least partially decoupled, component development and leads to an efficient design process that is also robust to variations. An important milestone for the success of Solution Spaces within the automotive design was the stochastic algorithm published in 2013 by Zimmermann & von Hössle [83] and Graff [30]. It enabled the optimisation of box-shaped Solution Spaces with respect to their size for any arbitrarily non-linear high-dimensional system. Zimmermann & von Hössle applied a stochastic approach, which optimises a solution box with respect to a size measure μ. The box is required to have at least a specified fraction of good designs. A good design is defined here as a design, which does not violate any constraints, such as performance requirements. Whether the required fraction of good designs is reached is tested by Monte-Carlo (MC) sampling. Recently, new approaches, which use classical optimisation instead of a stochastic approach to derive the optimal box for general linear problems, have been developed [21, 22]. These approaches ensure that each design inside the box is a good design.

A major challenge for Solution Spaces has been that in real-world applications the sizes of permissible intervals for crucial design variables are often not large enough to encompass all uncertainty and to ensure feasibility. In short, the optimised Solution

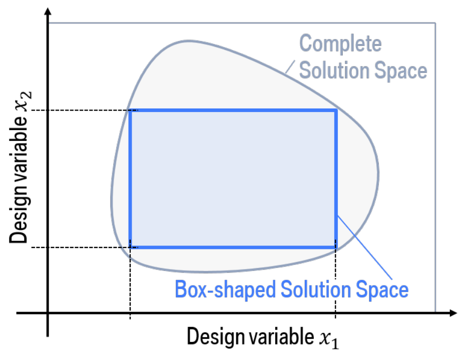


Figure 1.1 Complete solution space (grey area) and a box-shaped Solution Space (blue box) for a two-dimensional problem.

Space is simply too small [74]. In some cases, the constraints are such that not even a single design exists which fulfils all design goals.

System decomposition. Each complex system can be decomposed into subsystems and the subsystems can be decomposed into components [20]. Figure 1.2 shows the system, the subsystem as well as the component level for a passenger vehicle. Each unit on every level has different properties like mass, cost, length, stiffness, etc. In chapter 7, a typical early phase engineering problem for chassis design is introduced where the axles shall be optimised. The axle is a subsystem of the vehicle. In order to optimise the axle, properties of the subsystem components like tyre, anti-roll bar, and bump stop are modified.

V-model. The V-model considers requirements on the objective quantities first and breaks these down from the system level to the subsystem and the component level. This is visualised by the left leg of the V in figure 1.2. After developing the components with respect to their requirements each component is evaluated in order to check if it fulfils the requirements on each level. This is visualised by the right leg of the V. The V-model enables a cascading development process, which is more efficient than the classical iterative development process. In addition, the V-model does not presume an initial solution, which is particularly valuable in revolutionary design where the entire design space should be considered [82].

Target cascading. Complex systems include a huge number of connected design variables, which are often subject to uncertainty. This makes it difficult to develop the top-down requirements, which are necessary to apply the V-model. A computational approach to derive quantitative requirements for components based on system requirements is given by target cascading [38]. The classical target cascading approach computes a target point for the component requirements. This means that each component property has to assume a specific value. In the industry, this approach is usually difficult to realise since different departments are involved in designing the system. Each department will optimise their own design objectives while ignoring most other design objectives. This leads to the classical point-based design process visualised in figure 1.3 (a). The chassis

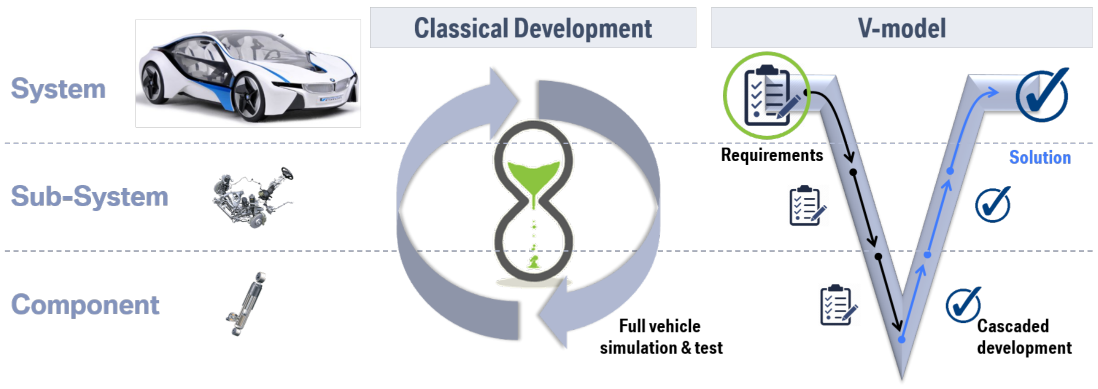


Figure 1.2 The classical development approach and the design according to the V-model [82]. Arrows show the shift of focus during the development.

design process is considered as an example here. It starts with an initial design (blue box), which is then iteratively modified to fulfil the requirements with respect to ride, driving dynamics, structural durability, and geometrical integration. Since the requirements are in conflict with each other (visualised by the different directions of the arrows), the final result will be a compromise.

For the purpose of this thesis, target regions are computed rather than a single target design. The idea is visualised in figure 1.3. Each department specifies their specific requirements. The intersection of all requirements (green box) is called the complete solution space. Each design point included is a design that fulfils all performance requirements (good design). Hence, the Solution Space approach circumvents an iterative design process and will result in a good design.

Robustness and robustness constraints. In this thesis, we consider the early design phase; therefore, lack of knowledge due to uncontrollable variations of system, subsystem, and component properties needs to be taken into account. One option to cope with this lack of knowledge is to compute sufficiently large Solution Spaces. As long as unforeseen changes to properties happen within their target regions, a redesign of the system is not required. We call this ability of a system to remain stable when unforeseen changes occur robustness. This definition is in accordance with Wieland & Wallenburg who define robustness as "the ability of a system to resist change without adapting its initial stable configuration" [78]. Note that no universally accepted definition for robustness exists. A more classical definition is given by Duddeck & Wehrle, who describe robustness as the system feature, which assures that performance loss is small with respect to design variations [15]. In this thesis, we introduce so-called robustness constraints. These require a specific minimal interval size for certain design variables. This minimal interval size is expected to be sufficiently large to compensate for any uncontrollable design variations, which might occur during the development process, and hence ensures that no iterative redesign of the entire system is necessary.

Aim of the thesis: In order to make the Solution-Space method viable for any

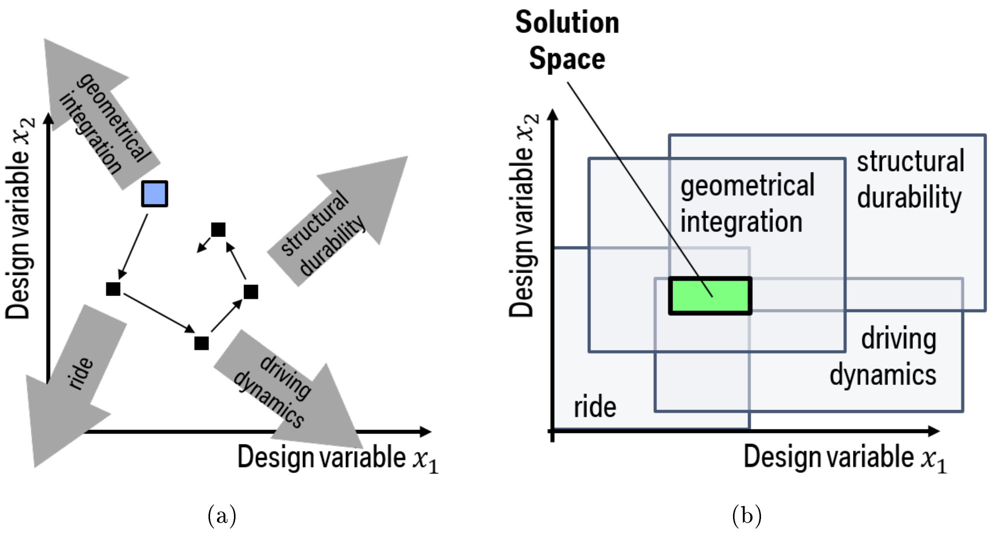


Figure 1.3 (a) Example for the classical point-based design process applied to chassis design. The blue box represents the starting point (e.g. a predecessor design) (b) Target regions from different disciplines. The green area represents the complete Solution Space [53].

real-world application with an arbitrarily rigorous set of constraints, new approaches are introduced in this thesis. These new approaches enlarge the feasible intervals for crucial design variables.

1.2. Overview of the methods

In this thesis, different approaches to compute set-based designs are either reviewed or newly introduced. Table 1.1 gives an overview of all methods considered. The methods are divided by their applicability to systems with non-linear constraints and their results. Classical box-shaped Solution Spaces are state of the art and the foundation for all of the new ideas introduced in the thesis. Therefore, they are reviewed in chapter 2.

Solution-Compensation Spaces (SCSs) were first introduced in [74] by the author of this thesis. The idea behind Solution-Compensation Spaces is to define two subsets of variables: early- and late-decision variables. The interval sizes for all early-decision variables and hence their robustness are increased compared to classical box-shaped Solution Spaces. The late-decision variables are used to compensate.

Optimal Constraint Relaxations (OCRs) is a method, which is commonly used to examine optimisation problems where not a single solution exists. A set of minimal changes to the constraints can be determined to make the problem feasible [7]. This idea is modified so that it is applicable to optimisation problems, which aim to find a set of results rather than just a single design, such as Solution Spaces and Solution-Compensation Spaces.

1.3. Structure of the thesis

In this thesis, state of the art methods are reviewed, deficiencies of state of the art methods are highlighted, new approaches to compute set-based designs are introduced, an application concerning chassis design is examined and conclusions are found. In particular the chapters include:

Chapter 1 An outline is given as to why Solution Spaces are recommended in the early phases of system design. Then, the motive behind increasing the size of permissible intervals for design variables is explained. An overview of the methods, which are considered in order to reach this goal is given.

Chapter 2 State of the art methods to determine set-based designs, prior work on chassis design with Solution Spaces as well as methods to treat infeasible problem statements are reviewed. All methods and applications introduced in this thesis are built on the methods described in the reviewed publications.

Chapter 3 The chassis design problem examined in this thesis is briefly introduced. It is shown that none of the state of the art methods is able to solve the problem described to a sufficient extent. Based on this fact, the aims of the thesis are derived. New methods, which aim to find larger feasible regions for component properties are sought.

Chapter 4 Solution-Compensation Spaces (SCSs) are introduced. Solution-Compensation Spaces are an evolution of Solution Spaces and are able to increase the ranges of the permissible intervals for crucial design variables. After detailing the idea and defining the Solution-Compensation Space problem statement as an optimisation problem, different algorithms are introduced in order to compute Solution-Compensation Spaces.

Chapter 5 Optimal Constraint Relaxation (OCR) for Solution Spaces is introduced. The goal of this method is to find a minimal set of relaxed constraints such that any Solution Space problem with arbitrarily strict constraints becomes feasible. After detailing the idea and defining the problem statement as an optimisation problem, different algorithms are introduced in order to compute Solution-Compensation Spaces.

Chapter 6 Optimal Constraint Relaxation (OCR) for Solution-Compensation Spaces is introduced. The goal of this method is to combine the advantages of the Solution-Compensation Space approach and Optimal Constraint Relaxation. The result is an optimal set of relaxed constraints while considering the increased target range based on Solution-Compensation Spaces. After detailing the idea and defining the problem statement as an optimisation problem, different algorithms are introduced in order to compute Solution-Compensation Spaces.

Chapter 7 The chassis design application, which was briefly introduced in chapter 3, is explained in detail. Then, the methods introduced in chapters 4, 5, and 6 are applied to the chassis design problem. It is shown that these methods compensate for the deficiencies of the state of the art methods and lead to satisfying solutions for the chassis design problem. The different algorithms are compared in terms of computational effort, accuracy, and performance with respect to their target function.

Chapter 8 A critical reflection on whether the research questions stated in chapter 3 have been answered is given. In the end, an outlook for the approaches introduced in this thesis is provided, key findings are summarised, the main conclusions are drawn and ideas for future research work are presented.

	linear performance constraints	**non-linear performance constraints**
Classical box-shaped Solution Spaces (reviewed in chapter 2)	Vertex Tracking Algorithm ([20], see section 2.1.3)	Stochastic Solution Space Algorithm ([83], see section 2.1.4)
Solution-Compensation Spaces (SCSs) (introduced in chapter 4)	Basic Projection Algorithm (see section 4.2.2) Fourier-Motzkin Elimination (see section 4.2.3)	Stochastic SCS Algorithm (see section 4.3.2)
Optimal Constraints Relaxation (OCR) for Solution Spaces (introduced in chapter 5)	Optimal Constraint Relaxation for systems with linear constraints (see section 5.2)	Optimal Constraint Relaxation Algorithm for systems with non-linear constraints (see section 5.3)
Optimal Constraint Relaxation for Solution-Compensation Spaces (introduced in chapter 6)	Bisection Algorithm (see section 6.2.1) Static Algorithm (see section 6.2.2) Shifting Algorithm (see section 6.2.3)	–

Table 1.1 Overview of the methods included in the thesis. The methods are categorised column-wise according to their applicability to systems with linear or non-linear performance constraints. The methods are categorised row-wise according to their result.

Chapter 2

State of the Art

As stated in chapter 1, a strong motivation to apply set-based design approaches in the early phases of complex product development exists. In this chapter, we review classical Solution Spaces, which are applied in order to determine set-based designs expressed as intervals. An overview of publications concerning chassis design with Solution Spaces is given. Finally state of the art approaches on how to treat infeasible problem statements are reviewed.

2.1. Solution Spaces

The Solution Space method is a set-based design method. A Solution Space is a set of good designs. A good design is defined here as a design, which does not violate any constraints such as performance requirements.

In the following, set-based design is introduced as a method to design complex products. A specific focus is set on box-shaped Solution Spaces, which are explained in detail. Therefore, two state of the art algorithms to compute box-shaped Solution Spaces are reviewed: Stochastic Solution Space optimisation as well as Vertex Tracking (VT). In the end, a method on how to represent high-dimensional Solution Spaces in lower dimensions is reviewed.

2.1.1. Set-based design

The modern meaning of set-based design was mainly coined by Ward, who published his first paper on it in 1989 [75]. In 1999, Sobek et al. drew significant attention in the scientific community when they published their famous article *Toyota's principles of set-based concurrent engineering* [63]. Following this article, a plethora of publications concerning set-based design have been published, e.g. [52, 56, 62, 76].

In classical set-based design approaches, an initial feasible region is derived. According to certain optimality criteria, the size of the initial feasible region is iteratively decreased until only one single optimal design is left. According to Rekuc et al., it is crucial to eliminate as many feasible designs as possible during the early phases of product development in order to decrease the number of alternatives and hence decrease the complexity of the development process [56]. The Solution Space approach follows a different idea. Instead of searching for a unique design, the size of Solution Spaces is maximised. The Solution Space approach "seeks flexibility for integration of many requirements and encompassing uncertainty" [82]. An in-depth comparison of classical set-based design and Solution Spaces can be found in [24].

2.1.2. Box-shaped Solution Spaces

Solution Spaces are areas of good designs, which can have an arbitrarily non-linear, non-convex, and non-connected shape (see figure 1.1). A good design is defined here as a design, which does not violate any constraints such as performance requirements. In this thesis, all requirements will be treated as constraints. Especially in high dimensions, the area an Solution Space covers is difficult to compute and is difficult to describe. Therefore, Zimmermann & von Hössle [83] propose box-shaped Solution Spaces. Their aim is to find large box-shaped sets of good designs. Box-shaped Solution Spaces can be expressed as a product of intervals and thus allow the decoupling of design variable requirements. This enables a simultaneous and distributed component development and leads to an efficient design process that is also robust to variations.

Definitions

The set of considered design variables is called x:

Design points or *designs* are represented by the vector

$$x = (x_1, x_2, \dots, x_d) \in \mathbb{R}^d, \tag{2.1}$$

where d is the dimensionality of the problem. For the final design manufacturability is crucial. In this thesis, we assume that the design space Ω_{ds} is always chosen such that all included designs are manufacturable. Only designs $x \in \Omega_{\mathrm{ds}}$, which are part of the design space are considered.

The response of the system at x is given by

$$z = f(x) \qquad f : \mathbb{R}^d \to \mathbb{R}^m \tag{2.2}$$

where f is the *performance function* and m is the number of performance constraints. A classical optimisation problem reads

$$\underset{x}{\mathrm{maximise}}(\varphi(x)) \qquad \varphi : \mathbb{R}^m \to \mathbb{R} \qquad x \in \Omega_{\mathrm{ds}}\ , \tag{2.3}$$

with φ being an appropriate objective function. By contrast, the Solution Space approach does not seek a single design point but a set of designs described by the lower x_i^l and the upper x_i^u bounds for each variable

$$\Omega = I_1 \times I_2 \times ... \times I_d \subset \mathbb{R}^d \tag{2.4}$$

with $I_i = [x_i^l, x_i^u]$. This box-shaped Solution Space is the solution of the following optimisation problem [83]:

Problem Statement 1.

$$\underset{\Omega \subseteq \Omega_{\mathrm{ds}}}{\mathrm{maximise}}\, \mu(\Omega) \tag{2.5a}$$

$$s.t. \quad f(x) \leq f_c, \quad \forall x \in \Omega, \tag{2.5b}$$

where $\mu(\Omega)$ is a size measure and f_c is a threshold value for the performance criteria. The solution is the box maximising $\mu(\Omega)$ while every design that is part of the box satisfies all constraints.

Deriving Solution Space constraints from performance functions

In order to apply any Solution Space method, performance constraints $f(x) \leq f_c$ need to be available. According to Erschen, three common ways to derive the performance constraints exist [20]:

- **Directly from the performance function:** A critical value g_c is given, which the output of the performance function $z = f(x)$ must not exceed $f(x) \leq g_c$.

- **From an approximation of the performance function:** In case, the performance function is unknown or too expensive to compute, a surrogate modelling method such as linear regression [26] or machine learning [47] can be applied in order to compute an approximation of the original performance function $z = \tilde{f}(x)$. Similarly to the direct method, a critical value g_c needs to be defined $\tilde{f}(x) \leq g_c$.

- **From a mathematical description of the hull of the Solution Space:** Instead of computing specific performance functions a classifier can be trained. This classifier predicts whether a certain design fulfils the considered performance constraints or not. Classical methods, which can be applied to derive such a classifier are Support Vector Machine (SVM) [31] or convex hull algorithms [6]. The disadvantage of this method is that a new classifier has to be trained whenever the critical performance value changes.

Optimising Box-shaped Solution Spaces

As explained in the introduction, we aim to maximise the size of Solution Spaces. When optimising the size of box-shaped Solution Spaces, generally two different types of optimisation approaches are distinguished:

1. **Gradient-based approaches**, which aim to optimise the solution of problem statement 1, such as Vertex Tracking. The result is a box-shaped Solution Space including 100% good designs.

2. **Stochastic approaches**, which aim to optimise the solution of a relaxed problem statement. The result is a box-shaped Solution Space, a likelihood and a fraction. The box-shaped Solution Space includes at least the specified fraction of good designs with the specified likelihood.

2.1.3. Gradient-based Solution Space Algorithms

Recently new approaches were developed, which use classical optimisation instead of a stochastic approach to derive the optimal box for general linear problems [21, 22]. These approaches ensure that each design inside the box is a good design.

Vertex Tracking, as introduced by Erschen [21], is a very versatile gradient-based Solution Space algorithm since it can be applied to any problem with linear performance constraints and to certain problems with non-linear performance constraints. Instead of checking all points inside the box, only certain critical vertices are checked. This way, the infinite number of constraints from problem statement 1 can be reduced to a finite amount. Note that the number of constraints in problem statement 1 is considered infinite. Despite the fact that Ω is a bounded space with a finite number of dimensions Ω includes an infinite number of distinct designs x and each design x must fulfil the performance constraints $f(x) \leq f_c$.

The idea of Vertex Tracking is illustrated in figure 2.1. As long as a single critical vertex (blue dot) fulfils the performance constraint, each design inside the box fulfils this performance constraint. For each constraint, such a critical vertex can be found. Vertex Tracking is explained in detail in [20].

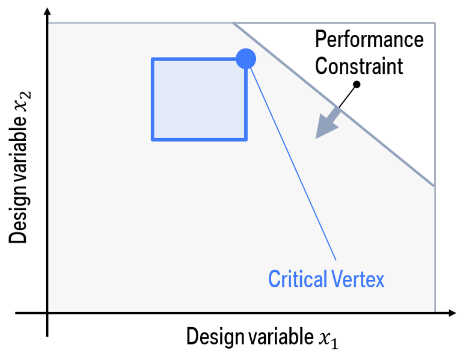


Figure 2.1 Example for Vertex Tracking. The grey area represents the complete solution space and the blue box represents a box-shaped Solution Space.

2.1.4. Stochastic Solution Space Algorithm

The stochastic Solution Space algorithm proposed by Zimmermann & von Hössle optimises a solution box with respect to the size measure μ and requires it to have at least a specified fraction of good designs, denoted by a [83]. Whether the required fraction is reached is tested by MC sampling.

The following optimisation problem based on Bayesian probabilities as described in [44] is solved:

Problem Statement 2.

$$\underset{\Omega \subseteq \Omega_{\mathrm{ds}}}{\text{maximise}}\, \mu(\Omega) \tag{2.6a}$$

$$s.t. \quad P(a_l < a | m, N) > 1 - \alpha_c, \tag{2.6b}$$

where N is the number of MC sample designs $x \in \Omega$, m is the number of good sample designs $x \in \Omega, f(x) \leq f_c$, a_l is the lower boundary of the confidence interval and $1 - \alpha_c$ is the confidence level.

2.2. Chassis design with box-shaped Solution Spaces

The Solution Space method has been applied to a variety of industrial problems: chassis design [16, 17, 18, 20, 49, 73, 74, 84], crash design [13, 22, 24, 41, 83, 84], front rail design [39] as well as chemical process design [68].

In the following, we explain why it is beneficial to apply box-shaped Solution Spaces in chassis design and why different types of algorithms are required to compute box-shaped Solution Spaces for different industrial applications.

2.2.1. Advantages of box-shaped Solution Spaces for chassis design

Zimmermann & von Hössle [83], Eichstetter [17, 18], and Erschen [20] point out that the chassis design process with box-shaped Solution Spaces has the following advantages:

- Requirements on the system level, which are provided as lower or upper thresholds w.r.t. some properties of the system, can be expressed as permissible intervals on the component level. This enables a development process in accordance with the V-model (see figure 1.2). In addition, all requirements on the component level are decoupled. As such, a cooperative design process is possible. Different groups are able to develop the details of a component simultaneously and independently.

- Computing intervals for the design variables rather than a single design provides space for uncertainties. This is of particular importance for uncertainties, which arise due to lack of knowledge since it is impossible to describe these uncertainties with a specific probability distribution. Note that it is important to maximise the size of the intervals for the design variables, since a larger interval provides more robustness and flexibility (see section 1.1).

- Intervals for design variables enable a simple visualisation and understanding of the resulting Solution Space (e.g. figures 3.1, 3.2, 3.3). This is especially important in high dimensions.

- It is possible to consider further requirements without a complete redesign of the system. This can be easily achieved by computing the common space of different Solution Spaces.

  In case new requirements are added, a new Solution Space is computed. It has the property that each design included fulfils all new requirements. Then, a common space with the original Solution Space is derived by computing the overlap of each interval. The result is a new Solution Space, which fulfils all old and new requirements.

- The centre of the box-shaped Solution Space can be determined as the most reliable design, since the distance to the edges of the box is maximised. Keep in mind that outside the box, designs might fail w.r.t. the requirements.

- It is simple to consider robustness towards an uncontrollable variable. In case an uncontrollable variable is part of the system, an interval with a specific lower and upper bound is considered. When optimising the box-shaped Solution Space, this variable is treated as a normal design variable, with the only difference being that additional constraints are set, which keep the lower and upper bound of the interval at their initial values.

- Product family design is enabled by box-shaped Solution Spaces. When the intervals of the Solution Spaces of different vehicles overlap it is an indication that communality is possible (particular components can be shared between the vehicles). By superposing the Solution Spaces of multiple vehicles, communality can be optimised such that a minimal number of different components is necessary. This leads to a faster and more cost-efficient development process [64].

2.2.2. Application of box-shaped Solution Spaces in chassis design compared to crash design

In recent years, two of the main fields where Solution Space methods have been applied are chassis design [16, 17, 18, 20, 49, 73, 74, 84] and crash design [13, 22, 24, 41, 83, 84]. From a mathematical standpoint, there is no difference between the requirements of the two areas. Each time a set of parameters $x \in \mathbb{R}^d$, a design space Ω and a set of constraints $f(x) \leq f_c$, $f : \mathbb{R}^d \to \mathbb{R}^m$ is given. Each time problem statement 1 needs to be solved to find a box-shaped Solution Space. Still, the algorithms used are often different. This is due to the following reasons:

- The number of constraints m is usually a lot higher in crash design than in chassis design.

  In chassis design, it is possible to assess the performance of a design based on characteristic values of the output curves, which are generated by the model. Such characteristic values usually represent the maximum, the minimum or the gradient at a specific location (examples are shown in table 3.2). Hence, the relevant information of an entire graph can be condensed to a couple of values or in some cases even just a single value. In real world applications, the total number of constraints m is usually below 100.

  When crash design is combined with solution spaces, the so-called direct method is often used. In this case, the output of the models are force-deformation curves [22]. Modelling forces as design parameters is a special feature of crash design. Usually, the relevant information of the force-deformation curves cannot be condensed to a single value. Instead Fender proposes to divide the curves into segments. For each segment a lower and an upper boundary is determined [24]. Depending on the refinement of the segments this leads to a number of constraints in the magnitude of $m = 1000$.

- The use of non-linear models with low computational effort is common in chassis design but not in crash design.

  In chassis design it is common to use non-linear models, which approximate the results of the full-scale finite-elements models really well and are much faster to compute. Such models are white box models, which are derived based on physical relations. The most famous example is the two-track-model, e.g. [32].

  In crash design also simplified non-linear models exist. But these models can usually only be applied to very specific crash scenarios and are hence not as flexible as their counterparts in chassis design. In [23] a method to derive simplified models to assess crashworthiness is introduced. Currently, it is only suitable for very specific load cases. Further work can be found in [10, 42, 65].

Consequently, in crash design gradient-based algorithms, which evaluate linear systems, are preferred [22, 24]. The derivation of high-dimensional Solution Spaces on full-scale finite-element models would be too time-consuming. For specific load cases stochastic approaches to derive Solution Spaces on simplified models are applied [22, 83].

Generally, in chassis design stochastic Solution Space algorithms, which evaluate non-linear systems are preferred [16, 74, 84]. The models are fast enough to allow a high number of function evaluations. This way any error, which would result from linearisation, is prevented. If very quick results are warranted, it can still be useful to apply the gradient-based Solution Space algorithms to linearised models. Note that in special cases the gradient-based approaches are also applicable to non-linear systems [20]. In these cases, gradient-based approaches are usually preferred since they are deterministic and guarantee that only good designs are included.

2.3. Infeasible problem statements

As introduced in section 2.1.2, in order to determine an Solution Space, requirements in the form $f(x) \leq f_c$, $f : \mathbb{R}^d \rightarrow \mathbb{R}^m$ need to be available. These requirements are derived on the system level and relate to costumer-relevant properties. Unfortunately, sometimes the requirements are such that it is impossible to optimise even a single design point inside the design space, which fulfils all requirements $x \in \Omega, f(x) \leq f_c$.

In the following, we introduce aspects from Requirements Engineering as a way to define, document and maintain requirements. Then, we introduce methods to treat infeasible problem statements.

2.3.1. Requirements Engineering

Definition of requirements

Different definitions for requirements can be found in the literature. A classical definition is given by Lindemann, who defines requirements as demanded functions or properties of a product. These functions and the properties are derived according to demand, boundary conditions and restrictions [45]. Zimmermann provides a more generalised definition, which is not specific to product design: Requirements are the relation of an attribute and a value that is desired. Satisfying a requirement is the condition to have reached the design goal. A requirement is typically expressed as an attribute and a target value [81]. In this thesis requirements are associated with critical performance values. Hence the more general definition according to Zimmermann is used.

Types of requirements

According to Pohl, three different types of requirements exist [54]:

- **Functional requirements** define functions to be provided by the system. In most industrial applications, the functional requirements are fulfilled because of a predefined architecture. For example, in chassis design we usually apply a very similar set of components: 4 tyres, 4 dampers, 4 bump stops, 2 axles, etc. The combination of those components will result in a chassis, which fulfils the functional requirements.
- **Quality requirements** define qualitative characteristics that the system should have. Typically, quality requirements refer to the performance, availability, reliability, scalability, or portability of the system. In chassis design, the degrees of freedom are the component properties, such as the stiffness of a damper. The design goal is to optimise these component properties such that quality requirements are fulfilled.
- **Boundary conditions** are organisational or technological specifications that restricts the way in which the system can be realised. In chassis design, the boundary conditions usually dictate the design space and hence the intervals in which the design variables can be adjusted. Note that this type of condition is called a *constraint* within the mathematical community.

Pohl's distinction of requirements mainly focuses on physical products. For software products numerical requirements should be added. These can be categorised as a subgroup of quality requirements. Typical numerical requirements are accuracy and efficiency. In addition, Pohl does not consider requirements derived from corporate strategies such as the representativeness and the relevance of a product.

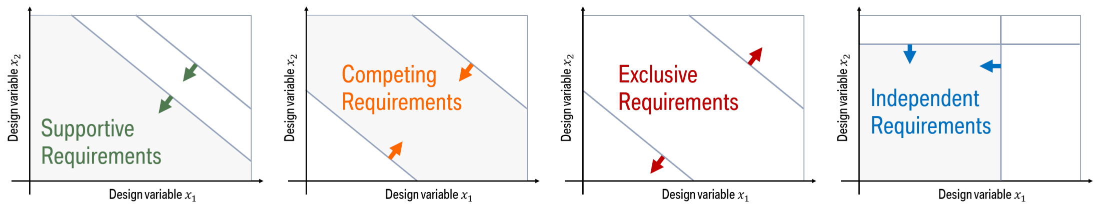
Requirement specifications

Requirement specifications change during product development: new requirements are added, requirements that have already been included change, or they even disappear. New requirements arise, for example, when a competitor launches a new product on the market that must be taken into account. It is also possible for the customer to change their ideas and thus their requirements for the product [45]. Changes of requirements create uncontrollable variations of the system and can hence be categorised as lack of knowledge uncertainties. As introduced in section 1.1, Solution Spaces are applied in this thesis to cope with lack of knowledge.

Requirement dependencies

Often different requirements have dependencies; hence, they are coupled. These dependencies do not refer to a direct relation between the requirements but indicate that reaching a specific requirement might affect the satisfaction of another requirement. As visualised in figure 2.2, three types of dependencies can be differentiated [45]:

- Requirement dependencies are classified as **supportive**, when optimising the system properties such that the first requirement is fulfilled also improves the system



(a) dependent (b) dependent (c) dependent (d) independent

Figure 2.2 2d-examples for the different types of requirements. The grey area shows the complete Solution Space. The arrows represent the normals of the constraints.

with respect to the second requirement. An example for this would be a pair of performance constraint hyper-planes where the normals are identical. A simple 2d-example is given in figure 2.2a.

- Requirement dependencies are classified as **competing**, when optimising the system properties such that the first requirement is fulfilled deteriorates the system with respect to the second requirement. An example for this would be a pair of performance constraint hyper-planes where the normals are subtended and point at each other. A simple 2d-example is given in figure 2.2b.

- Requirement dependencies are classified as **exclusive**, when it is impossible to fulfil both requirements at the same time. Note that exclusive requirements automatically lead to an infeasible problem statement. An example for this would be a pair of performance constraint hyper-planes where the normals point away from each other and the hyper-planes do not intersect within the design space. A simple 2d-example is given in figure 2.2c.

In addition, requirements can be independent:

- Requirements are classified as **independent**, when the system properties, which influence their performance are completely different. A simple 2d-example is given in figure 2.2d.

In case of competing requirements, Lindemann proposes a weighting between them in order to find an optimal design [45]. These weighting factors can be adjusted in an iterative process in order to optimise the final design. This leads to the classical point-based design shown in figure 1.3.

2.3.2. Treatment of infeasible problem statements

In case requirements are exclusive (see figure 2.2c), the problem statement is infeasible. In this case the question arises: What is the smallest set of changes, which needs to be applied to the constraints, in order to make the problem feasible?

The smallest set of changes can be computed by reformulating the feasibility problem as an optimisation problem [7]. This relates to the basic idea of any *phase 1 technique*, which means that a constrained optimisation problem is modified in order to find an initial feasible solution. A typical modification is to add artificial non-negative variables to the constraints of the optimisation problem. The target function is formulated such that the artificial variables are minimised. If all artificial variables reach a value of zero the initial constraints are fulfilled and a feasible solution is found [37]. In this thesis we execute the following steps in order to treat an infeasible problem statement:

1. Create an objective function $\mu(x)$ that measures the degree of violation of the constraints at any given point.
2. Minimize this new objective function $\min \mu(x)$, $x \in \Omega$.

Chinneck [7] highlights three types of objective functions $\mu(x)$, which can be used to measure the degree of infeasibility at any given point. They are all constructed such that $\mu(x) = 0$, if x is feasible and $\mu(x) > 0$, if x is infeasible:

- Number of Infeasibilities (NINF), i.e. the number of violated constraints $\mu(x) = N$. A constraint $f(x) \leq f_c$ is violated if $f(x) - f_c > 0$. Note that for numerical purposes in many applications constraints are counted as violated if $f(x) - f_c > \epsilon$ where ϵ is a small number like 1.0×10^{-6}.
- Sum of Infeasibilities (SINF), i.e. the weighed sum of deviations from violated constraints $\mu(x) = \sum_{i=1}^{m} q_i(f_i(x) - f_{i,c})$. Negative deviations are not considered; if a constraint is fulfilled its deviation is set to zero.
- Sum of the lengths of the feasibility vectors (SLVF), $\mu(x) = \sum_{i=1}^{m} |\zeta_i(x)|$. A feasibility vector $\zeta_i(x)$ is defined for each individual constraint as the vector extending from an infeasible point to its orthogonal projection on the constraint [8]. Hence, its length equals the shortest distance from the considered point to the constraint hyper-plane. The feasibility vector is zero for any constraint, which is fulfilled.

The result x of the newly formulated optimisation problem $\min \mu(x)$, $x \in \Omega$ can be used to derive the minimal set of changes required to make the initial problem statement feasible. The idea is that each constraint, which is violated at the location of x, is moved by $\Delta f_{c,i}$ such that $f(x) = f_c + \Delta f_{c,i}$. Depending on the chosen measure $\mu(x)$, different positions for x are optimal. This is visualised in figure 2.3. The blue cross is the result of an optimisation according to the size measure NINF. Only one constraint is violated. The orange cross is the result of the optimisation according to the size measure SLVF. The distance to each constraint is maximised in the input space. Hence, the final result lies in the centre of the triangle. The green cross is the result of the optimisation according to the size measure SINF. The distance to each constraint is maximised in the output space. The optimal result lies inside the triangle. The final position depends on the gradient of the performance functions $f(x)$. In this example, the gradient of the bottom left constraint is very steep compared to the gradient of the other constraints.

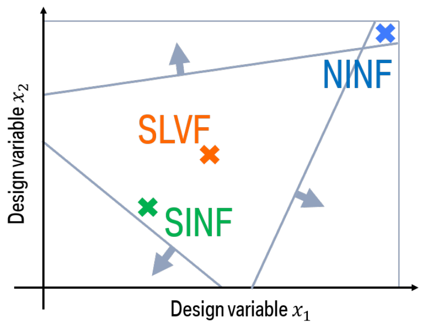


Figure 2.3 An example for optimal designs concerning different size measures: NINF, SINF, and SLVF .

In numerical applications, both NINF and SINF suffer from the row-scaling problem (=the output functions need to be weighed against each other). For NINF this might be unexpected but in numerical applications the comparison between ϵ and the deviation is necessary. In addition, NINF represents a discontinuous target function, which is usually ill-suited for numerical optimisation. Even though SLVF has the desirable property of being immune to row-scaling and has an intuitive meaning, SINF is the commonly preferred option for numerical optimisation. This is due to the fact that for non-linear constraints the feasibility vectors $\zeta_i(s)$ are very difficult to obtain [7].

Chapter 3

Research Questions

In this chapter, the aims and objectives of this thesis are defined. This is done by first clarifying the context in the field of driving dynamics and then illustrating deficiencies of existing approaches. The necessity of new improved methods, presented in the subsequent chapters, is hence illustrated.

To achieve this, an exemplary design problem is described with typical design parameters and functional requirements. It is shown that the corresponding state of the art approach for Solution Spaces fails here in identifying a set of designs, which is sufficiently robust with respect to all relevant uncertainties. Hence, methods to optimise the tolerance to these uncertainties are proposed in this thesis.

3.1. Deficiencies of the state of the art methods

An engineering problem is considered here in order to motivate the aims and objectives of this thesis. The problem considered is a typical chassis design problem in the early stage of vehicle design. State of the art design methods are applied to find appropriate solutions. It is shown that these state of the art methods are not sufficient to find satisfactory solutions concerning both robustness and performance. This motivates the new methods proposed in the subsequent chapters of the thesis.

3.1.1. Outline of the exemplary chassis design problem

For the chassis design problem, we assume that conceptional decisions concerning the structure of the vehicle have already been made. Therefore, the genes of the car such as mass, rear axle load, and the height of the centre of gravity are fixed. Now chassis components need to be designed so that customer relevant properties concerning vehicle dynamics are optimised. The chassis design problem is briefly described in this section. Details are found in section 7.1.

The input space is created by eight design variables listed in table 3.1. These variables refer to tyres, suspension, bump-stop, and anti-roll bar, which all have a significant influence on the driving dynamics behaviour of a vehicle. The design variables μ_{max}, c_{arb}, and c_{bs} are directly linked to the component properties of tyres, anti-roll bars, and bump stops. An axle including these components is shown in figure 7.1. The remaining design variable h_{Ro} (roll centre height) depends on the roll centre.

Table 3.1 Overview of the design variables

variable		unit	part	description
$\mu_{\max,X}$	$\mu_{\max,Y}$	–	tyre	Maximum friction coefficient in the tyre longitudinal/transverse axle
$h_{Ro,\mathrm{RA}}$	$h_{Ro,\mathrm{FA}}$	m	axle	Roll centre height at the rear/front axle.
$c_{\mathrm{arb,RA}}$	$c_{\mathrm{arb,FA}}$	$\frac{\mathrm{N}}{\mathrm{m}}$	anti-roll bar	Stiffness of the anti-roll bar at the rear/front axle.
$c_{\mathrm{bs,RA}}$	$c_{\mathrm{bs,FA}}$	$\frac{\mathrm{N}}{\mathrm{m}}$	bump stop	Stiffness of the bump stop at the rear/front axle.

Three standardised driving manoeuvres, which allow for an objective and reproducible assessment of the performance of the vehicle, are regarded here. Different vehicle designs are simulated and assessed with respect to customer relevant properties, which are objectified by the performance measures $f(x)$ listed in table 3.2. All design goals are met when the performance measures are between their lower and upper bound, i.e., $f(x) \leq f_c$.

Table 3.2 Vehicle performance measures and the associated requirements represented by lower and/or upper bounds.

plot colour	perf. measure	lower bound	upper bound	unit (linear)	description, manoeuvre
🟫	z_α		20.6	$\left[\mathrm{rad}/\frac{\mathrm{m}}{\mathrm{s}^2}\right]$	Self-steering gradient, QSSC
🟦	z_{a_y}	9.1		$\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right]$	Maximum lateral acceleration, QSSC
⬛	z_{F_z}	560.0		[N]	Minimum vertical tyre force at the maximum lateral acceleration, QSSC
🟥	z_{zRA}		0.009	[m]	Vertical displacement of the rear part of the car body at a specified lateral acceleration, RAST
🟥	z_{zFA}		0.009	[m]	Vertical displacement of the front part of the car body at a specified lateral acceleration, RAST
🟥	z_Φ		0.055	[–]	Roll angle of the body while cornering with a specified lateral acceleration, QSSC
🟪	z_C		2.90	[–]	Maximum amplitude of the frequency response of the vehicle body when passing a one sided road bump
🟪	z_δ	4.5		[–]	Maximum steering angle factor before loss of stability, SWD

3.1.2. Technical problem statements

The following technical problem statements typically arise in the early development stages of chassis design:

Technical problem statement 1

Derive a set of chassis component properties, such that each design included in the set is permissible in the sense that all performance requirements are fulfilled. In addition, the set shall be constructed such that it provides a defined minimal robustness concerning variations of the tyre and axle design variables.

The goal is to find a set-based design for the design variables shown in table 3.1 such that all performance constraints included in table 3.2 are met. Since we are in the early stages of product design, we need to account for all uncertainties, which are part of the design process [82]. Hence, a certain robustness concerning variations of the variables must be provided. In this thesis, robustness is considered as *the ability of a system to resist change without adapting its [...] configuration* [78]. In order to meet the additional robustness requirements, the design intervals for the maximum friction coefficient μ_{max} and the roll centre height h_{Ro} need to have a certain minimum length (see table 3.3). Changes of the variable values within these intervals can be executed during the component design phase without the need to change the design of the entire system. Two scenarios are considered in order to reflect different tyre requirements for the premium and the standard tyre. Typically, additional flexibility is demanded for the latter since more flexibility leads to larger target intervals, which leads to lower development costs. Since both the anti-roll bar and the bump stop are from a specified set of components, their design parameters can be modified in the later stages of the vehicle design process without huge significant effort. Hence, no additional constraint with respect to the robustness of c_{arb} and c_{bs} is demanded.

Table 3.3 Examples for the minimal interval sizes for axle and tyre parameters to ensure sufficient robustness of the derived design.

	$\mu_{\mathrm{max},X}$	$\mu_{\mathrm{max},Y}$	$h_{\mathrm{Ro,RA}}$	$h_{\mathrm{Ro,FA}}$
scenario 1 (premium)	0.03 [–]	0.03 [–]	0.02 [m]	0.02 [m]
scenario 2 (standard)	0.06 [–]	0.06 [–]	0.02 [m]	0.02 [m]

Technical problem statement 2

Seek a set of relaxed performance constraints and derive an associated set of chassis component properties, such that each design included in the set is permissible in the sense *that all relaxed performance requirements are fulfilled. In addition, the set shall be constructed, such that it provides a defined minimal robustness with respect to uncertainties associated with and axle design variables.*

In case no solution to technical problem statement 1 can be found, problem statement 2 is motivated: Which is the smallest set of changes we need to make to the performance constraints, such that technical problem statement 1 can be solved? Hence, a new relaxed set of performance constraints is optimised. Note that the minimal robustness requirements (see table 3.3) still need to be considered. Typically, these requirements cannot be relaxed. Especially tyre and axle designs are complex tasks with many uncertainties; therefore, it is necessary to provide the amount of robustness needed to compensate for all uncertainties.

3.1.3. Performance of state of the art methods

In the following, state of the art algorithms [4, 20, 83] are applied to the chassis design problem described in section 7.1 in order to solve the technical problem statements formulated in subsection 3.1.2.

Performances with respect to technical problem statement 1

Figures 3.1 (a)-(d) show an initial vehicle design. This design fulfils all performance requirements and hence lies in the feasible area (green). Each coloured dot in the figure is a different vehicle design. For each dimension not shown in a plot, the design variables are chosen according to their initial design value. Green dots suggest the design fulfils all performance requirements. If at least one requirement is not met, the dot is coloured differently. The colour then indicates which constraint is not met. For the colour-code consult table 3.2. If multiple constraints are violated, one of their colours is chosen to indicate that the dot represents a design that does not fulfil all performance requirements (bad design).

Approach (1). One approach to solve technical problem statement 1 is to construct an Solution Space around the initial point. This is visualised in figure 3.1. The dashed boxes represent Solution Space, which fulfil the robustness requirements defined in table 3.3. As can be seen, these boxes include bad designs. Hence, applying this approach does not solve technical problem statement 1.

Approach (2). Another approach to solve technical problem statement 1 is to optimise classical Solution Space. Therefore, both the results obtained by applying the Solution Space algorithm applying VT [20] (see figure 3.2 (a)-(d)) and the stochastic Solution Space algorithm [83] (see figure 3.2 (e)-(h)) are considered. For the stochastic algorithm a confidence interval and a confidence level need to be specified. The confidence interval describes the percentage of good designs inside the Solution Space and the confidence level describes the likelihood that the actual percentage of good designs lies within the confidence interval. The size of the resulting Solution Space is heavily influenced by the numbers chosen. For this example a confidence interval of $[0.97; 1.00]$

and a confidence level of $1-\alpha_c = 95\%$ are stipulated. This corresponds to the standard configuration where 100 sample points per Solution Space evaluation are sufficient [44]. In most industrial applications even more strict requirements are necessary since a chance of more then 3% to develop a bad design is not acceptable. Stricter requirements lead to a smaller Solution Space.
For each dimension not shown in a plot the design variables are chosen randomly from the Solution Space. Since figures 3.1 (a)-(d) show a focal design, the Solution Space has zero volume and hence the boundaries are sharp since no random sampling needs to be done in the hidden dimensions. Figures 3.2 (a)-(d) show the results of Solution Space optimisations, therefore boundaries are fuzzy. As expected, the resulting Solution Space of the Solution Space algorithm applying VT [20] is smaller than the result of a stochastic Solution Space algorithm. Therefore, the Solution Space algorithm applying VT guarantees that 100% of the Solution Space consists of good designs, while the stochastic algorithm is sampling-based and hence only gives a statistical statement on the amount of good designs in the Solution Space [83]. Comparing the results of both algorithms (black box) to the required box size (dashed black boxes), it can be seen that the sizes of the permissible intervals are still too small. The only exception is $\mu_{\max,X}$ where the intervals derived are large enough to encompass uncertainty to a sufficient extent for the premium tyre (see figure 3.2 (e)).

Approach (3). In order to further enlarge the interval size for the maximum friction coefficient $\mu_{\max}$ as well as the roll centre height h_{Ro}, the anti-roll bars c_{arb} and the stiffness of the bump stop c_{bs} are set to a single design value. Particle Swarm Optimisation (PSO) is used in order to determine a design x_{opt}, which is optimal with respect to the following target function:

$$\underset{x}{\text{minimise}} \left\{ \max_{j} \frac{f_j(x) - f_j^{\text{crit}}}{f_j^{\text{crit}}} \right\}. \tag{3.1}$$

Where x are the design variables from table 3.1 and f_j^{crit} is the upper/ lower boundary for the j-th performance constraint taken from table 3.2. The result of equation (3.1) is a good design and maximises the distance to the closest output constraint in the output space. After fixing the values for c_{arb} and c_{bs}, a classical Solution Space optimisation is run. The results for both VT and stochastic Solution Space optimisation are shown in figures 3.2 (i)-(p). Even though the volume of the Solution Space further increases compared to what happens for Approach 2, with the exception of $\mu_{\max,X}$ for premium tyres, the individual interval sizes are still not large enough to fulfil the robustness requirements.

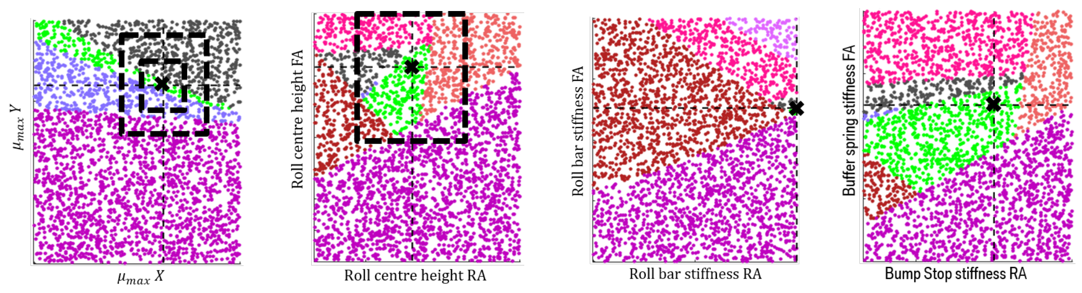
An overview of the interval sizes as well as the resulting volume of the Solution Space for $\mu_{\max}$ and h_{Ro} for Approaches 2 and 3 is given in table 3.4. The interval sizes are normalised with respect to the requirements of scenario 1 ($\mu_{\max}[1:0.03]$; $h_{Ro}[1:0.02\text{m}]$).

Performances with respect to technical problem statement 2

Unfortunately, for the technical problem statement 1 none of the applied state of the art methods generated an Solution Space, which satisfies the robustness requirements from

Table 3.4 Approach (2) & (3). Normalised interval sizes (normalised with respect to requirements of scenario 1) and the resulting volume of the Solution Space for $\mu_{\max}$ and h_{Ro}.

	$\mu_{\max,X}$	$\mu_{\max,Y}$	$h_{\mathrm{Ro,RA}}$	$h_{\mathrm{Ro,FA}}$	volume
scenario 1 (premium)	1.00	1.00	1.00	1.00	1.00
scenario 2 (standard)	2.00	2.00	1.00	1.00	4.00
Approach 2 (Vertex Tracking)	1.05	0.51	0.24	0.31	0.04
Approach 2 (stochastic)	1.71	0.76	0.33	0.84	0.36
Approach 3 (Vertex Tracking)	1.73	0.77	0.75	0.65	0.65
Approach 3 (stochastic)	1.61	0.92	0.63	0.94	0.88



(a) intial design (b) initial design (c) initial design (d) initial design

Figure 3.1 Approach (1). Nominal design (black cross) and cross-sections of the complete Solution Space (feasible designs are marked as green area) and two Solution Spaces, which fulfil the robustness requirements with respect to scenario 1 and 2 (dashed boxes, identical for the dimensions shown in (b)-(d)).

Table 3.5 Approach (4). Relative amount of relaxation with respect to the performance constraints.

	z_α	z_{a_y}	z_{F_z}	$z_{z_{RA}}$	$z_{z_{FA}}$	z_Φ	z_C	z_δ
scenario 1	0	0	0	+2.22%	0	+1.30%	0	-1.33%
scenario 2	0	-3.94%	-1.81%	+2.22%	0	+3.67%	0	-2.44%

table 3.3. Hence, the question arises: which is the smallest set of changes required, such that technical problem statement 1 becomes feasible? For problem statements where only a single good design is sought this question can be answered by applying classical methods for the treatment of infeasible problem statements. For high-dimensional boxes with non-linear constraint functions these are not applicable.

Approach (4). A basic approach to determine a set of relaxed constraints considers all vertices of the desired Solution Space and relaxes each constraint such that each vertex is feasible. Since Approach 3 with the stochastic Solution Space optimisation yielded the best results so far, the surrounding dashed boxes are used as the desired Solution Space and the values for c_{arb} and c_{bs} are fixed at the appropriate value for this approach. The problem is 4-dimensional, hence the Solution Space has $2^4 = 16$ corners, which need to be considered. For problems where all constraints are either linear or monotonically increasing/decreasing Vertex Tracking can be applied to reduce the number of corners which need to be considered. With Vertex Tracking only m corners need to be considered, where $m = 8$ is the total number of constraints [20]. Figure 3.3 shows the results of the relaxed problem statement and table 3.5 shows by how much each individual constraint has been relaxed. Even though the results might look satisfying, this method has three severe drawbacks:

- The number of corners, which have to be checked, increases exponentially 2^d with the number of dimensions, which leads to an prohibitively high computational effort in high dimensions.
- For non-linear performance constraints, it is not sufficient to only consider the vertices since inner points of the Solution Space might be bad designs even if all vertices are good designs.
- The constraint relaxation is most probably not minimal since the position of the Solution Space that is chosen initially is not optimised with respect to minimal constraint relaxation.

3.2. Aims and Objectives of this thesis

Large Solution Space are sought in the early phases of product development since they are able to compensate for the uncertainties, which arise due to a lack of knowledge. In addition, large solution spaces yield flexibility for decision makers as well as robustness to epistemic uncertainty. Hence, large solution spaces are able to prevent time consuming and expensive iterations in the development process. In order to fulfil the requirements with respect to the size of particular intervals of an Solution Space, so-called robustness constraints are introduced in this thesis. They define a minimal interval size for each affected design variable.

As shown in section 3.1, current state of the art methods are not always able to compute Solution Spaces of sufficient size to fulfil the robustness constraints of industrial problems. The *first aim* of this thesis is to develop new methods, which are able to significantly increase the size of permissible intervals for crucial design variables. For this, it is important to maintain the decoupling nature of Solution Space in order to enable distributed development.

This is achieved by introducing Solution-Compensation Spaces. The idea is that different kinds of design variables can often be distinguished in a development process: *early-* and *late-decision variables*. By delaying the specification of values for late-decision variables to the later stages of the development process, increased flexibility and robustness can be achieved with respect to early-decision variables.

Even though Solution-Compensation Spaces are able to increase the size of permissible intervals in many industrial applications, the size of these intervals might still be too small to compensate for all uncertainties. Therefore, the *second aim* of this thesis is to develop a method, which computes a minimal set of changes applied to the performance constraints such that the problem statement becomes feasible with respect to both, the relaxed performance constraints and the initial robustness constraints. This is achieved by introducing Optimal Constraint Relaxation for Solution Spaces.

The *third aim* of this thesis is to develop an algorithm, which is able to combine the advantages of Solution-Compensation Spaces with the advantages of Optimal Constraint Relaxation for Solution Spaces. Thus making any problem statement where an Solution Space, with early- and late-decision variables as well as robustness requirements, is sought feasible while relaxing the initial performance constraints as little as possible. This is achieved by introducing Optimal Constraint Relaxation for Solution-Compensation Spaces. The idea is that Solution-Compensation Spaces are computed after relaxing the performance constraints of the initial problem statement.

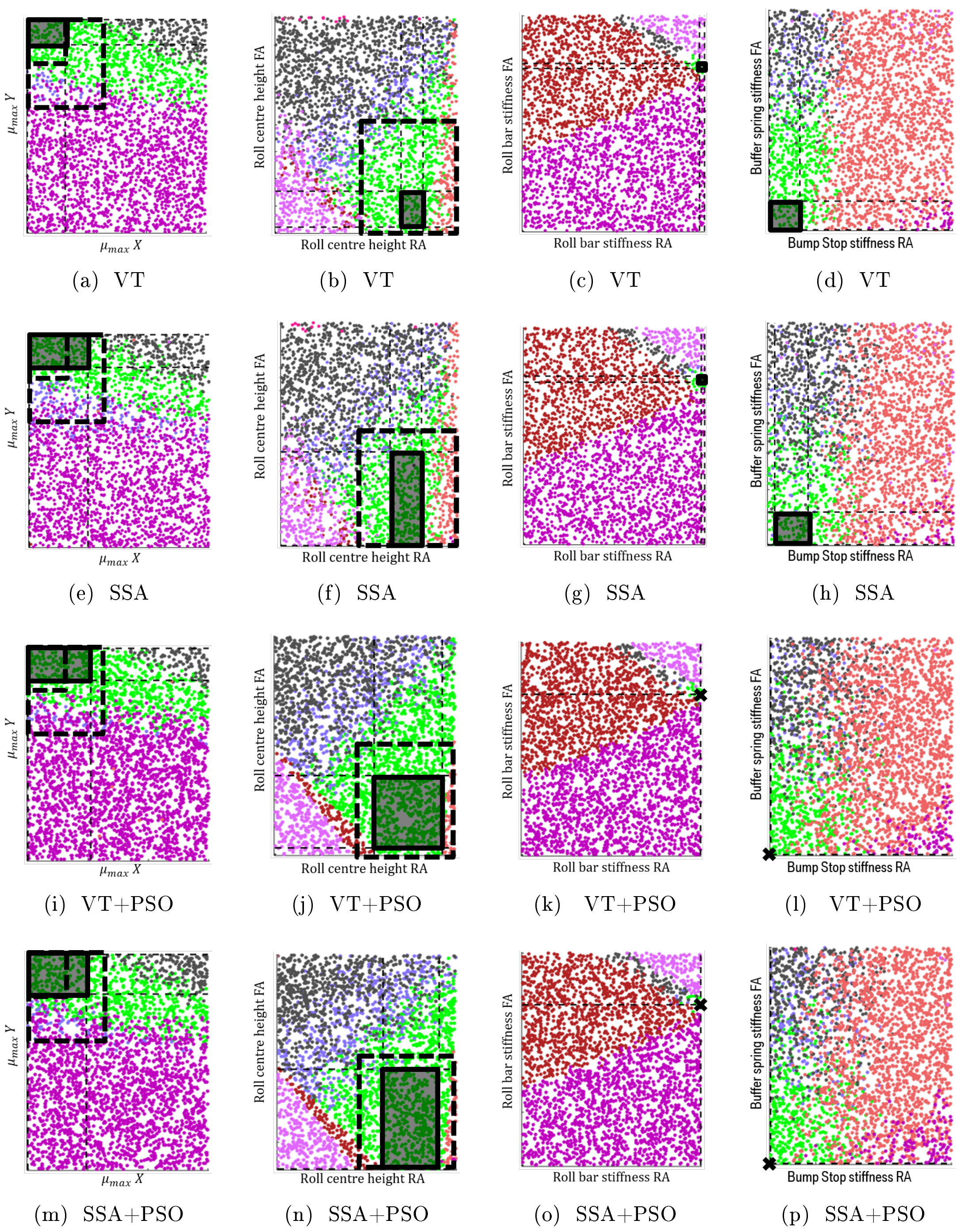
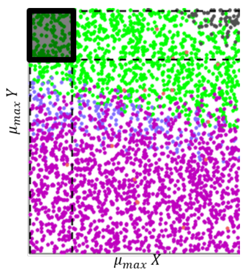
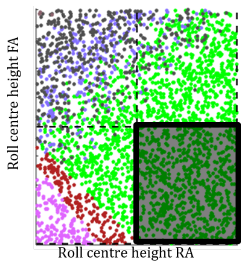


Figure 3.2 (a)-(h) **Approach (2).** Classical box-shaped Solution Space (black box) optimised either with Vertex Tracking (VT) or the Stochastic Solution Space Algorithm (SSA) and an Solution Space, which fulfils the robustness requirements (dashed box).
(i)-(p) **Approach (3).** Combination of Particle Swarm Optimisation (PSO) and a classical box-shaped Solution Space (black box) optimised either with Vertex Tracking or the SSA.

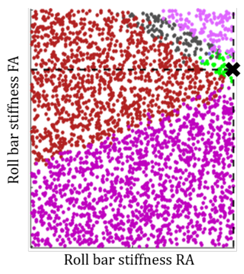
Note that for each method developed in this thesis an algorithm is given that is applicable to full-scale vehicle dynamics problems, which can have well over 100 design variables.



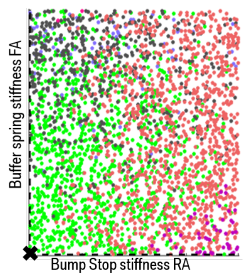
(a) basic relaxed



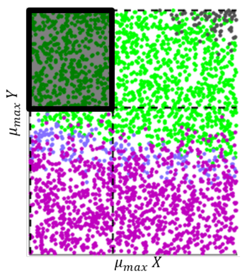
(b) basic relaxed



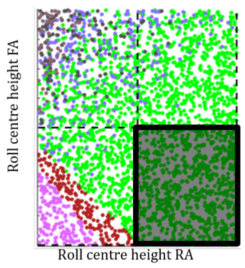
(c) basic relaxed



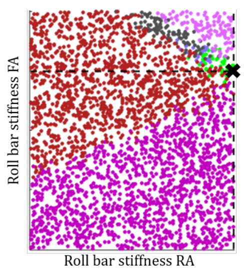
(d) basic relaxed



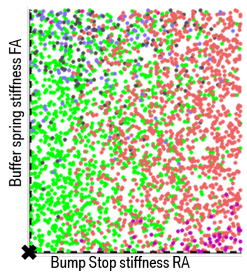
(e) basic relaxed



(f) basic relaxed



(g) basic relaxed



(h) basic relaxed

Figure 3.3 Approach (4). (a)-(d)/(e)-(h) An Solution Space (black box) that fulfils the robustness requirements for scenario 1/2 (see table 3.3) while fulfilling the relaxed performance constraints for scenario 1/2 (see table 3.5).

Chapter 4

Solution-Compensation Spaces

As the classical Solution Space approach, the Solution-Compensation Spaces represent a high-dimensional Solution Space by intervals for each design variable. New is that the design variables are now divided into a set of early- and a set of late-decision variables. Early-decision variables are associated with permissible intervals on which they may assume any value. Late-decision variables are associated with intervals where they can be adjusted to any specific value. Solution-Compensation Spaces have the property that for all values of early-decision variables from their permissible intervals there exists at least one set of late-decision variable values from their intervals such that the resulting design reaches all design goals. In order to enclose as many good designs as possible, the size of the permissible intervals for the early-decision variables is maximised.

In this chapter, mathematical formulations of the underlying optimisation problem are stated and two different approaches to compute Solution-Compensation Spaces for systems with linear constraints are proposed. In order to compute Solution-Compensation Spaces for systems with non-linear constraints a stochastic approach, based on the classical Solution Space algorithm [83], is presented at the end of the chapter.

4.1. Idea, overview, and problem statement

As introduced in section 2.1.2, box-shaped Solution Spaces are maximised with respect to a size measure (e.g. the volume of the box) in order to maximise the size of the permissible intervals. Unfortunately, in many industrial applications, the size of the permissible intervals for crucial design variables derived by the Solution Space approach is not sufficiently large to compensate for all uncertainties during the development process.

As already published by the author of this thesis in [74], Solution-Compensation Spaces are proposed, which allow to compute larger intervals by introducing the distinction between early- and late-decision variables as follows:

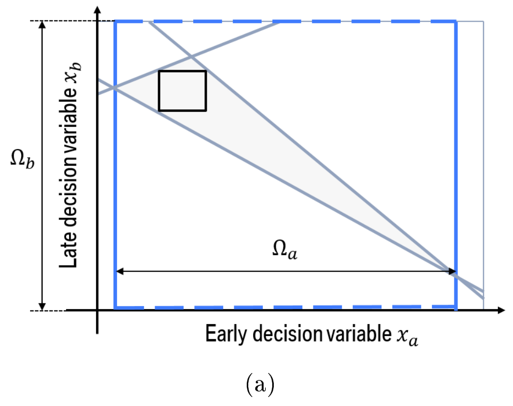
- *Early-decision variables* underlie large uncertainty and have to be bounded during the early phases of the development process since they have a strong influence on the performance of the system (e.g. the design of a suspension). When computing Solution-Compensation Spaces, these variables are associated with permissible intervals on which they *may* assume any value. It is attractive to identify the largest intervals possible for these variables.
- *Late-decision variables* are specified in the late stages of the development process since they are associated with parts, which are easy to adjust (e.g. the tuning variables of a control system or the design of an anti-roll bar). When computing Solution-Compensation Spaces, these variables are associated with intervals on which they *have to* be able to assume any value. Here, it is not necessary to have the largest possible intervals.

The Cartesian product of the early-decision variable intervals and the late-decision variable intervals is called an Solution-Compensation Space. Solution-Compensation Spaces serve to increase the size of the permissible intervals for early-decision variables while requiring additional conditions for late-decision variables. This means that an additional dependency is allowed compared to the classical Solution Spaces.

4.1.1. Basic idea of Solution-Compensation Spaces

In order to illustrate the basic idea of computing Solution-Compensation Spaces, a two-dimensional example problem is shown in figure 4.1a. The three straight lines forming the grey triangle represent the constraints. The triangle depicts the area of all good designs where all requirements are satisfied, called the complete Solution Space. The complete design space Ω_{ds} is represented by the grey surrounding line. Inside of the complete Solution Space, the box-shaped Solution Space is shown. Comparing the box-shaped Solution Space with the complete Solution Space shows that the box only covers a small part of the complete Solution Space. Therefore, restricting development to the Solution Space represents a significant loss of good designs.

It is assumed here that the Solution Space interval for the early-decision variable x_a is too small and should be enlarged. This is made possible by changing the character of x_b : rather than treating it as an uncertain variable that may assume any value within some interval, it is considered as a variable that can be adjusted arbitrarily well to any desired value from an assigned interval. Its final value will be determined in a later stage, in particular after x_a is chosen. One possibility to derive the early-decision variable interval for x_a is to project the complete Solution Space onto the x_a axis. The late-decision variable interval for x_b is chosen to be its entire design interval. Note that it is possible to determine a minimal interval for each late-decision variable x_b within Ω_b, which is sufficient to provide the same feasible regions for the early-decision variables x_a as the entire Ω_b. In industrial applications Ω_b is chosen such that any design x_b within is viable. Hence, more precise intervals for x_b yield no benefit. In the 2d example (see figure 4.1a), the Solution-Compensation Space is represented by the blue line. As can be seen, for all values in the x_a interval, there exists at least one value in $x_b \in \Omega_b$ such that the resulting design reaches all design goals. Figure 4.1b shows an example of the resulting late-decision variable interval Ω_b^* for a chosen early-decision variable value $x_a^* \in \Omega_a$.



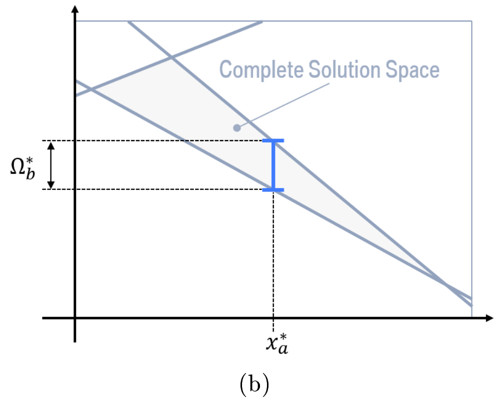


Figure 4.1 (a) The box-shaped Solution Space (black) and the SCS (blue). Dashed/solid lines indicate that the respective axis is associated with a late/early-decision variable. (b) A realised value for the early-decision variable x_a^* and the resulting late-decision variable permissible interval Ω_b^* (blue solid line).

4.1.2. Design Process with Solution-Compensation Spaces

A typical iterative design process consists of a component design phase in which the design is chosen and iteratively improved (see figure 4.2). The classical Solution Space approach extends the component design phase by a system design phase in which permissible intervals for all variables are derived first (see figure 4.3). This enables the development of a single design, which satisfies all design goals without any iterative steps. As depicted in figure 4.4, the Solution-Compensation Space approach adds a third design step called the compensation phase. In the system design phase, a set of early- and late-decision variables needs to be determined. Then, the enlarged permissible intervals for the early-design variables x_a are derived. Note that larger intervals for x_a grant more flexibility during the development process. This accounts for the lack-of-knowledge situation in early development stages. In the component design phase, all early-decision variables x_a are chosen. In the compensation phase, all late-decision variable intervals are computed in dependency on the decisions made for the early-decision variables; their values x_b are chosen such that in combination with the early-decision variables a good design is generated.

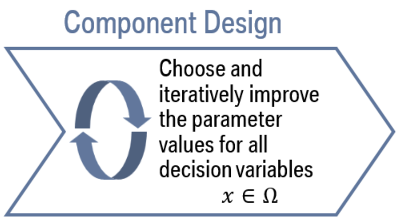

Component Design
Choose and iteratively improve the parameter values for all decision variables $x \in \Omega$

Figure 4.2 Iterative design process according to [43].

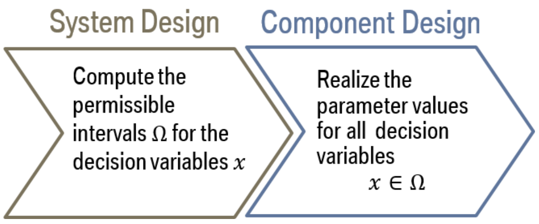

System Design
Compute the permissible intervals Ω for the decision variables x
Component Design
Realize the parameter values for all decision variables $x \in \Omega$

Figure 4.3 Approach using system and component design of classical Solution Spaces according to [83].

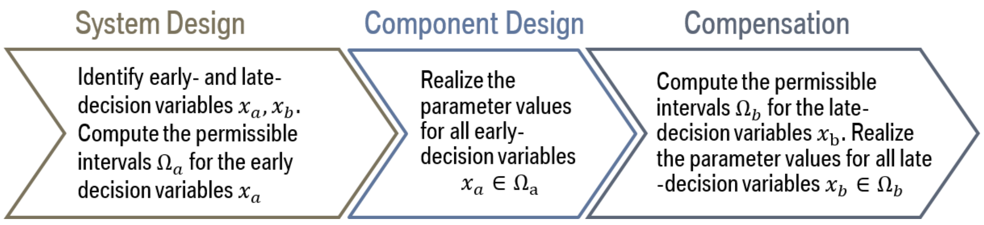

System Design
Identify early- and late-decision variables x_a, x_b. Compute the permissible intervals Ω_a for the early decision variables x_a
Component Design
Realize the parameter values for all early-decision variables $x_a \in \Omega_a$
Compensation
Compute the permissible intervals Ω_b for the late-decision variables x_b. Realize the parameter values for all late -decision variables $x_b \in \Omega_b$

Figure 4.4 Extension to a three-step design approach for sequential development using SCSs.

4.1.3. Problem statement

Concerning Solution-Compensation Spaces, design points are represented by the vector

$$x = (x_a, x_b) \tag{4.1}$$

with $x_a = (x_{a,1}, x_{a,2}, ..., x_{a,p})$ and $x_b = (x_{b,1}, x_{b,2}, ..., x_{b,q})$ where p and q are the total numbers of early- and late-decision variables, respectively. The index a indicates an early-decision variable whereas the index b indicates a late-decision variable.

Solution-Compensation Spaces are computed similarly to classical Solution Spaces (see Section 2.1.2), i.e. a box-shaped set of design points is derived, described by the lower and upper bounds; the difference is that only early-decision variables x_a and its associated Solution Space are optimisation variables. A Solution-Compensation Space is the solution of the following optimisation problem

Problem Statement 3.

$$\underset{\Omega_a \subseteq \Omega_{\mathrm{ds},a}}{\text{maximise}}\, \mu(\Omega_a) \tag{4.2a}$$

$$s.t. \quad \forall x_a \in \Omega_a,\ \exists x_b \in \Omega_{\mathrm{ds},b},\ f(x_a, x_b) \le f_c\,. \tag{4.2b}$$

Note that only $\Omega_a = I_{a,1} \times I_{a,2} \times ... \times I_{a,p}$ defines the degrees of freedom (dof) whereas $\Omega_b = \Omega_{\mathrm{ds},b}$ is fixed. For every design that is part of Ω_a there exists at least one set of values for the late-decision variables x_b such that all constraints are satisfied.

4.2. Linear Performance Constraints

In this section, the problem statement with linear constraints is presented. Afterwards the Basic Projection Algorithm as well as the Fourier-Motzkin Elimination with redundancy removal are introduced to solve the problem statement.

4.2.1. Problem statement with linear performance constraints

This thesis proposes two algorithms (see Sections 4.2.2 and 4.2.3) for systems with linear constraints

$$f(x) = Fx + c = Ax_a + Bx_b + c, \qquad F \in \mathbb{R}^{m\times(p+q)} \quad A \in \mathbb{R}^{m\times p} \quad B \in \mathbb{R}^{m\times q} \quad c \in \mathbb{R}^m \tag{4.3}$$

where $p \in \mathbb{N}$ is the number of early-decision variables and $q \in \mathbb{N}$ is the number of late-decision variables. $m \in \mathbb{N}$ represents the number of performance constraints.

Similarly to expression (4.2), box-shaped Solution-Compensation Spaces for linear system responses for the early-decision variables x_a are sought. In order to derive these intervals, the following optimisation problem is solved:

Problem Statement 4.

$$\underset{\Omega_a \subseteq \Omega_{\mathrm{ds},a}}{\text{maximise}}\ \mu(\Omega_a) \tag{4.4a}$$

$$s.t. \quad \forall x_a \in \Omega_a,\ \exists x_b \in \Omega_{\mathrm{ds},b},\ Ax_a + Bx_b \le f_c. \tag{4.4b}$$

Note that the constant $c \in \mathbb{R}^m$ is included in f_c. Considering the design space boundaries as linear constraints, the linear problem statement (4.2b) can be rewritten as

Problem Statement 5.

$$\underset{\Omega_a}{\text{maximise}}\ \mu(\Omega_a) \tag{4.5a}$$

$$s.t. \quad \forall x_a \in \Omega_a,\ \exists x_b,\ Gx \le g_c, \tag{4.5b}$$

$$G = \begin{bmatrix} A \quad B \\ -I \\ I \end{bmatrix} \in \mathbb{R}^{(2p+2q+m)\times(p+q)} \qquad g_c = \begin{bmatrix} f_c \\ -x^l_{\mathrm{ds}} \\ x^u_{\mathrm{ds}} \end{bmatrix} \in \mathbb{R}^{(2p+2q+m)}, \tag{4.5c}$$

where the boundaries of the entire design space $\Omega_{\text{ds}} = \Omega_{\text{ds},a} \times \Omega_{\text{ds},b}$ are represented by $x^l_{\text{ds}} = \{x^l_{\text{ds},i}\}$ and $x^u = \{x^u_{\text{ds},i}\}$ with $i = 1, ..., p+q$. I represents the identity matrix.

In order to apply the following algorithms (in sections 4.2.2, 4.2.3), which solve problem statement 4/5, the Solution Space described by $x|Gx \leq g_c, x \in \mathbb{R}^{p+q}$ is assumed to be fully dimensional and no duplicate constraints exist. This can be checked by applying appropriate preprocessing:

- Search for an interior point. If an interior point does exist, full dimensionality of the system can be assumed [69].
- Delete all constraints, which are multiples of other constraints.

4.2.2. Basic Projection Algorithm

In this section, the idea, the notation, and the computational effort of the basic projection algorithm are described.

Idea

The idea of the *Basic Projection Algorithm* is to modify the constraints (4.2b) of the initial problem statement such that the late-decision variables x_b are eliminated from the expression. Solving the following problem:

Problem Statement 6.

$$\text{Given} \quad A \in \mathbb{R}^{m \times p}, \quad B \in \mathbb{R}^{m \times q}, \quad f_c \in \mathbb{R}^m, \quad \Omega_{\text{ds},a} \in \mathbb{R}^p, \quad \Omega_{\text{ds},b} \in \mathbb{R}^q \tag{4.6a}$$

$$\text{Find} \quad \tilde{A} \in \mathbb{R}^{\tilde{m} \times q}, \quad \tilde{f}_c \in \mathbb{R}^{\tilde{m}} \tag{4.6b}$$

$$\text{s.t.} \quad \forall \{x_a \in \Omega_{\text{ds},a} | \exists x_b \in \Omega_{\text{ds},b} : Ax_a + Bx_b \leq f_c\} \Rightarrow x_a : \tilde{A}x_a \leq \tilde{f}_c \tag{4.6c}$$

$$\wedge \quad \forall \{x_a \in \Omega_{\text{ds},a} | \nexists x_b \in \Omega_{\text{ds},b} : Ax_a + Bx_b \leq f_c\} \Rightarrow x_a : \tilde{A}x_a > \tilde{f}_c. \tag{4.6d}$$

Late-decision variables x_b are eliminated by projecting the complete Solution Space into the design space of the early-decision variables $\Omega_{\text{ds},a}$. This is accomplished in four steps:

Details

In the following, the four steps of algorithm 1 are explained in detail.

Step (1). In the first step the intersections of all constraint hyper-planes Λ_c are computed. This is visualised in figure 4.5 (a). Therefore, the following procedure is applied: Construct all possible sub-matrices G_k of G as defined in equation (4.5) by removing rows such that

```
Algorithm 1: Basic projection algorithm
Data: Initial System $Ax_a + Bx_b \leq f_c$ with $x_a \in \Omega_{\mathrm{ds},a}$ and $x_b \in \Omega_{\mathrm{ds},b}$
Result: Projected System $\tilde{A}x_a \leq \tilde{f}_c$ with $x_a \in \Omega_{\mathrm{ds},a}$
(1) Seek the intersections of all constraint hyper-planes.
(2) Determine the set of intersections, which satisfy all constraints and are part of
    the design space.
(3) Project the intersection points onto $\Omega_{\mathrm{ds},a}$.
(4) Determine the convex hull of the projected intersection points.
```

- $\mathrm{rank}(G_k) = p + q$ and
- $G_k \in \mathbb{R}^{(p+q)\times(p+q)}$.

Then, construct $g_{c,k}$ for each G_k by removing the same rows from g_c and solve the linear equations

$$G_k x_k = g_{c,k} \, . \tag{4.7}$$

The set of all basic solutions found is denoted as Λ_c. Note that these solutions are not necessarily feasible.

Step (2). The set of all feasible vertices Λ is denoted as

$$\Lambda = \{x \in \Lambda_c | \, Gx \leq g_c\} \, . \tag{4.8}$$

Step (3). As visualised in figure 4.5 (b), in the third step all feasible vertices Λ are projected. Therefore, a simple projection operator is used:

$$p : \mathbb{R}^{p+q} \rightarrow \mathbb{R}^p \qquad p(x) = \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_p \\ x_{p+1} \\ \vdots \\ x_{p+q} \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix} \tag{4.9}$$

$$\Lambda_p = \{p(x) | x \in \Lambda\} \, . \tag{4.10}$$

Step (4). In the last step, the convex hull of Λ_p is determined. Note that this step is necessary in order to derive the feasible region for the early-decision parameters.

$$\Omega_{c,a} = \mathrm{conv}\{\Lambda_p\} = \{x_a | \, \tilde{G}x_a \leq \tilde{g}_c\} = \{x_a | \, \tilde{A}x_a \leq \tilde{f}_c, x_a \in \Omega_{\mathrm{ds},a}\}. \tag{4.11}$$

In order to compute the convex hull, the quick hull algorithm developed by [2] is used.

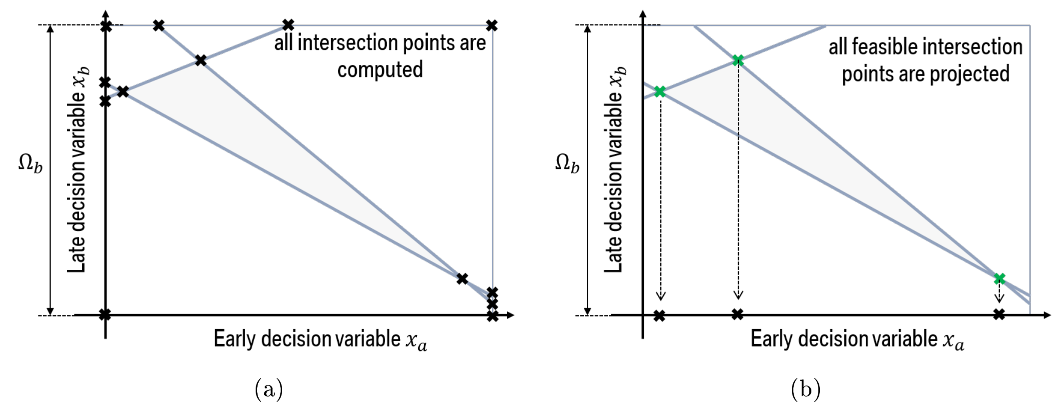

all intersection points are computed
Late decision variable x_b
Ω_b
Early decision variable x_a
(a)
all feasible intersection points are projected
Late decision variable x_b
Ω_b
Early decision variable x_a
(b)

Figure 4.5 (a) Step (1) of the Basic Projection Algorithm: intersection points of all constraints hyper-planes (black crosses) (b) In Step (2) and (3): projection of all feasible intersection points (green crosses) .

The resulting polytope is called the *early-decision-variable space after projection.*

After the late-decision variables x_b are eliminated from the problem statement it can be written as a classical Solution Space problem

$$\underset{\Omega_a \subseteq \Omega_{\mathrm{ds},a}}{\text{maximise}}\, \mu(\Omega_a) \tag{4.12a}$$

$$s.t. \quad \forall x_a \in \Omega_a,\ \tilde{A}x_a \leq \tilde{f}_c. \tag{4.12b}$$

At this point, the stochastic Solution Space algorithm [83] as well as the VT approach [21] can be used to find the box-shaped Solution Space.

Computational Effort

The basic projection algorithm produces a result for any linear problem within a high-dimensional space. It is divided into four steps:

Step (1). The upper bound of the binomial coefficient $\begin{pmatrix} n \\ k \end{pmatrix}$ can be estimated with $\left(\frac{ne}{k}\right)^k$ where $n = 2p + 2q + m$ and $k = p + q$ [12]. This means exponential growth for the computational effort.

Step (2). The computational complexity to determine, which of the intersections fulfil all constraints, is $O(s)$ where s is the number of intersections.

Step (3). The computational complexity to project all good design points is $O(s_g)$ where s_g is the number of good intersection points.

Step (4). According to [2], the computational complexity of the quick hull algorithm for a space with dimensionality higher than three is $O(nf_r/r)$. Where n is the number of input points, r is the number of processed points, and f_r is the maximum number of facets created by r vertices.

The computationally most expensive part is step 1, which grows exponentially with the number of dimensions. Its computational effort can be reduced by predicting a subset of all possible matrices G_k that do not have full rank. Whenever a requirement has both, an upper and a lower bound (e.g. all design space boundaries), this leads to two parallel constraint hyper-planes, which cannot intersect. Thus, any matrix G_k that includes an upper and a lower boundary of the same constraint does not have full rank and can immediately be dismissed. If all matrices G_k that include the upper and the lower bound of the same requirement are dismissed a reduced total of

$$\sum_{i=0}^{p+q} \binom{s}{p+q-i} \binom{r}{i} 2^i \tag{4.13}$$

linear equalities has to be solved in step 1 of the basic projection algorithm. Where r is the number of constraints that have both an upper and a lower bound and s is the number of constraints with a single bound. Each matrix G_k consists of a total of $p+q$ rows from the initial matrix G. In every part of the sum, a set of $p+q-i$ single bounded constraints and i double bounded constraints are chosen. For each of the i double bounded constraints, the upper and the lower bound have to be accounted for. Thus, multiplying each part of the sum with 2^i. Unfortunately, even with this performance improvement the basic projection algorithm is not suitable for very high-dimensional problems since the effort still grows exponentially with the number of dimensions.

Note: It is possible to reduce the number of linear equalities even further by checking if each combination of constraints is linearly dependent. But this does not lead to an improved performance, since checking for linear dependency is equivalent for solving the linear system $Ax = 0$ and determining whether it has non-trivial solutions.

4.2.3. Fourier-Motzkin Elimination with redundancy removal

In this section, the Fourier-Motzkin Elimination (FME) [11] with Redundancy Removal (RR) is applied in order to compute Solution-Compensation Spaces with linear constraints (see problem statement 4). This idea was first published by the author in [73]. It is shown that the algorithm scales very well with the number of dimensions d and thus can be applied to high-dimensional problems.

Idea

The FME is an algorithm, which eliminates variables from a system of linear inequalities. It was first introduced by Fourier in order to test polyhedrons for emptiness. This is done by eliminating all but one dimension from the constraints. If the one-dimensional inequality has a solution it can be deduced that the polyhedron is not empty. Here FME is used in a similar way, but instead of eliminating all but one dimension only the late-decision variables x_b are eliminated from the constraints $Ax_a + Bx_b \leq f_c$ of equation (4.4). This yields a new system $\tilde{A}x_a \leq \tilde{f}_c$ with the property that for each x_a that is a solution of the new system there exists an x_b such that (x_a, x_b) is a solution of $Ax_a + Bx_b \leq f_c$ [33]. This is equivalent to the constraint transformation carried out in the basic projection algorithm (see subsection 4.2.2).

The FME algorithm eliminates each variable step by step. Unfortunately, in the worst case, the number of inequalities grows double exponentially with the number of eliminated variables (see equation (4.22)). This can be circumvented by complete removal of all redundancies between the elimination of each subsequent late-decision variable [1]. Efficient ways to remove redundancies are proposed by Clarkson [9] and Fukuda [28]. Even though Fukuda's combinatorial redundancy detection has a slightly faster runtime for some applications, the Clarkson Algorithm is chosen here for this application since it is not restricted to non-negative variables. The following algorithm is executed:

Algorithm 2: FME algorithm with RR

```
Data: Initial System $Ax_a + Bx_b \leq f_c$ with $x_a \in \Omega_{\text{ds},a}$ and $x_b \in \Omega_{\text{ds},b}$
Result: Projected System $\tilde{A}x_a \leq \tilde{f}_c$ with $x_a \in \Omega_{\text{ds},a}$
for each dimension $i$ of $\Omega_{ds,b}$ do
    (1) Project the $i$-th coordinate in the direction $-e_i$ using the FME. $e_i$ is the
        standard basis vector for the $i$-th coordinate;
    (2) Eliminate all redundant constraints, using the Clarkson Algorithm [9];
end
```

Details

In the following, the two steps of the FME algorithm with RR (alg. 2) are explained in detail.

Step (1). The FME can be understood as the projection of an n-dimensional polytope in the direction of a standard basis vector e_i. For the coordinate i, a late-decision parameter dimension is chosen. When applying the FME algorithm to Solution-Compensation Spaces, the n-dimensional polytope represents the complete Solution Space.

In order to represent the FME algorithm graphically, a two-dimensional problem is considered. In two dimensions it is sufficient to consider the bounding vertices of the polygon [67]. Graphically speaking, these are the vertices an observer can see when looking at the polygon in the direction of the projection. As depicted in figure 4.6 (a), these vertices are either created by the combination of facets, which are directed towards and away from the view point (marked with blue and red arrows respectively), or by facets, which are parallel to the direction of the projection (marked with black arrows).

Definition 4.1. Front facing, back facing, and parallel facets:
A facet f_c is front facing if $\langle f_c, -e_i \rangle < 0$.
A facet f_c is back facing if $\langle f_c, -e_i \rangle > 0$.

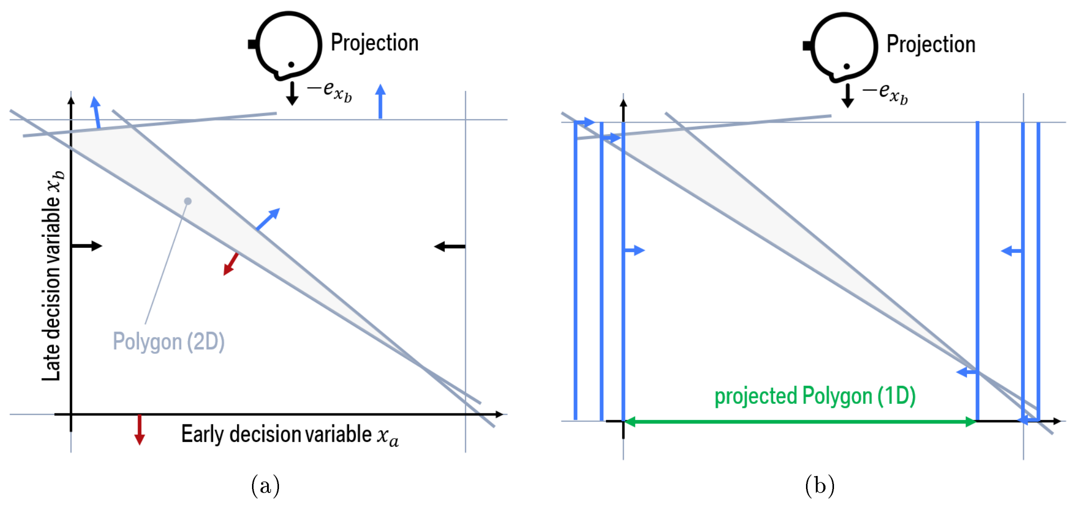


Figure 4.6 (a) Two-dimensional polygon, which represents the complete Solution Space before projection and the corresponding facets. Blue arrows indicate front facing facets, red arrows indicate back facing facets and black arrows represent parallel facets. (b) One-dimensional projected polygon (green arrow) and the corresponding constraints (blue lines).

A facet f_c is parallel if $\langle f_c, -e_i \rangle = 0$.
Where $-e_i$ is the direction of the projection and $< x, y >$ is the scalar product of the vectors x and y [36].

Unfortunately, a priori it is not known which facets define the polygon. Hence, all possible combinations of front and back facing facets as well as all parallel constraints need to be considered. This leads to a total of m_{t1} constraints when projecting a single dimension:

$$m_{t1} = m_f \times m_b + m_p, \tag{4.14}$$

where m_f is the number of front facing facets, m_b is the number of back facing facets and m_p is the number of parallel facets. The projected polygon is depicted in figure 4.6 (b). Only the two inner constraints are non-redundant. Note that according to equation (4.14) eight projected constraints should exist for this example, while only six are shown in figure 4.6 (b). The 7th constraint lies too far on the left so it is not shown. The 8th constraint does not exist since one of the front facing constraints is parallel to a back facing constraint.

How to derive the FME for an arbitrary high-dimensional problem $Gx \leq g_c$, with $G \in \mathbb{R}^{m_t \times d}$ and $g_c \in \mathbb{R}^{m_t}$, is explained in the following. $m_t = m + 2d$ is the total amount of performance as well as design space constraints of the initial system. FME solves problem statement 6, by projecting each single dimension of x_b in a recurring iteration of the same procedure. Considering the design space boundaries as linear constraints (see problem statement 5) and projecting only a single dimension, problem statement 6 can be simplified to:

Problem Statement 7.

$$\text{Given} \quad G \in \mathbb{R}^{m_t \times d}, \quad g_c \in \mathbb{R}^{m_t} \tag{4.15a}$$

$$\text{Find} \quad \tilde{G} \in \mathbb{R}^{m_{t1} \times (d-1)}, \quad \tilde{g_c} \in \mathbb{R}^{m_{t1}} \tag{4.15b}$$

$$\text{s.t.} \quad \forall \left\{ x_a \in \mathbb{R}^{d-1} | \exists x_b \in \mathbb{R} : G(x_a, x_b) \leq g_c \right\} \Rightarrow x_a : \tilde{G} x_a \leq \tilde{g_c} \tag{4.15c}$$

$$\wedge \quad \forall \left\{ x_a \in \mathbb{R}^{d-1} | \nexists x_b \in \mathbb{R} : G(x_a, x_b) \leq g_c \right\} \Rightarrow x_a : \tilde{G} x_a > \tilde{g_c}. \tag{4.15d}$$

The initial system $Gx \leq g_c$ is modified by rearranging the columns such that the dimension, which is to be eliminated x_{b_i}, is in the first column and by rearranging the rows according to the sign of the first entry: $+$,$-$ and 0. Each row, with a non-zero first entry, is normalised by dividing it through the absolute value of its first entry. This results in:

$$\begin{bmatrix} 1 & g'_1 \\ \vdots & \vdots \\ 1 & g'_{m_f} \\ -1 & g'_{m_f+1} \\ \vdots & \vdots \\ -1 & g'_{m_f+m_b} \\ 0 & g'_{m_f+m_b+1} \\ \vdots & \vdots \\ 0 & g'_{m_f+m_b+m_p} \end{bmatrix} \begin{bmatrix} x_b \\ x_a \end{bmatrix} \leq \begin{bmatrix} g'_{c,1} \\ \vdots \\ g'_{c,m_f+m_b} \\ g_{c,m_f+m_b+1} \\ \vdots \\ g_{c,m_f+m_b+m_p} \end{bmatrix} \tag{4.16}$$

Where $g'_i \in \mathbb{R}^{1 \times (d-1)}$ represents the modified i-th row of the initial matrix G excluding the first entry and $g'_{c,i} \in \mathbb{R}$ represents the modified i-th entry of g_c. Considering the first $m_f + m_b$ inequalities of equation (4.16), the following has to be true for x_b:

$$x_b \leq g'_{c,i} - g'_i x_a, \quad i = 1, ..., m_f \tag{4.17a}$$

$$x_b \geq -g'_{c,i} + g'_i x_a, \quad i = m_f + 1, ..., m_f + m_b \tag{4.17b}$$

Combining equations (4.17) yields:

$$\max_{i=m_f+1,\dots,m_f+m_b} -g'_{c,i} + g'_i x_a \leq x_b \leq \min_{j=1,\dots,m_f} g'_{c,j} - g'_j x_a \tag{4.18}$$

In case $m_f = 0/m_b = 0$, the upper/lower bound for x_b becomes $\inf/-\inf$. In these cases, inequality (4.18) can be ignored for the further construction of the projected system. After eliminating x_b from equation (4.18) and adding the inequalities in lines $m_f + m_b + 1$ to m_t of equation (4.16), the projected system can be written as:

$$(g'_i + g'_j)x_a \leq g'_{c,i} + g'_{c,j}, \quad i = 1,\dots,m_f, \quad j = m_f+1,\dots,m_f+m_b \tag{4.19a}$$

$$g_i x_a \leq g_{c,i}, \quad i = m_f+m_b+1,\dots,m_f+m_b+m_p \tag{4.19b}$$

The new system, which is the solution to problem statement 7, is given by equation (4.19). Equation (4.19a) poses a total of $m_f \times m_b$ constraints and equation (4.19b) poses a total of m_p constraints. Hence, the new system $\tilde{G}x_a \leq \tilde{g}_c$ consists of $m_{t1} = m_f \times m_b + m_p$ constraints and is $(d-1)$-dimensional. This derivation is based on [58, 67, 72, 73].

Step (2). After projecting the system in step (1), all redundant constraints are removed by applying the Clarkson Algorithm. The idea is that the amount of non-redundant constraints is generally much smaller than the total amount of constraints. Hence, the double exponential growth is prevented. For a more detailed explanation see subsection 4.2.3. Note that the system $\tilde{G}x_a \leq \tilde{g}_c$ derived in step (1) is considered the initial system for step (2) and hence the variables are renamed to $Gx \leq g_c$.

Redundancy is defined as follows:

Definition 4.2. Redundant: An inequality $G_i x \leq g_{c,i}$, $G_i \in \mathbb{R}^{1\times d}$, $g_{c,i} \in \mathbb{R}$ is called redundant in $Gx \leq g_c$, $G \in \mathbb{R}^{m\times d}$, $g_c \in \mathbb{R}^{m\times 1}$ if the set of solutions to $Gx \leq g_c$, stays unchanged when the inequality is removed.

The following problem needs to be solved:

Problem Statement 8. (H-Redundancy Removal/ Half-Space Redundancy Removal) Given $G \in \mathbb{R}^{m\times d}$, $g_c \in \mathbb{R}^m$ find an equivalent subsystem of $Gx \leq g_c$, which is free of redundancies.

Problem statement 8 can be solved by individually checking the redundancy of each constraint $G_k x \leq g_k$. Therefore, the following linear program denoted as *Red*(M, k) needs to be solved:

$$\begin{aligned} \max \quad & G_k x \\ s.t. \quad & G_i x \leq g_i, \quad \forall i = M/k \\ & G_k x \leq g_k + \epsilon, \quad \epsilon > 0, \end{aligned} \tag{4.20}$$

where $G_k, G_i \in \mathbb{R}^{1\times d}$, $g_k, g_i \in \mathbb{R}$, $M = \{1,\dots,m_t\}$, and x^* is the optimal solution. If $G_k x^*$ is strictly greater than g_k, the constraint is non-redundant. Otherwise, it is redundant

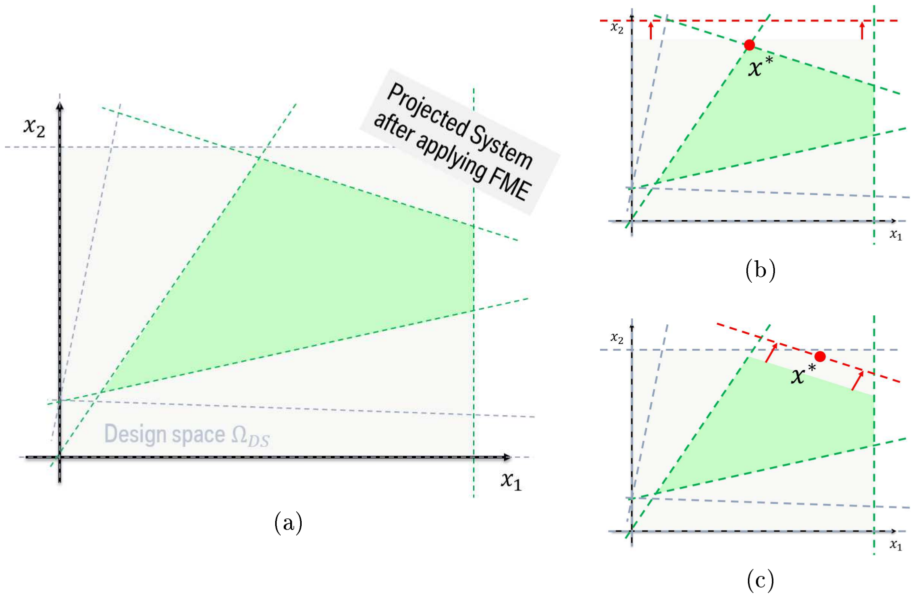

Projected System after applying FME
x_2
x_1
Design space Ω_{DS}
(a)
x^*
(b)
(c)

Figure 4.7 (a) A projected system after applying the FME. The green area is the complete Solution Space. The grey dashed lines are redundant constraints. The green dashed lines are non-redundant constraints. (b)&(c) The red dashed lines represent the relaxed constraint $G_i x \leq g_i + \epsilon$ and the red dot represents x^*, which is the optimal solution to $Red(M, i)$.

and can be removed from the set of constraints [27]. Adding ϵ to g_k can be considered as a relaxation of the concerning constraint. In order to visualise this, the initial system shown in figure 4.7 (a) is considered. Figure 4.7 (b) and figure 4.7 (c) show the relaxation of a single constraint (red dashed line). In figure 4.7 (c) the result of the optimisation x^* is not part of the initial polytope. Hence, adding ϵ to g_k enlarges the initial polytope and the constraint is non-redundant. The opposite is true for the constraint in figure 4.7 (b).

Often, the amount of redundant constraints is considerably higher than the amount of non-redundant constraints. This is visualised in figure 4.7 (a) where the grey dashed lines are redundant and only the green dashed lines are non-redundant constraints. Hence, the optimisation problem of equation (4.20) has a high number of constraints and solving it for each constraint is computationally expensive. Clarkson proposes a more efficient way to find all non-redundant constraints [9].

In the following, the Clarkson Algorithm is explained. Proof of why the Clarkson Algorithm can be applied in order to solve problem statement 8 is given in [69]. An interior point solution of the system, which satisfies $Gx < g_c$, can be found by using a classical interior point optimisation [46]. If an interior point is known, the Clarkson Algorithm (algorithm 3) returns the indices of the non-redundant constraints [9]. The Clarkson Algorithm consists of a while-loop, which runs through each single constraint and classifies it as either being redundant (Set R) or being non-redundant (Set S). When all constraints are classified, the algorithm concludes. At the beginning, the sets

Algorithm 3: Clarkson Algorithm [9]

```
Data: Initial System $Gx \leq g_c$ and an interior point $z$ that satisfies $Gz < g_c$ with
      $G \in \mathbb{R}^{m \times d}$, $z \in \mathbb{R}^d$ and $g_c \in \mathbb{R}^m$
Result: The set of all indices $S$ of the non-redundant constraints
begin
    $R := \{\}$, $S := \{\}$
    while $R \cup S \neq [d]$ do
        Choose any $i \in \{1, ..., d\}/(R \cap S)$ and check whether $i$ is redundant with
          respect to in $S$ using $Red(S, i)$. $x^*$ is the solution found for $Red(S, i)$;
        if $G_i x^* \leq g_i$ (i is redundant with respect to S) then
            $R = R \cup \{i\}$ ;
        else
            $S = S \cup \{j\}$, with $j = RayShoot(G, g_c, z, ray)$, with $ray = x^* - z$
        end
    end
end
```

Algorithm 4: RayShoot Algorithm [27]

```
Data: Initial System $Gx \leq g_c$, an interior point $z \in \mathbb{R}^d$ that satisfies $Gz < g_c$,
      with $G \in \mathbb{R}^{m \times d}$, $G_i \in \mathbb{R}^{1 \times d}$, $g_c \in \mathbb{R}^m$ and a $ray \in \mathbb{R}^d$
Result: The index $j$ of the facet that is hit first by the ray
begin
    $\Delta = 1$
    for $i = 1, ..., m$ do
        if $G_i ray > 0$ then
            $\delta = \frac{g_{c,i} - G_i z}{G_i ray}$;
            if $\delta < \Delta$ then
                $\Delta \leftarrow \delta$;
                $j \leftarrow i$;
            end
        end
    end
end
```

of classified constraints are empty. Starting with a chosen uncategorised constraint i, the algorithm checks whether it is redundant with respect to the Set S of already classified non-redundant constraints Red(S,i). Since S is empty at the beginning, the first chosen constraint i is always non-redundant with respect to S. Constraints that are checked later during the execution of the algorithm might be redundant with respect to S and hence can immediately be classified redundant to M since $S \subset M$. This is visualised in figure 4.8a where the red constraint is redundant with respect to the green set of classified non-redundant constraints S. In case the chosen constraint i is not redundant with respect to S the associated *ray* is calculated. The *ray* is the vector, which connects the interior point z to the optimal solution x^* found for $Red(S, i)$. A *ray* is visualised in figure 4.8b. Then, the RayShoot Algorithm is executed $RayShoot(G, g_c, z, ray)$ returns the index j of the constraint hyper-plane $G_j x = g_{c,j}$, which is hit first by the ray that shoots from the interior point z in the direction of x^*. Then the constraint j is categorised as non-redundant.

The RayShoot Algorithm (algorithm 4) [27], which is denoted as $RayShoot(G, g_c, z, ray)$ consists of a for-loop that runs through each constraint i of the initial system $Gx \leq g_c$. If the ray hits the considered constraint ($G_i ray > 0$), then δ is calculated. δ is a measure that determines which constraint hyper-plane $G_i x = g_{c,i}$ is hit first by the ray. The smaller the value of δ the earlier the constraint is hit. In figure 4.8c the purple constraint is hit before the blue constraint, which was initially moved, hence $\delta_2 < \delta_1$. If δ_i is less than the value of any constraint checked before $\delta < \Delta$, then Δ is updated with δ and the index i is saved as the currently first-hit constraint. The initial value of Δ is set to 1 since this assures that at least one constraint (namely the constraint that was relaxed in order to generate x^*, see figure 4.8c) fulfils the inequality $\delta_i < 1$.

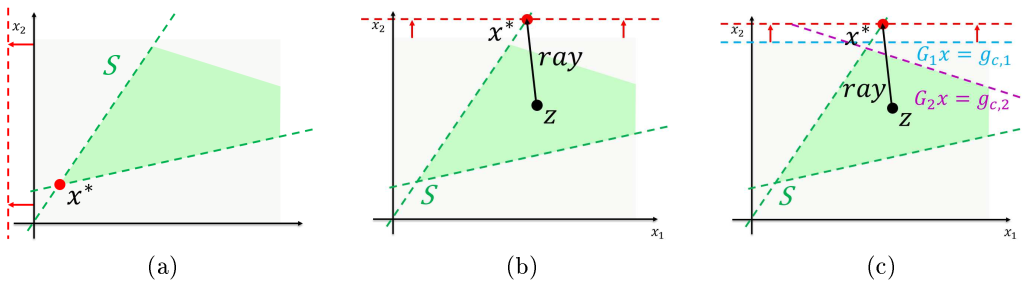


Figure 4.8 (a) The green two-dimensional polygon represents the complete Solution Space (see figure 4.7). Green dashed lines represent a subset S of all non-redundant constraints. The red line represents the relaxed constraint $G_i x \leq g_i + \epsilon$. The red dot x^* represents the optimised solution of $Red(S, i)$. (b) The black arrow represents the *ray* from inside the polygon z to the optimised solution x^*. (c) The blue and the purple dashed lines represent constraints that are hit by the *ray*.

Computational Effort

In the majority of applications where Solution-Compensation Spaces are applicable, multiple late-decision variables are considered. Therefore, the FME needs to be applied multiple times (equivalent to the amount of late-decision variables). The computational effort for multiple FME steps can be estimated with the number of constraints, which need to be calculated. In the worst case, the number of constraints splits evenly between front facing and back facing facets in each projection step, leading to

$$m_{r_1} = \left(\frac{m_t}{2}\right)^2 \tag{4.21}$$

constraints per step. Equation (4.21) is an appropriate estimate for real-world applications since the design space constraints $x^l_{\text{ds}} \leq x \leq x^u_{\text{ds}}$ are always evenly split between front facing and back facing facets.

Generalizing equation (4.21) for a total of q projected dimensions, without any RR between the steps, results in

$$m_{r_q} = \sum_{k=1}^{q} \frac{m_t^{(2^k)}}{2^{\sum_{i=1}^{k} 2^i}} \tag{4.22}$$

constraints, which need to be calculated. Hence, the number of constraints grows double exponentially with the number of projected dimensions. Even though calculating a new constraint is a simple operation where a front facing constraint is subtracted from a back facing constraint, the number of operations needed grows so fast, with the number of projections, that the computational effort becomes unbearable very quickly. Hence, RR is applied after each projection step of the FME. This prevents the exponential growth rate of the number of constraints [28].

In order to describe the computational effort of the RR techniques, which were introduced in this section, the number of Linear Programming Problems (LPPs), which need to be solved is crucial, since their computation dominates the other tasks. $LP(d,m)$ denotes the computational effort needed in order to solve an LPP $Ax \leq f_c$ with m constraints and d dimensions: $A \in \mathbb{R}^{m \times d}$.

In order to check a single constraint for redundancy, the optimisation problem $Red(M,k)$ (see equation (4.20)) needs to be solved. Therefore, the constraints require us to solve an LPP of size $d \times m$. Hence, the computational effort is $LP(d,m)$. Repeating this process for each constraint individually results in a total computational effort of $m \times LP(d,m)$. In order to check all m constraints for redundancy by applying the Clarkson Algorithm (see algorithm 3), the optimisation problem $Red(S,i)$ has to be solved m times. S is the set of already classified non-redundant constraints. The upper boundary for elements of the set S is s, which denotes the total amount of non-redundant constraints in the system $s \leq m$. Hence, the computational effort for the Clarkson Algorithm can be estimated with the upper boundary of $m \times LP(d,s)$, which is always less than the effort needed to check each constraint individually [27].

The combined computational effort of the FME in combination with the Clarkson Algorithm can be estimated with the computational effort needed for the Clarkson Algorithm only. This is due to the following facts:

- RR prevents the double exponential growth rate of the constraints,
- Calculating a new constraints is a simple subtraction operation, which gets dominated by the LPPs.

4.3. Non-linear Performance Constraints

In case the considered system includes non-linear performance constraints and cannot be linearised without losing significance, neither the basic projection algorithm (see subsection 4.2.2) nor the enhanced FME algorithm (see subsection 4.2.3) are applicable. Hence, in this chapter, new approaches to derive Solution-Compensation Spaces for highly non-linear and high-dimensional problem statements are examined. In section 4.3.2, a new algorithm based on the stochastic Solution Space algorithm by Zimmermann & von Hoessle [83] is introduced.

4.3.1. Support Vector Machine

One of the fundamental problems of learning theory is binary pattern recognition. Consider an empirical data set where each input vector x is assigned to either 1 or -1. The set is given by

$$x_1, \dots, x_i \in \mathbb{X} \times \pm 1. \tag{4.23}$$

Based on this information, a new function $f : x \to \{\pm 1\}$, which predicts the affiliation of a formerly unknown design x, is derived. There exists a plethora of machine learning methods that solve this problem, the most prominent ones are Artificial Neural Networks (ANNs) [3] and Support Vector Machines (SVMs) [31]. When applied to the binary pattern recognition problem, SVMs have several advantages compared to alternative machine learning techniques: absence of deceptive local minima in the optimisation, small amount of tuneable parameters, and excellent performance on high-dimensional data [31].

Idea

Considering the early-decision variable space only, Solution-Compensation Spaces can be seen as a classification problem. Each design inside the early-decision variable space can be classified as either good or bad depending on whether a design for late-decision variables exists such that all requirements are fulfilled (see problem statement 3).

A four-step procedure to compute Solution-Compensation Spaces using pattern recognition is proposed. In the first step, a set of MC samples in the early-decision variable design space is computed. In the second step, an optimisation for each sample is run, which aims to find a fitting late-decision design such that all requirements are fulfilled. Based on the result of the optimisation each sample is labelled as either good or bad. In the third step, the binary pattern recognition problem is solved by applying SVM. In the fourth step, a stochastic Solution Space algorithm [83] is run on the SVM classifier in order to optimise the Solution-Compensation Space with respect to the size measure μ.

Computational Effort

The effectiveness of SVM is directly linked to the volume fraction of good designs in the early-decision variable space. If the fraction is very low, then most sample points are assigned as bad designs. Hence, a large number of sample points needs to be computed. Considering an example problem with eight dimensions $d = 8$ and assuming 10% of each dimension contains acceptable values $v_i = 0.1, i = 1, \ldots, 8$ leads to a very small total fraction of good designs $V = v^d = 0.1^8$. This means that on average $100,000,000$ MC samples are needed to find just one good design. Keep in mind that for each sample an optimisation for the late-decision variables needs to be run. The computational effort is very high and usually grows exponentially with the number of dimensions.

Unfortunately, the SVM is hence not applicable to the eight-dimensional vehicle dynamics problem introduced in section 7.1. Therefore, the SVM approach is not expanded upon in this thesis. An in-depth analysis of SVM for Solution-Compensation Spaces can be found in [67].

4.3.2. Stochastic Solution-Compensation Space Algorithm

The original stochastic algorithm to compute Solution Spaces for robust design [83] was developed in order to optimise the size of box-shaped Solution Spaces for arbitrary non-linear high-dimensional systems (see section 2.1). With some modifications, its procedure can be applied to Solution-Compensation Spaces as well. In this section, a modification of the stochastic Solution Space algorithm is introduced in order to compute Solution-Compensation Spaces with non-linear performance constraints of equation (4.2). Unfortunately, the algorithm does not guarantee a resulting Solution-Compensation Space, which fulfils all constraints. Instead a relaxed problem statement of equation (4.24) is solved. It demands, with a certain confidence, that at least a certain fraction of the resulting Solution-Compensation Space fulfils all constraints. It is shown that this algorithm scales very well with the number of dimensions d and thus can be applied to high-dimensional problems.

Idea

Since the design space for the early-decision variables $\Omega_{\mathrm{ds},a}$ is assumed to be continuous, any considered subspace Ω_a includes an infinite number of designs. In addition, the performance function $f(x_a, x_b)$ is usually unknown. Hence, it is impossible to check whether for each design $x_a \in \Omega_a$ an x_b exists such that the constraint $f(x_a, x_b) \leq f_c$ is fulfilled. Instead of solving the non-linear Solution-Compensation Space problem statement of equation (4.2), a relaxed Solution-Compensation Space problem based on Bayesian probabilities as described in [44] is solved:

Problem Statement 9.

$$\underset{\Omega_a \subseteq \Omega_{\mathrm{ds},a}}{\text{maximise}}\, \mu(\Omega_a) \tag{4.24a}$$

$$s.t. \quad P(a_l < a | m, N) > 1 - \alpha_c, \tag{4.24b}$$

where N is the number of MC early-decision variable sample designs $x_a \in \Omega_a$, m is the number of good sample designs $x_a \in \Omega_a\, \exists x_b \in \Omega_b, f(x_a, x_b) \leq f_c$, a_l is the lower boundary of the confidence interval and $1 - \alpha_c$ is the confidence level. Since the relaxed problem statement 9 does not ensure full feasibility any more, it is only applicable to certain real world application. It is not recommended for safety relevant designs.

The introduced algorithm starts with an initial box around a good design, the box is iteratively modified in order to optimise its size, while retaining a certain likelihood that at least a certain fraction of the volume consists of good designs. This condition is checked by applying Bayesian statistics to a set of MC sampled designs (see Section 2.1.2). The stochastic Solution Space algorithm is executed with two major modifications:

- The optimal box is searched within the subspace of the complete design space $\Omega_{\mathrm{ds},a} \subset \Omega_{\mathrm{ds}}$. Hence, the box is only optimised with respect to the early-decision variables x_a. The late-decision-variable space Ω_b is fixed.

- For each sample design x_a in the early-decision-variable space, an optimisation in the late-decision-variable space Ω_b is executed in order to determine whether it fulfils $f(x_a, x_b) \leq f_c$ and hence, is a good design.

Details

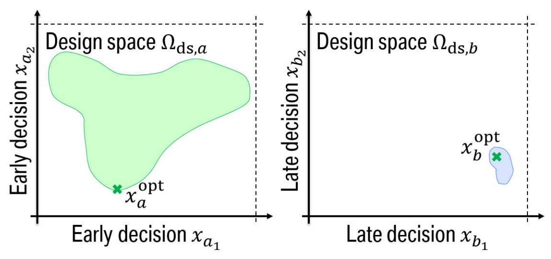
In the following, the steps of the stochastic Solution-Compensation Space algorithm (alg. 5) are explained in detail.

Initialisation. In order to be able to construct a first candidate box, a good design $f(x_a, x_b) \leq f_c$ needs to be known. Therefore, a classical optimisation such as differential evolution [66] is carried out. In case no good design can be found, the algorithm is aborted. Otherwise a first candidate box Ω_a including the optimised design x_a^{opt} is constructed. Its volume is zero and it is part of the early-decision variable design space $\Omega_{\mathrm{ds},a}$.

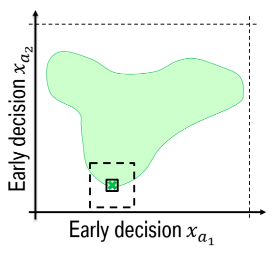
Phase I. In phase I the box is iteratively modified until its size $\mu(\Omega_a)$ does not change significantly any more. The iterative procedure to modify the box consists of four steps.

Step (1). In the first step, the candidate box is extended. Since there is no information on the structure of the performance function f, the algorithm enlarges the boundaries of the candidate box equally by adding a constant interval size to each boundary. The interval size is determined by the product of the growth rate g and the size of the design space $(x^{\mathrm{u}}_{\mathrm{ds},j} - x^{\mathrm{l}}_{\mathrm{ds},j})$ in the early variable dimension $j \in 1, \ldots, p$. This results in:

$$\Omega_a^{\mathrm{update}} = [x^{\mathrm{l}}_{\mathrm{ds},1} - g(x^{\mathrm{u}}_{\mathrm{ds},1} - x^{\mathrm{l}}_{\mathrm{ds},1}) \quad x^{\mathrm{u}}_{\mathrm{ds},1} + g(x^{\mathrm{u}}_{\mathrm{ds},1} - x^{\mathrm{l}}_{\mathrm{ds},1})] \times \cdots$$

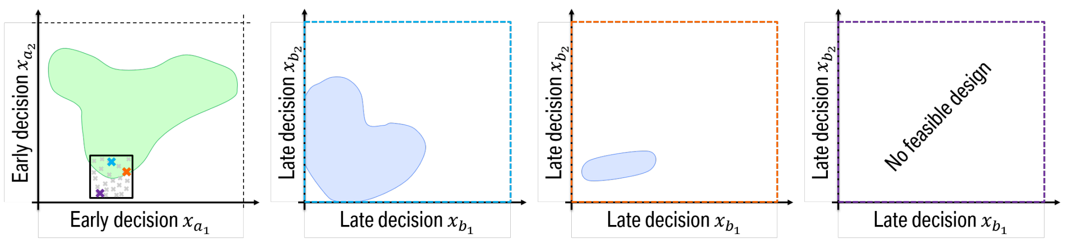


(a) Initialisation



(b) Phase I-(1)

Figure 4.9 A four-dimensional problem with two early- and two late-decision variable dimensions. (a) The green area represents the complete Solution-Compensation Space for the early-decision variable. The blue area represents the complete Solution Space for a fixed early-decision variable design x_a^{opt}. The green cross represents the resulting design of a classical optimisation algorithm x^{opt}. (b) The black box represents the initial candidate box. The dashed box represents the candidate box after a single extension step.



(a) (b) (c) (d)

Figure 4.10 Phase I-(2). (a) The crosses represent sample designs in the early-decision variable space. (b)-(d) For each specified set of early-decision variables the space of permissible late-decision variable values is shown.

$$\cdots \times [x^{\mathrm{l}}_{\mathrm{ds},p} - g(x^{\mathrm{u}}_{\mathrm{ds},n} - x^{\mathrm{l}}_{\mathrm{ds},p}) \quad x^{\mathrm{u}}_{\mathrm{ds},n} + g(x^{\mathrm{u}}_{\mathrm{ds},n} - x^{\mathrm{l}}_{\mathrm{ds},p})]. \tag{4.25}$$

Step (2). In the second step, N MC samples in the early-decision variable space $x_a^j \in \Omega_a$ for $j = 1, \ldots, N$ are computed.

Step (3). In the third step, good and bad sample designs are determined. For each sample design x_a^j a classical optimisation is run to solve

$$\underset{x_b \in \Omega_b}{\text{minimise}}\, f(x_a^j, x_b). \tag{4.26}$$

The optimisation is terminated once a valid design $f(x_a^j, x_b) \leq f_c$ is found or a maximum number of iteration steps is reached. In case a valid design is found, the design x_a^j is categorised as a good design. Else, it is categorised as a bad design. As the optimisation algorithm of choice "a modified particle swarm optimiser" [60] is suggested in this thesis.

Particle swarm optimisation has received broad attention in recent years since it has a fast convergence to acceptable solutions [4]. Those attributes make it especially fitting for the posed optimisation problem of equation (4.26) since the convergence speed of the algorithm is very important, due to the fact that it has to be executed for each single sample design, while the accuracy is not as important, since it is sufficient to find any valid design. This might lead to some sample designs being classified as false negatives.

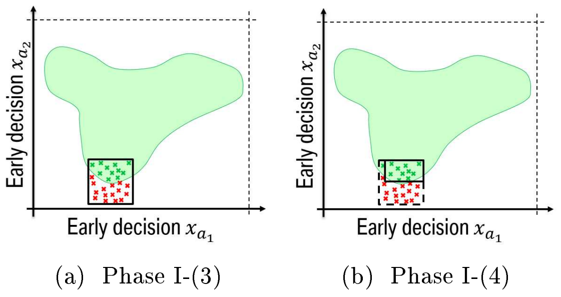


Figure 4.11 (a) Green/Red Crosses represent good/bad designs. (b) The dashed/continuous black box represents the candidate box before/after a removing all bad designs.

Step (4). The fourth step is only executed if the fraction of good designs m/N is not sufficiently large to fulfil equation (4.24b). In this case, bad sample designs are removed from the candidate box Ω_a until only good designs are left. Therefore, algorithm 6 is executed. In accordance with Zimmermann & von Hössle [83], the removal is performed in three nested loops. In the outer loop each single good design x_a^i is considered. For each good design a new Solution-Compensation Space $\Omega_a^{i,N}$, including only good designs, is derived. In the first inner loop each bad design is considered. Therefore, the bad designs are sorted according to their performance $f(x_a^i, x_b)$ where x_b is the best design found in the step "Determine good and bad sample designs" of algorithm 5 for the respective x_a^i. Starting with the highest performance bad design, $\Omega_a^{i,j}$ is modified for each bad design such that the current bad design is cut off, while the current good design is retained. In the second inner loop each early variable dimension is considered in order to determine the optimal cut. Figure 4.12 shows a two-dimensional example for the second inner loop. Considering the highlighted green and red crosses as currently considered designs in the loop, there are two options on how to perform the current cut 1. Option 1 is superior since no good sample designs are lost. In case multiple options to perform the cut are equal, the option where the least amount of volume is lost is chosen. The resulting cut shown in figure 4.12 (c) is obtained after iterating through the second inner loop once. Since there are still bad designs left, a second cut is needed as shown in figure 4.12 (d). After executing the outer loop for each good sample design, the optimal candidate box with respect to its size $\max_i \mu(\Omega_a^i)$ is chosen.

Phase 1 of algorithm 5 is executed until the size of the candidate box does not change significantly any more $\|\frac{\Omega^{i+1}}{\Omega^i}\| \leq c_{\text{tol}}$. After Phase 1 there are a couple of interim steps where a new MC sample is computed and all sample designs are categorised.

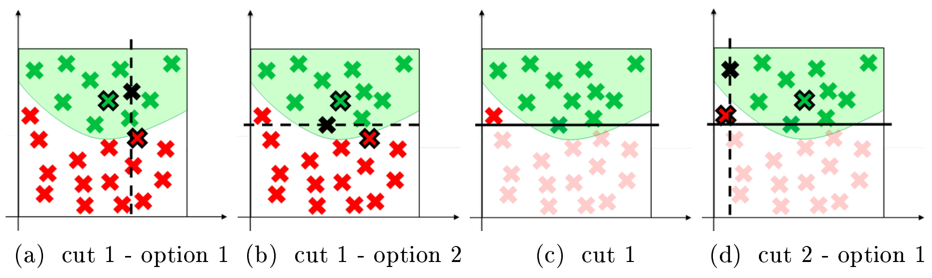


Figure 4.12 Green/red crosses represent good/bad designs. The dashed lines represent different options on how to cut the candidate box. The continuous line in (c) represents the first cut, which is chosen.

Phase II. Phase 2 is also called the consolidation phase. It is similar to Phase 1 with the most important difference being the missing growth step. At the start of each loop we check whether the amount of good samples in the candidate box Ω_a is sufficient to fulfil the requirement $P(a_l < a|m, N) > 1 - \alpha_c$. If it is, Phase 2 is immediately terminated and the current box is the final result. Otherwise, in step (1) bad sample designs are removed according to algorithm 6. Then, an MC sample is calculated, all sample designs are categorised, and Phase 2 starts over.

Computational Effort

Computational runtime for the stochastic algorithm (alg. 5) is usually dominated by the runtime of the function evaluation $f(x)$ and hence, scales linearly with the number of iterations, samples, and particles:

$$\sim v_1 N_1 v_2 N_2 F, \tag{4.27}$$

where v_1 equals the number of iterations for the box algorithm from both Phase I and II as well as the transition step $v_1 = v_1^{\text{Phase I}} + v_1^{\text{Phase II}} + 1$. v_2 equals the number of iterations needed during the particle swarm optimisation. N_1 equals the number of samples per iteration while N_2 equals the number of particles used in the particle swarm optimisation.

The amount of iterations needed in Phase I, $v_1^{\text{Phase I}}$, is not predetermined, instead a boundary c_{tol} for the minimal change of the box size is given. If $\|\frac{\Omega^{i+1}}{\Omega^i}\| \leq c_{\text{tol}}$ then Phase I is terminated. In order to keep the computational effort in check, an upper boundary for the number of iterations can be implemented. The number of iterations for Phase II, $v_1^{\text{Phase II}}$, is also not predetermined. Phase II aborts as soon as a sufficient amount of samples in the current candidate box is classified as good designs. Usually, Phase II needs very few iterations compared to Phase I.

N_1 is predefined. In most applications 100 samples are sufficient. According to Bayesian theory if 100/100 random samples are good designs with a likelihood of 95%, at least 97% of the candidate box are good designs [83]. Since no knowledge about the

	v_1^{PhaseI}	$v_1^{PhaseII}$	N_1	v_2	N_2
parameter value	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
size $\mu(\Omega)$	$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$
confidence level $1-\alpha_c$	$\rightarrow$	$\uparrow$	$\uparrow$	$\rightarrow$	$\rightarrow$

Table 4.1 The influence of the parameters of the Stochastic SCS Algorithm on the size and the confidence level of the resulting SCS

shape of the complete Solution Space is available in advance, a prior with a β-distribution of $\beta(1,1)$ is assumed [44]. These statements are independent from the number of dimensions.

v_2 differs based on the termination criterion. In order to keep the computational effort in check, a maximum number of iterations is predefined. If that number is reached, the design is classified as a bad design. Hence, a lower maximum number of iterations leads to more false negative classifications.

Since the computational effort of the PSO algorithm gets dominated by the function evaluations, it makes sense to choose N_2 such that the expected number of function evaluations is minimised. Therefore, Trelea [70] suggests a medium number of particles. In his experiments, the optimum was generally between 15 and 30 particles. When using too few particles, the success rate went down significantly. Using too many particles resulted in a higher computational effort. F is the time needed for a single function evaluation. Table 4.1 summarises the influence of the parameters on the size and the confidence level of the resulting Solution-Compensation Space. As long as saturation is not reached, an increased number of iterations v_1 leads to a better solution with respect to the size measure. The number of samples per box N_1 increases the confidence level but decreases the size of the box since the size of the bad solution space included in the final result decreases. The opposite is true for the particle swarm parameters v_2 and N_2. If they are increased, the number of false negative design points decreases, which increases the size of the resulting Solution Space.

Since neither v_1, v_2, N_1 nor N_2 depend on the dimensionality of the problem, the stochastic Solution-Compensation Space algorithm is independent from the number of dimensions. Despite this fact, in most real world applications higher dimensions will still lead to an increased computational effort. First, more different options for possible cuts need to be considered in step (4). Second, in most cases high-dimensional function evaluations are more costly. Third, more iterations are needed to improve the likelihood to find the global optimum.

In general, even if the optimisation parameters v_1, v_2, N_1, and N_2 are chosen optimally, the number of function evaluations needed to obtain a satisfying result is still relatively high. Hence, this method is mostly recommended for systems with a low effort of evaluating a single design. In case function evaluations are too expensive to apply the stochastic Solution-Compensation Space algorithm, ANNs are a recommended method to speed up the process by approximating the original function [3].

Algorithm 5: Algorithm for the stochastic SCS algorithm based on [83]

```
Data: Initial System $f(x_a + x_b) \leq f_c$ with $x_a \in \Omega_{\mathrm{ds},a}$ and $x_b \in \Omega_{\mathrm{ds},b}$
Result: Optimised box $\Omega_a$ s.t. at least $a_l$ percent of the volume of $\Omega_a$ consists of
        good designs with a likelihood of at least $1 - \alpha_c$
begin
    Initialisation
    Classical optimisation on $\Omega_{\mathrm{ds}}$ to identify a good design $x_a^{\mathrm{opt}}$;
    Construct a first candidate box $\Omega_a$ with zero volume.;
    Phase I
    while $\mu(\Omega_a)$ is changing significantly do
        (1) Extend candidate box;
        (2) Compute MC sample in $\Omega_a$;
        (3) Determine good and bad sample designs:
        for each sample design $x_a^j$ of the MC sample do
            Optimise $x_b$ to check if an $x_b \in \Omega_b$ exists such that $f(x_a^j, x_b) \leq f_c$;
        end
        (4) Remove bad sample designs.;
    end
    Compute MC sample in $\Omega_a$;
    Determine good and bad sample designs:
    for each sample design $x_a^j$ of the MC Sample do
        Optimise $x_b$ to check if an $x_b \in \Omega_b$ exists such that $f(x_a^j, x_b) \leq f_c$;
    end
    Phase II
    while the threshold for the fraction of good sample designs in the candidate
     box is not reached do
        (1) Remove bad sample designs.;
        (2) Compute MC sample in $\Omega_a$;
        (3) Determine good and bad sample designs:
        for each sample design $x_a^j$ of the MC Sample do
            Optimise $x_b$ to check if an $x_b \in \Omega_b$ exists such that $f(x_a^j, x_b) \leq f_c$;
        end
    end
end
```

```
Algorithm 6: Algorithm to remove bad sample designs in accordance with [83]
Data: Initial candidate box for the early-decision variables $\Omega_a$ including $N$
      sample designs $x_a^i \in \Omega_a, i = 1, \dots, N$ of which $m < N$ are categorised as
      good and a size measure $\mu$. The bad sample designs $x_a^i, i = m+1, \dots, N$
      are sorted by performance $f(x_a^i, x_b)$ in descending order.
Result: New candidate box for the early-decision variables $\Omega_a$ including only
        good sample designs $x_a^i \in \Omega_a, i = 1, \dots, m$.
begin
    for each good sample design $x_a^i, i = 1, \dots, m$ do
        Set $\Omega_a^{i,0} := \Omega_a$;
        for each bad sample design $x_a^j, j = m+1, \dots, N$ do
            for each early variable dimension $k = 1, \dots, p$ with
              $\Omega_a^{i,j} = [x_1^l \, x_1^u] \times \dots \times [x_p^l \, x_p^u]$ do
                if $x_{a,k}^i < x_{a,k}^j$ then
                    $N_k :=$ number of good designs contained in $\Omega_a^{i,j,k}$. $\Omega_a^{i,j,k}$
                     represents a modified $\Omega_a^{i,j}$ with $x_k^u := x_{a,k}^s$. Choosing
                     $s \in [1, \dots, m]$ such that $x_{a,k}^s \in \Omega_a^{i,j}$ minimises $x_{a,k}^j - x_{a,k}^s > 0$.
                else
                    $N_k :=$ number of good designs contained in $\Omega_a^{i,j,k}$. $\Omega_a^{i,j,k}$
                     represents a modified $\Omega_a^{i,j}$ with $x_k^l := x_{a,k}^s$. Choosing
                     $s \in [1, \dots, m]$ such that $x_{a,k}^s \in \Omega_a^{i,j}$ minimises $x_{a,k}^s - x_{a,k}^j > 0$.
                end
            end
            Choose $s \in [1, \dots, p]$ such $N_s$ is maximised and set $\Omega_a^{i,j+1} := \Omega_a^{i,j,s}$
        end
        Set $\Omega_a^i := \Omega_a^{i,N}$.
    end
    Set $\Omega_a := \max_i \mu(\Omega_a^i)$.
end
```

Chapter 5

Optimal Constraint Relaxation for Solution Spaces

The size of box-shaped Solution Spaces is optimised in order to maximise the size of the permissible intervals and hence enable maximum robustness of the development process. In case the specific robustness requirements for each optimisation variable are known, a minimal size for each permissible interval can be derived. The Cartesian product of these permissible intervals represents a box of fixed size. An Solution Space algorithm using Vertex Tracking can be applied to optimise the position of the box centre until all performance constraints are fulfilled and a valid Solution Space containing only good designs is found. In case no such position is found, an Solution Space, which fulfils the performance as well as the robustness requirements does not exist. Hence, the problem statement is infeasible. In this case, a set of relaxed constraints is optimised such that a feasible problem statement is obtained. The set of relaxed constraints is optimal with respect to a target function that weighs the relaxation of different constraints and aims to relax each constraint as little as possible. We call this Optimal Constraint Relaxation (OCR). Optimal Constraint Relaxation is categorised as a penalty method. Penalty methods are used to analyse infeasible problem statements [7]. In this chapter, mathematical formulations of the underlying optimisation problem are stated and an approach to compute Optimal Constraint Relaxation for Solution Space with linear and non-linear performance constraints is proposed.

5.1. Idea, overview, and problem statement

As introduced in section 2.1.2, box-shaped Solution Spaces are maximised with respect to a size measure in order to maximise the size of the permissible intervals. In many industrial applications, the size of the permissible intervals for crucial design variables derived by the Solution Space approach is not sufficiently large to compensate for all uncertainties during the development process. In chapter 4, a method to enlarge the size of permissible intervals for a crucial subset of all design variables was introduced. Unfortunately, the method is only applicable if a subset of late-decision variables can be identified. Even if that is the case, the resulting intervals might still not be large enough to compensate for all uncertainties during the development process. Hence, a new method is introduced in this chapter, which is able to compute an optimal set of relaxed performance constraints. The relaxed constraints are such that a box-shaped Solution Space can be optimised, which fulfils all robustness requirements while containing a sufficiently large fraction of good designs.

5.1.1. Basic idea of Optimal Constraints Relaxation for Solution Spaces

In order to illustrate the basic idea of computing Optimal Constraint Relaxation for Solution Space, a two-dimensional example problem is shown in figure 5.1a. The three straight lines forming the grey triangle represent the constraints. The triangle depicts the area of all good designs. The design space Ω_{ds} is represented by the grey lines. Inside of the complete Solution Space, the box-shaped Solution Space is shown (black). In addition, a box of fixed size is shown (blue). The box is such that the interval sizes for x_1 and x_2 exactly fulfil the robustness requirements, meaning they have a fixed length. Unfortunately, the box does not fit inside the complete Solution Space. Hence, an optimal set of relaxed constraints is computed, which provides a complete Solution Space that is large enough to include the box, as shown in figure 5.1b.

According to Chinneck there are three different ways in which a design can be optimal with respect to the violation of constraints: NINF, SINF, and SLVF (see subsection 2.3.2) [7]. For our applications in chassis design the amount by which a constraint is violated is essential, therefore we will focus on minimising the SINF in this thesis. SLVF could be considered as an alternative to SINF in order to circumvent the row scaling problem (=scaling of the output space). But we do not apply SINF in this thesis since the computational effort to compute it is too high. For SINF several versions are commonly used, e.g. weighed fixed penalties (independent of the magnitude of the individual violations) and weighed linearly and quadratically increasing penalties. How to apply these methods for single design optimisation for linear problem statements is shortly introduced in subsection 2.3.2 and described in detail by Chinneck [7]. In the following, we extend these ideas for box-shaped designs of fixed size and for monotonic non-linear constraint functions (see Section 5.3).

Even though the general purpose of Optimal Constraint Relaxation is to analyse an infeasible problem statement the algorithms we introduce in this chapter are applicable to both feasible and infeasible systems:

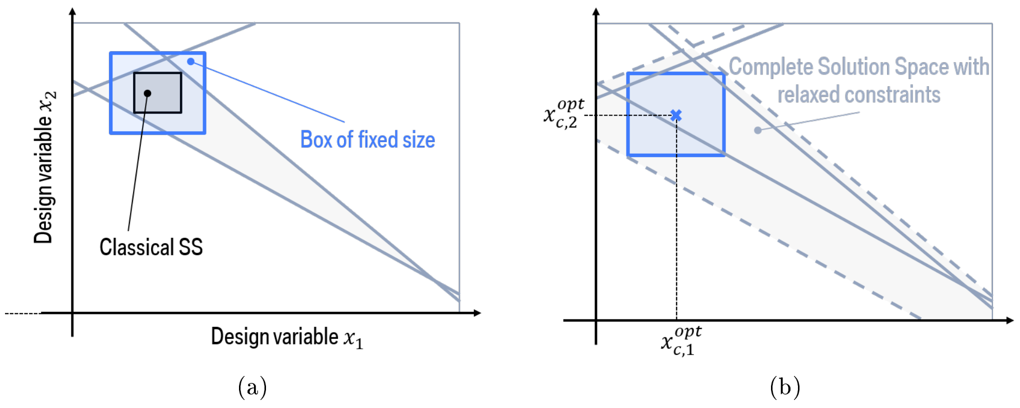


Figure 5.1 (a) The box-shaped Solution Space (black) and the box of fixed size, which fulfils the robustness requirements (blue). (b) An optimised set of relaxed constraints and the resulting Solution Space (blue).

- If the problem statement is **feasible** the result of Optimal Constraint Relaxation for Solution Spaces will be a position for the box of fixed size such that it includes only good designs.
- If the problem statement is **infeasible** the result of Optimal Constraint Relaxation for Solution Spaces will be a position for the box of fixed size such that all constraints are violated as little as possible. The amount of violation at critical vertices of the box determines by how much each constraint needs to be relaxed in order to fit the box. Thus, this also determines the optimal set of relaxed constraints.

5.1.2. Design Process with Optimal Constraint Relaxation for Solution Spaces

Optimal Constraint Relaxation for Solution Spaces is applied in the system design phase when, in addition to the classical performance constraints, robustness requirements exist. In industry, this is often the case if the design variables cannot be set to an arbitrarily precise value since they depend on additional factors. A typical example is tyre design where the design variables are the Functional Tyre Characteristics (FTCs), which are adjusted by varying the rubber compounds. Unfortunately, the dependence of the FTCs on the rubber compounds is highly non-linear and underlies statistical deviations. Hence, a minimal interval size for each FTC is required. In this case, we need to check whether the problem statement is feasible. In other words: Does the box with the minimal interval size for each design variable fit into the complete Solution Space? This can be checked by applying the Optimal Constraint Relaxation algorithms introduced in sections 5.2.1 and 5.3 for linear and non-linear constraints, respectively. If the box fits into the complete solution space, we can apply one of the classical Solution Space optimisers (e.g. SSA,

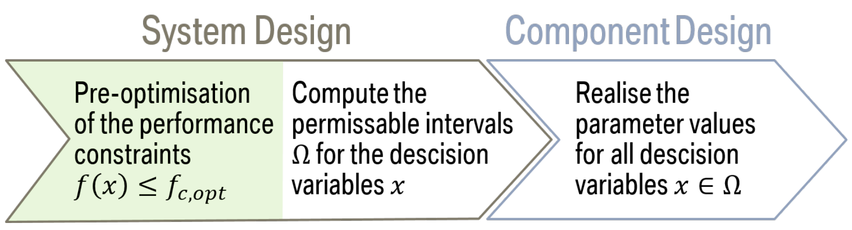

System Design
Component Design
Pre-optimisation of the performance constraints $f(x) \leq f_{c,opt}$
Compute the permissable intervals Ω for the descision variables x
Realise the parameter values for all descision variables $x \in \Omega$

Figure 5.2 Approach using system and component design of classical Solution Spaces with the addition of a pre-optimisation step for infeasible problem statements.

VT). Therefore, we use the box found by Optimal Constraint Relaxation as the starting box and we add additional constraints to the optimisation algorithm such that no interval can shrink below its initial size. This way we maximise the size of the Solution Space while ensuring that all robustness constraints are fulfilled. This design process matches the classical Solution Space approach with a system design and a component design step (see figure 4.3). If the box does not fit in the complete solution space, the Optimal Constraint Relaxation algorithm minimises the SINF. The result is used to determine an optimal set of relaxed constraints, which can then be used to make the problem feasible. No additional optimisation algorithm is needed afterwards since we already know the optimal position and size of the Solution Space with respect to the relaxed constraints. As shown in figure 5.2, the determination of the relaxed set of constraints can be seen as a pre-optimisation step during the system design phase.

5.1.3. Problem statement

The initial problem statement, which shall be solved is the classical Solution Space problem statement [83] with additional robustness constraints. Recall that a box-shaped Solution Space can be written as $\Omega = I_1 \times \ldots \times I_d$ with $I_i = [x_i^{lb}\ x_i^{ub}]$. Where $x_i^{lb}/\ x_i^{ub}$ are the lower/ upper boundary of the design interval in dimension i. Robustness constraints refer to the size of the permissible intervals for the design variables: $x^{ub} - x^{lb} \geq x_r$. This leads to the following problem statement:

Problem Statement 10.

$$\underset{\Omega\subseteq\Omega_{\text{ds}}}{\text{maximise}}\,\mu(\Omega) \tag{5.1a}$$

$$s.t. \quad \forall x \in \Omega, f(x) \leq f_c;\ x^{ub} - x^{lb} \geq x_r. \tag{5.1b}$$

As visualised in figure 5.1 (a), this problem statement can be infeasible, in which case no result is obtained. The idea of Optimal Constraint Relaxation for Solution Space is to reformulate problem statement 10 such that it is feasible for any arbitrarily strict requirements. Instead of optimising the size of an Solution Space, the goal is to optimise an Solution Space, which fulfils the robustness requirements exactly $x^{ub} - x^{lb} = x_r$. This leads to a box-shaped Solution Space of fixed size $\Omega = [x_c - \frac{1}{2}x_r;\ x_c + \frac{1}{2}x_r]$, x_c is the centre of the Solution Space. In case no such Solution Space can be found, the performance constraints are relaxed $f_c + \Delta f_c$. The new objective function minimises the amount of relaxation. The degrees of freedom are the position of the centre of the Solution Space x_c as well as the relaxation of the performance constraints Δf_c. This leads to the following problem statement:

Problem Statement 11.

$$\underset{\Delta f_c, x_c}{\text{minimise}}\, g(\Delta f_c) \tag{5.2a}$$

$$s.t. \quad \Delta f_c \geq 0; \qquad \forall x \in \Omega, f(x) \leq f_c + \Delta f_c; \quad \Omega \subseteq \Omega_{ds}\ . \tag{5.2b}$$

$$\text{with } \Omega = [x_c - \frac{1}{2}x_r;\ x_c + \frac{1}{2}x_r]\ .$$

Since the goal is to relax the performance constraints as little as possible the target function $g : \mathbb{R}^m \to \mathbb{R}$ has to fulfil the following criteria for any $\Delta f_c \geq 0,\ \Delta f_c \in \mathbb{R}^m$

1. $g(\Delta f_c) \geq 0$
2. $g(\Delta f_c) = 0 \Leftrightarrow \Delta f_c = 0$
3. $g(\Delta f_c)$ is strictly monotonically increasing with respect to each of its variables, which means:

$$g(\Delta \tilde{f_c}) - g(\Delta f_c) > 0 \quad \forall\, \Delta \tilde{f_c}, \Delta f_c,\ \Delta \tilde{f}_{c,i} > \Delta f_{c,i},\ \Delta \tilde{f}_{c,j} = \Delta f_{c,j}, \tag{5.3}$$

for any $i \in \{1, \dots, m\}$ with $j = \{1, \dots, m\}/\{i\}$ [25].

Shao proposes the following target function [59]

$$g(\Delta f_c) = \sum_{i=1}^{m} g_i(\Delta f_{c,i}) = \sum_{i=1}^{m} q_i(\Delta f_{c,i})^2 + c_i \Delta f_{c,i} + d_i \text{sign}(\Delta f_{c,i}), \tag{5.4}$$

with $q_i > 0$ and $c_i, d_i \geq 0$, since it is applicable to many industrial problems due to its flexibility and can be computed efficiently with the application of classical quadratic programming approaches [5]. In this thesis, we apply a distinct version of Shao's target function with $c_i, d_i = 0$. This target function relates to the Euclidean distance. It penalises large relaxations for individual constraints and hence results in a balanced relaxation of all constraints.

5.2. Linear Performance Constraints

In this thesis, we introduce an algorithm to compute Optimal Constraint Relaxation for Solution Spaces in systems with linear performance constraints (see Section 5.2.1):

$$f(x) = Fx + c \qquad F \in \mathbb{R}^{m \times d};\ c \in \mathbb{R}^m. \tag{5.5}$$

5.2.1. Problem statement

Similarly to expression (5.2), an optimal set of relaxed constraints as well as an optimal position for the centre of the Solution Space are sought.

Problem Statement 12.

$$\underset{\Delta f_c, x_c}{\text{minimise}}\, g(\Delta f_c) \tag{5.6a}$$

$$s.t. \quad \Delta f_c \geq 0; \qquad \forall x \in \Omega, Fx \leq f_c + \Delta f_c; \quad \Omega \subseteq \Omega_{ds}\ . \tag{5.6b}$$

$$\text{with } \Omega = [x_c - \frac{1}{2}x_r;\ x_c + \frac{1}{2}x_r]$$

Note that the constant c is included in f_c.

In this section, the idea, the notation, and the computational effort for Optimal Constraint Relaxation for Solution Space with linear performance constraints are described.

5.2.2. Idea

In order to solve the linear problem statement 12, we can apply the VT idea first introduced in [19, 21]. According to Erschen, for box-shaped Solution Spaces with linear performance constraints, it is sufficient to assess only particular vertexes of the polytopes to ensure that the enclosed set contains only good designs. Hence, we can modify the constraint function (5.6b) such that a finite number of designs x have to be checked, namely particular vertices.

5.2.3. Details

In the following, we apply the VT idea to modify problem statement 12 such that it can be solved by a classical interior point optimisation [80]. The target function is unchanged but the degrees of freedom are reduced to the centre of the box x_c.

Figure 5.3 visualises the idea of Vertex Tracking. It is sufficient to check a critical vertex, which is determined by the normal direction of the respective constraint hyperplane. If the critical vertex is a good design each design included in the considered Solution Space is a good design. A detailed proof that this idea is applicable for any linear constraint in arbitrary dimensions can be found in [20].

The following derivation is done in accordance with [59]. For simplicity we consider only a single performance constraint, which is determined by the i-th row of the constraint function (5.6b):

$$\forall x \in [x_c - \frac{1}{2}x_r;\ x_c + \frac{1}{2}x_r], f_i x \leq f_{c,i} + \Delta f_{c,i} \text{ with } F = \begin{bmatrix} f_1 \\ \vdots \\ f_m \end{bmatrix}. \tag{5.7}$$

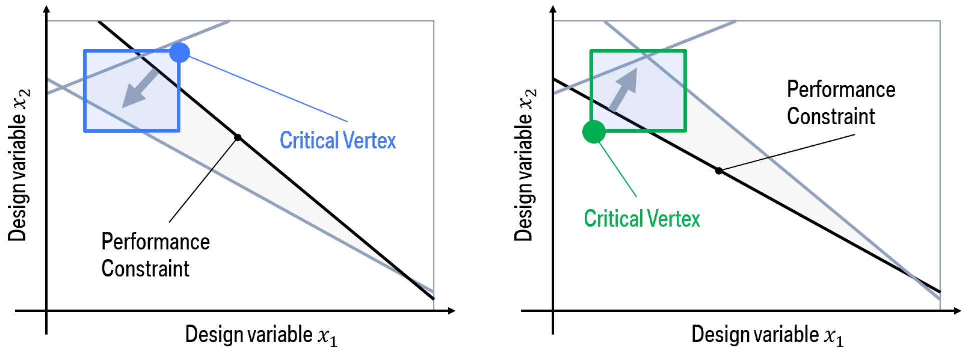


Figure 5.3 Example for VT. If the critical vertex is a good design the entire Solution Space contains only good designs with respect to the considered performance constraint.

According to VT, the critical vertex for the i-th constraint is

$$x_{\text{crit}_j} = x_c + \frac{1}{2}\begin{bmatrix} \text{sign}(f_{i,1})x_{r,1} \\ \vdots \\ \text{sign}(f_{i,d})x_{r,d} \end{bmatrix}. \tag{5.8}$$

The combination of equation (5.7) and (5.8) yields:

$$f_i x_{\text{crit}_i} \leq f_{c,i} + \Delta f_{c,i} \tag{5.9}$$

as the new constraint function. In contrast to equation (5.7), only a single vertex x_{crit_j} and not all designs inside the box $[x_c - \frac{1}{2}x_r;\ x_c + \frac{1}{2}x_r]$ is considered. Equation (5.9) can be further simplified to:

$$f_i x_c \leq f_{c,i} + \Delta f_{c,i} - \frac{1}{2}|f_i|x_r \text{ with } |f_i| = \begin{bmatrix} |f_{i,1}| & \dots & |f_{i,d}| \end{bmatrix}. \tag{5.10}$$

We apply VT to each constraint. Hence, problem statement 12 can be simplified to:

Problem Statement 13.

$$\underset{\Delta f_c, x_c}{\text{minimise}}\, g(\Delta f_c) \tag{5.11a}$$

$$s.t. \quad \Delta f_c \geq 0; \qquad F x_c \leq f_c + \Delta f_c - \frac{1}{2}|F|x_r; \quad x_c \pm \frac{1}{2}x_r \in \Omega_{ds}, \tag{5.11b}$$

with $|F| = [|f_1| \dots |f_m|]^T$.

In the following, we show that Δf_c can be eliminated as a degree of freedom. This will result in a highly non-linear target function but will also reduce the amount of constraints. In order to modify problem statement 13 such that its only degree of freedom is the centre of the box x_c, we introduce the function $\Delta f_{c_{opt}}(x_c) : \mathbb{R}^d \to \mathbb{R}^m$, which depends on the centre of the box x_c and determines the minimum relaxation required for each performance constraint depending on the positioning of the box. Its i-th entry is calculated as follows:

$$\Delta f_{c_{opt},i}(x_c) = \begin{cases} f_i x_c + \frac{1}{2}|f_i| x_r - f_{c,i} & \text{for } f_i x_c + \frac{1}{2}|f_i| x_r - f_{c,i} > 0 \\ 0 & \text{else} \end{cases} \tag{5.12}$$

We eliminate the performance constraint in equation (5.11b) by inserting $f_{c_{opt}}$ into the target function. This yields:

Problem Statement 14.

$$\underset{x_c}{\text{minimise}}\, g(\Delta f_{c_{opt}}(x_c)) \tag{5.13a}$$

$$s.t. \quad x_c \pm \frac{1}{2} x_r \in \Omega_{ds} \; . \tag{5.13b}$$

5.2.4. Computational Effort

In order to compute Optimal Constraint Relaxation for Solution Spaces with linear performance constraints, we can choose whether we want to solve problem statement 13 or 14. With respect to the global optimum they are interchangeable. For this thesis, the target function $g(\Delta f_c)$ is quadratic with respect to its argument Δf_c (see equation (5.4)). Hence, we choose to compute problem statement 13 since it can be solved very efficiently by the interior point algorithm.

For a fixed gap reduction, a mathematically proven upper boundary for the number of Newton iterations is given by $\mathcal{O}(\sqrt{m})$. Although, in most technical applications the number of iterations needed grows even more slowly with $\mathcal{O}(\log m)$. In order to derive the computational effort, this has to be multiplied with the cost of one Newton iteration, which depends on the number of problem dimensions d [5].

5.3. Non-linear Performance Constraints

In this section, we extend the idea of Optimal Constraint Relaxation for Solution Spaces to systems with non-linear performance constraints. Therefore, an iterative algorithm is introduced and reviewed with respect to its computational effort.

5.3.1. Idea

Recall that in order to solve the linear problem statement 12 we apply the VT idea. This idea can be extended to non-linear monotonic performance constraints [20]. Figure 5.4 gives an example of how monotonic constraints look like and connects them with the properties of the resulting complete Solution Space. In order to apply VT to non-linear constraints, these constraints need to be linearised. An algorithm is introduced, which iteratively moves the centre of the box towards an optimal point such that $g(\Delta f_c)$ is minimised. The system is linearised twice in each iteration loop: first to determine the critical vertices, second to linearise the system around the critical vertices. The double linearisation is executed in order to reduce the linearisation error. According to the Taylor series, the linearisation error grows with the distance to the point at which the function was linearised at (see equations (5.14) and (5.16)). We are most interested in the change of the function value close to the critical design. Hence, it is crucial to linearise each constraint $f_i x \leq f_{c,i}$ at the critical vertices x_{crit_i}. The critical vertices are derived by VT (see equation (5.15)) in the linearised system around the centre of the current box x_c:

$$f_i(x) \xrightarrow[\text{at } x_c]{\text{linearised}} \nabla f_i(x_c)(x - x_c) + f(x_c). \tag{5.14}$$

The critical vertices are

$$x_{\text{crit}_i} = x_c + \frac{1}{2} \begin{bmatrix} \text{sign}(\frac{\partial f_i(x)}{\partial x_1}\Big|_{x=x_c}) x_{r,1} \\ \vdots \\ \text{sign}(\frac{\partial f_i(x)}{\partial x_d}\Big|_{x=x_c}) x_{r,d} \end{bmatrix}. \tag{5.15}$$

The linearised constraints are derived with a second linearisation:

$$f_i(x) \xrightarrow[\text{at } x_{\text{crit}_i}]{\text{linearised}} \nabla f_i(x_{\text{crit}_i})(x - x_{\text{crit}_i}) + f(x_{\text{crit}_i}). \tag{5.16}$$

Finally, the problem statement to be solved in each iteration loop is obtained. It is a modified version of the original linear problem statement 13:

Problem Statement 15.

$$\underset{\Delta f_c, x_c}{\text{minimise}}\, g(\Delta f_c) \tag{5.17a}$$

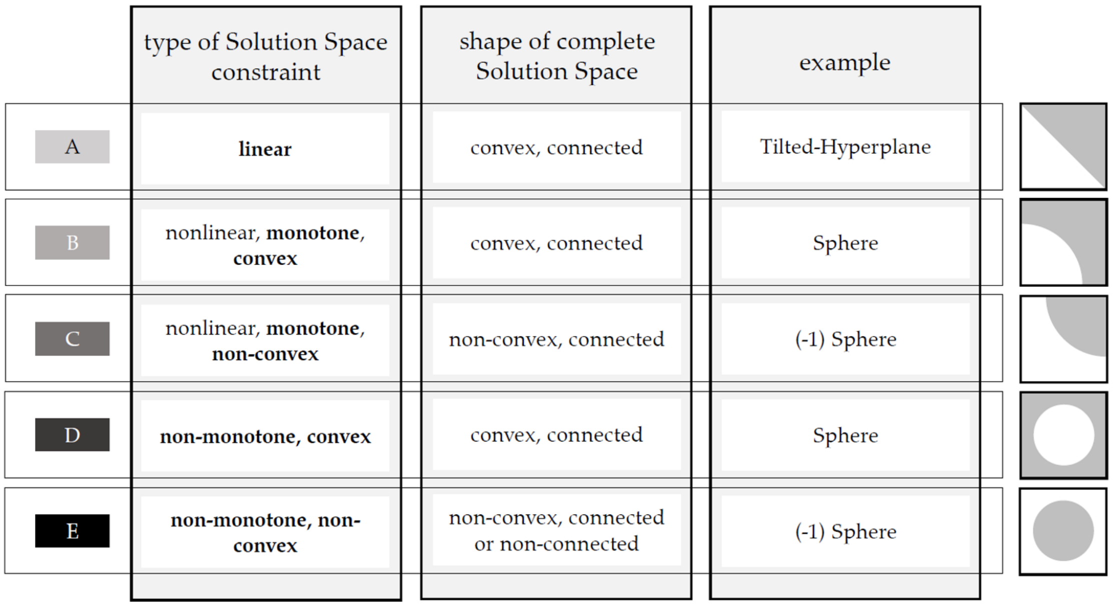
$$s.t.\quad \nabla f_i(x_{\text{crit}_i}) x_c \leq f_{c,i} - f_i(x_{\text{crit}_i}) + \nabla f_i(x_{\text{crit}_i}) x_{\text{crit}_i} + \Delta f_{c,i} - \frac{1}{2}|\nabla f_i(x_{\text{crit}_i})| x_r,$$

$$\text{for } i = 1, \dots, m; \quad \Delta f_c \geq 0; \quad x_c \pm \frac{1}{2} x_r \in \Omega_{ds}. \tag{5.17b}$$

5.3.2. Details

In the following, a definition for monotonic constraints is given and the steps of the Optimal Constraint Relaxation algorithm for Solution Space with non-linear performance constraints (alg. 7) are explained in detail.

A performance function $f(x)$ is monotonically increasing with respect to the design



	type of Solution Space constraint	shape of complete Solution Space	example
A	**linear**	convex, connected	Tilted-Hyperplane
B	nonlinear, **monotone, convex**	convex, connected	Sphere
C	nonlinear, **monotone, non-convex**	non-convex, connected	(-1) Sphere
D	**non-monotone, convex**	convex, connected	Sphere
E	**non-monotone, non-convex**	non-convex, connected or non-connected	(-1) Sphere

Figure 5.4 Type of Solution Space constraints, shape of Solution Spaces and examples according to [20].

variable x_i if it fulfils the following condition:

$$f(\tilde{x}) - f(x) \geq 0 \quad \forall\, \tilde{x}, x,\ \tilde{x}_i > x_i,\ \tilde{x}_j = x_j, \tag{5.18}$$

with $i \in \{1, \dots, m\}$ and $j = \{1, \dots, m\}/\{i\}$. Similarly, a performance function $f(x)$ is monotonically decreasing with respect to the design variable x_i if it fulfils the following condition:

$$f(\tilde{x}) - f(x) \leq 0 \quad \forall\, \tilde{x}, x,\ \tilde{x}_i > x_i,\ \tilde{x}_j = x_j, \tag{5.19}$$

with $i \in \{1, \dots, m\}$ and $j = \{1, \dots, m\}/\{i\}$ [25].

In order to compute Optimal Constraint Relaxation for Solution Space with non-linear monotonic constraints, algorithm 7 is executed in an iterative loop. The loop is terminated if either the solution converges, a maximum number of iterations is reached or a feasible Solution Space without any constraint relaxation is found:

If all constraints are monotonic, we can be sure the Optimal Constraint Relaxation algorithm, introduced in this chapter, converges to the optimal solution. For engineering purposes we can also apply the algorithm to arbitrary non-linear problems. The idea is that if the performance constraints behave monotonically in a certain area around the box, the algorithm converges to a local optimum. In order to optimise the result, the algorithm can be executed several times with different starting points. In addition, a pre-optimisation can be run to compute a single optimal starting point for $x_{c,opt}$. Therefore, the non-linear Optimal Constraint Relaxation problem statement 11 is computed without robustness constraints $x_r = 0$. Then, the Solution Space includes only a single design

```
Algorithm 7: Optimal Constraint Relaxation for Solution Spaces with non-linear
performance constraints
Data: Initial System with performance constraints $f(x) \leq f_c$, design space
      constraints $\Omega \subseteq \Omega_{ds}$, robustness constraints $x^{\mathrm{ub}} - x^{\mathrm{lb}} \geq x_r$ and the target
      function $g(\Delta f_c)$.
Result: Position of the Solution Space and relaxation $\Delta f_c$ optimised with
        respect to the target function $g(\Delta f_c)$.
while $x_c$ changes significantly and $|\Delta f_c| > 0$ and maximum number of iterations
 has not been reached do
    (1) Linearise each performance constraint around the current centre of the box.
    (2) The critical vertices according to VT are determined (see equation (5.10))
     for the linearised system derived in step (1).
    (3) Each constraint is linearised again at its respective critical vertex.
    (4) Problem statement 13 is computed for the linearised system derived in
     step (3).
end
```

$\Omega = x_c$ and a classical single point optimiser such as particle swarm optimisation can be applied [60]. If the additional computational effort is acceptable, instead of checking only critical vertices, each vertex can be checked with respect to performance constraint violation. Hence, step (3) of algorithm 7 is modified such that *each constraint is linearised at* ***each*** *vertex.* This helps to track convex constraints such as the sphere shown in figure 5.4.D. Unfortunately, the computational effort to compute all vertices grows exponentially with the number of dimensions $\mathcal{O}(d^2)$ and hence, is not applicable for high dimensions.

5.3.3. Computational Effort

The total computational effort to compute Optimal Constraint Relaxation for Solution Spaces with non-linear monotonic performance constraints can be estimated by:

$$i \times c_{\mathrm{ps15}} = i \times \left[(2m \times c_{lin}) + c_{\mathrm{ps13}} \right], \tag{5.20}$$

where c_{ps13} and c_{ps15} represent the computational effort required to solve problem statement 13 and 15. c_{lin} represents the computational effort required to linearise the system. i is the number iterations and m the number of constraints.
The total computational effort depends linearly on the number of iterations i needed to compute algorithm 7. In each iteration, the non-linear Optimal Constraint Relaxation problem statement 15 needs to be solved. Therefore, a number of linearisation steps equal to twice the number of constraints m is necessary. Their computational effort lin depends on the system dimensions, on how expensive a single function evaluation is and whether the first derivatives are known explicitly. If the derivatives are unknown, they need to be estimated by finite differences. In addition, the linear Optimal Constraint Relaxation problem statement 13 needs to be solved in each iteration. If the quadratic target function (see equation (5.4)) is used this can be efficiently computed with an

interior point optimiser.

5.4. Determination of the weighting factors

In order to apply Optimal Constraint Relaxation for Solution Spaces, a target function $g(\Delta f_c)$ needs to be defined. As introduced in subsection 5.1.3 for this thesis, we use the following target function:

$$g(\Delta f_c) = \sum_{i=1}^{m} q_i (\Delta f_{c,i})^2, \tag{5.21}$$

where q_i are the weighting factors. In the following, a generic approach, a case-specific approach and an interactive approach to determine the weighting factors are introduced.

5.4.1. Generic weighting factors

Deriving the target function based on generic weighting factors works for any application of Optimal Constraint Relaxation for Solution Spaces. The idea is to model the weighting factors q based on the critical performance values f_c, such that the units of the outputs get eliminated and the relaxation of different performance constraints becomes comparable. For each performance constraint $f_i(x) \leq f_{c,i}$ three different scenarios are possible:

1 The performance $f_i(x)$ has **both an upper and a lower boundary**. Hence, a related constraint $f_j(x) \leq f_{c,j}$ exists, which fulfils the following criteria: $f_{c,j} f_{c,i} < 0$ and $\forall x \in \Omega_{ds}$, $f_j(x) = -f_i(x)$. If that is the case, the weighting factor q_i is determined as follows:

$$q_i = \frac{1}{(f_{c,i} - f_{c,j})^2}. \tag{5.22}$$

2 The performance $f_i(x)$ has **either an upper or a lower boundary**, which means no related constraint $f_j(x) \leq f_{c,j}$ exists, which fulfils the aforementioned conditions and the **critical value** of the constraint **is not zero**: $f_{c,i} \neq 0$. If that is the case, the weighting factor q_i is determined as follows:

$$q_i = \frac{1}{f_{c,i}^2}. \tag{5.23}$$

3 The performance $f_i(x)$ has **either an upper or a lower boundary** and the **critical value** of the constraint **is zero**: $f_{c,i} = 0$. If that is the case, no weighting according to the critical value is possible and q_i is set to 1.

5.4.2. Case-specific weighting factors

As the name suggests, case-specific weighting factors depend on the case and are only applicable if additional information about the performance constraints is available. In this section, an example for case-specific weighting factors is introduced. The example

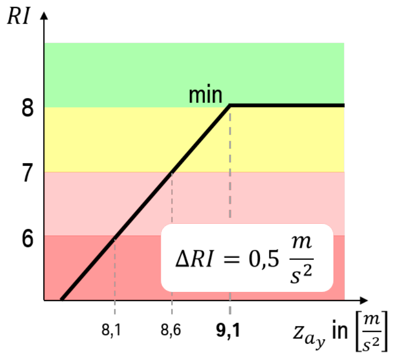


Figure 5.5 Example on how to calculate the Rating Index for vehicle dynamics for the maximum lateral acceleration z_{a_y}.

is applicable to the chassis design problem considered in this thesis. Therefore, we convert the different performances $f_i(x)$ into a uniform scale, which is specific to driving dynamics. The scale describes the Rating Index for vehicle dynamics (RI) and ranges from 1.0 to 10.0. 1.0 describes the worst possible performance and 10.0 the best possible performance. The critical values are designed such that they represent exactly the value eight on the rating scale $\text{RI}(f_{c,i}) = 8$. Hence, each performance that is eight or better fulfils the performance constraint:

$$\text{RI}(f_i(x)) \geq 8.0. \tag{5.24}$$

Since an RI of 8.0 is sufficient, we are not interested in the precise RI value if the performance constraints are fulfilled; we simply set it to 8.0. If the performance constraint is not fulfilled, the value for the RI decreases linearly. The individual gradients $\Delta\text{RI}_i > 0$ are known, hence we can define the RI function as follows:

$$\text{RI}(f_i(x)) = \begin{cases} 8 - \frac{f_i(x) - f_{c,i}}{\Delta\text{RI}_i} & \text{for } f_i(x) > f_{c,i} \\ 8.0 & \text{for } f_i(x) \leq f_{c,i} \end{cases} \tag{5.25}$$

Figure 5.5 shows how the RI is calculated for the constraint with respect to maximum lateral acceleration (see table 3.2). Note that generally the RI is capped at a value of 1.0 and cannot decrease any further. For industrial applications, such low RIs are not interesting anyway due to the fact that a vehicle with an RI value of 5.0 would never be built.

Since the RI system allows us to translate each constraint violation into the same scale it is perfectly suitable to determine the weighting factors for the target function of Optimal Constraint Relaxation for Solution Spaces. The weighting factors are chosen as

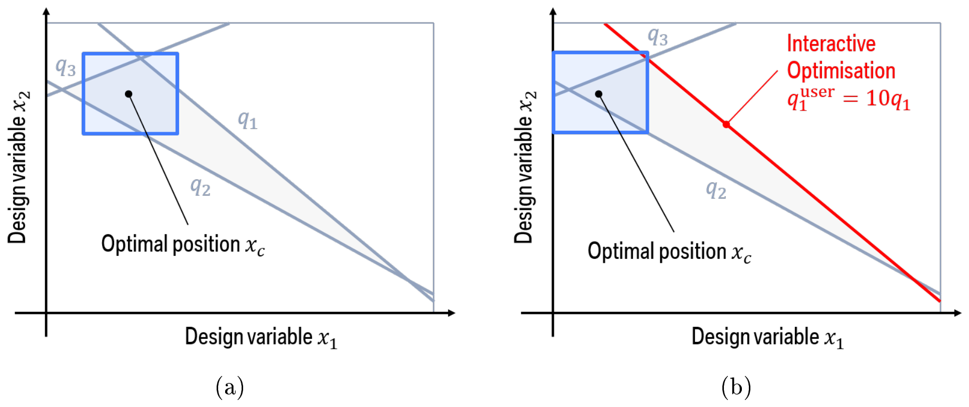


Figure 5.6 Example for interactive optimisation.(a) Each constraint is weighed equally $q_1 = q_2 = q_3$. (b) Constraint one q_1 is more important $q_1 = 10q_2 = 10q_3$.

follows:

$$q_i = \frac{1}{\Delta \mathrm{RI}_i^2}. \tag{5.26}$$

Considering equations (5.25), (5.26) and $\mathrm{RI}(f_i(x)) = \mathrm{RI}(f_{c,i} + \Delta f_{c,i})$ the target function can be simplified to:

$$g(\Delta f_c) = \sum_{i=1}^{m} \left[8 - RI(f_{c,i} + \Delta f_{c,i})\right]^2. \tag{5.27}$$

5.4.3. Interactive Optimisation

Interactive optimisation is often used in industrial applications where an expert is able to evaluate the result of the Optimal Constraint Relaxation. In order to apply interactive optimisation, an initial weighting system has to be chosen e.g. generic or case-specific weighting factors. After calculating the results of Optimal Constraint Relaxation for Solution Spaces with the initial weighting factors, the result is reviewed. In case it is not satisfactory because some constraints have been relaxed by too much, their weighting factors can be increased. A new Optimal Constraint Relaxation for Solution Spaces is executed where it is more costly to relax the chosen constraints. The result is reviewed once more. This way, the weighting factors can be adjusted in an iterative loop. Figure 5.6 shows an example for interactive optimisation. In figure 5.6 (a), each constraint violation is weighed identically. The user decides that constraint one is more important and hence multiplies its respective weighting factor by 10. The new result is shown in figure 5.6b.

Chapter 6

Optimal Constraints Relaxation for Solution-Compensation Spaces

Solution-Compensation Spaces (SCSs) where introduced in chapter 4. They are an evolution of the classical Solution Space approach introduced by Zimmermann & von Hössle [83]. In order to compute Solution-Compensation Spaces, the design variables are divided into a set of early- and a set of late-decision variables. Solution-Compensation Spaces have the property that for all values within the early-decision permissible intervals there exists at least one set of values from the late-decision intervals such that the resulting design fulfils all design goals.

In case the typical Solution-Compensation Space problem statement is infeasible or the acquired permissible intervals for the early-decision variables are not large enough to encompass all uncertainties, Optimal Constraint Relaxation (OCR) for Solution-Compensation Spaces can be applied. In this case, a minimal size for each early-decision permissible interval needs to be defined. The Cartesian product of these intervals represents a box of fixed size. This box exists only in the early-decision parameter dimensions. The goal of Optimal Constraint Relaxation for Solution-Compensation Spaces is to seek a set of relaxed constraints and the position of the box such that for all values within the early-decision permissible intervals there exists at least one set of values within the late-decision variable intervals such that all relaxed design goals are fulfilled. The set of relaxed constraints is optimal with respect to a target function that weighs the relaxation of different constraints and aims to relax each constraint as little as possible. In this chapter, mathematical formulations of the underlying optimisation problem are stated and an approach to compute Optimal Constraint Relaxation for Solution-Compensation Spaces with linear performance constraints is proposed.

6.1. Idea, overview, and problem statement

In the early stages of product development, it is crucial to derive designs, which are sufficiently robust with respect to changes, which are made in the later stages of the development process. Therefore, the size of Solution Spaces is optimised. Unfortunately, in many industrial applications the resulting Solution Spaces either do not exists or are too small. In order to enlarge these Solution Spaces, Solution-Compensation Spaces and Optimal Constraint Relaxation for Solution Space were introduced in chapters 4 and 5. The main advantages of the Solution-Compensation Space method are:

- Enlarged intervals for all early-decision parameters.
- The set of performance constraints does not need to be relaxed.

The main advantages of Optimal Constraint Relaxation for Solution Space are:

- The performance constraints are relaxed optimally with respect to a target function.
- Any Solution Space problem statement with arbitrarily strict performance and robustness constraints becomes feasible.

In this chapter, a new method is introduced, which aims to combine the advantages of both former approaches: Optimal Constraint Relaxation for Solution-Compensation Spaces. The idea is to make any Solution Space problem statement with robustness constraints feasible by taking advantage of the flexibility provided by Solution-Compensation Spaces. In addition, to all the advantages of Optimal Constraint Relaxation for Solution Spaces, the method relaxes the performance constraints by a smaller margin due to the increased flexibility of the early-decision parameters.

6.1.1. Basic idea of Optimal Constraint Relaxation for Solution-Compensation Spaces

In order to illustrate the basic idea of Optimal Constraint Relaxation for Solution-Compensation Spaces, a two-dimensional example problem is shown in figure 6.1. The three straight lines forming the blue triangle represent the constraints. The triangle depicts the area of all good designs where all requirements are satisfied. Inside the complete Solution Space of figure 6.1a a classical box-shaped Solution Space is shown, which is optimised with respect to its volume. More robustness is required with respect to the design variable x_a. In order to increase the interval size for x_a, Ω_b is used as a compensation space. The optimised Solution-Compensation Space is depicted in figure 6.1b. Even though the interval size of x_a is increased significantly, it is still not sufficiently large to account for all uncertainties. Hence, Optimal Constraint Relaxation for Solution Spaces is applied in order to derive an optimal set of relaxed performance constraints such that the robustness requirement with respect to x_a is fulfilled $x_a^{ub} - x_a^{lb} \geq x_{r,a}$. The required box size for x_b is therefore set to zero $x_b^{ub} = x_b^{lb}$. The result is shown in figure 6.1c. Unfortunately, the relaxation of the performance constraints is too big.

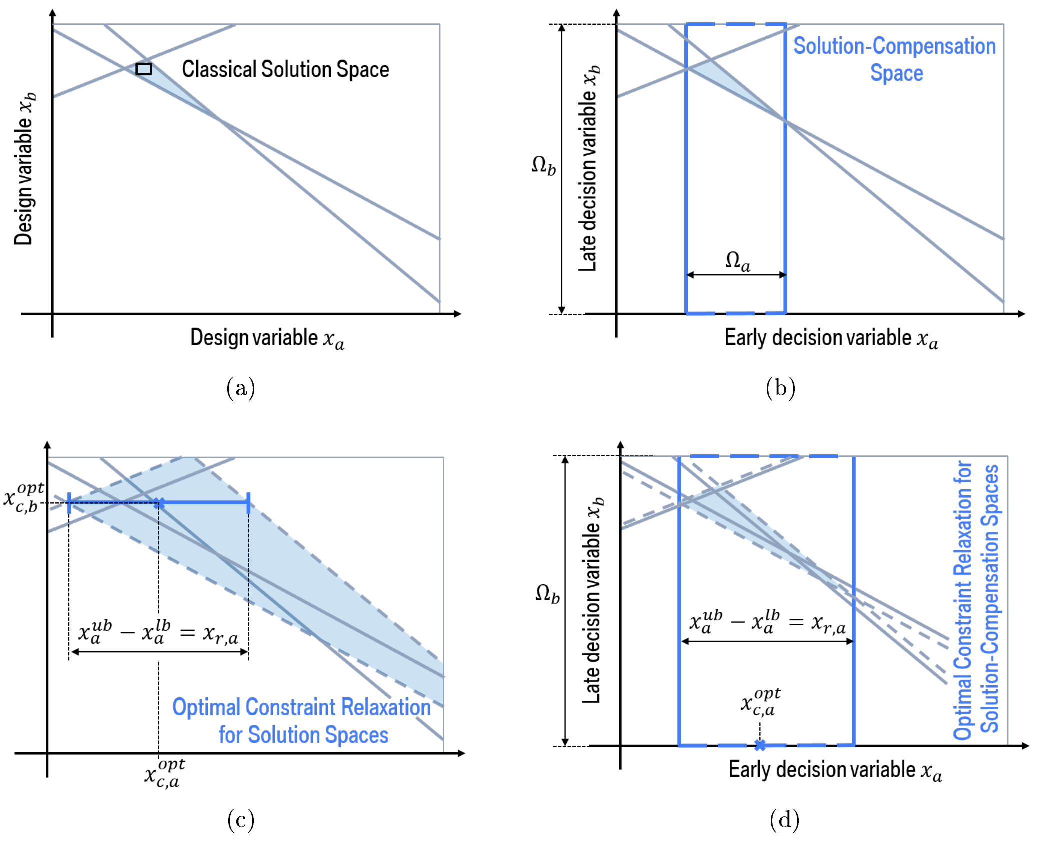

Classical Solution Space
Design variable x_b
Design variable x_a
(a)
Solution-Compensation Space
Ω_b
Late decision variable x_b
Ω_a
Early decision variable x_a
(b)
$x_{c,b}^{opt}$
$x_a^{ub} - x_a^{lb} = x_{r,a}$
Optimal Constraint Relaxation for Solution Spaces
$x_{c,a}^{opt}$
(c)
Ω_b
Late decision variable x_b
$x_a^{ub} - x_a^{lb} = x_{r,a}$
$x_{c,a}^{opt}$
Optimal Constraint Relaxation for Solution-Compensation Spaces
Early decision variable x_a
(d)

Figure 6.1 (a) An optimised box-shaped Solution Space (black). (b) An optimised SCS (blue box). (c) An optimised set of relaxed performance constraints (grey dashed lines) derived by OCR for Solution Spaces and the optimised box-shaped Solution Space (blue line). (d) An optimised set of relaxed performance constraints (grey dashed lines) derived by OCR for SCSs and the optimised SCS (blue box).

Figure 6.1d shows the result of Optimal Constraint Relaxation for Solution-Compensation Space. The variable x_b is defined as a late-decision variable. In addition, the performance constraints are relaxed by Δf_c such that an Solution-Compensation Space with the required interval size for x_a becomes feasible. Therefore, compared to the classical Optimal Constraint Relaxation for Solution Spaces approach a far smaller relaxation of the performance constraints is needed. $x_{c,a}^{\text{opt}}$ is the optimised position of the final Solution-Compensation Space, which fulfils the robustness requirements $x_a^{ub} - x_a^{lb} \geq x_{r,a}$ as well as the relaxed performance requirements $Fx \leq f_c + \Delta f_c$.

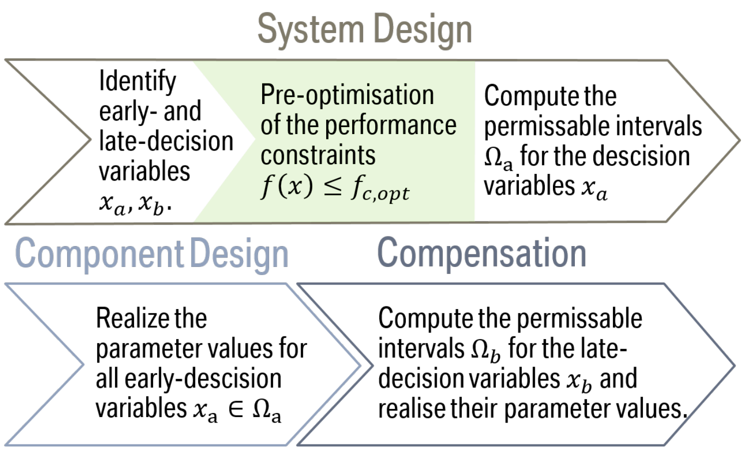

System Design
Identify early- and late-decision variables x_a, x_b.
Pre-optimisation of the performance constraints $f(x) \leq f_{c,opt}$
Compute the permissable intervals Ω_a for the descision variables x_a
Component Design
Realize the parameter values for all early-descision variables $x_a \in \Omega_a$
Compensation
Compute the permissable intervals Ω_b for the late-decision variables x_b and realise their parameter values.

Figure 6.2 Approach using system and component design of classical Solution Spaces with the addition of a pre-optimisation step for infeasible problem statements.

6.1.2. Design Process with Optimal Constraint Relaxation for Solution-Compensation Spaces

As introduced in subsection 4.1.2, the classical Solution Space approach consists of a system design phase and a component design phase. During the system design phase permissible intervals are derived. In the component design phase the value for each variable is specified. Similarly to the design process with Solution-Compensation Spaces, Optimal Constraint Relaxation for Solution-Compensation Spaces adds a compensation phase in which the permissible intervals for all late-decision variables x_b are computed and a final design is realised. As shown in figure 6.2, Optimal Constraint Relaxation for Solution-Compensation Spaces requires a pre-optimisation of the performance constraints during the system design phase. After the early- and the late-decision variables are identified, an algorithm is executed, which determines whether an Solution-Compensation Space, which fulfils all robustness requirements can be found. If not, the constraints are relaxed until the problem becomes feasible. For this, the SINF is minimised.

6.1.3. General problem statement

The initial problem statement, which shall be solved is the Solution-Compensation Space problem statement (see equation (4.2)) with additional robustness constraints for the early-decision variables. Robustness constraints refer to the permissible interval sizes for the design variables: $x_a^{ub} - x_a^{lb} \geq x_{r,a}$. This leads to the following problem statement:

Problem Statement 16.

$$\underset{\Omega_a \subseteq \Omega_{\mathrm{ds},a}}{\text{maximise}}\, \mu(\Omega_a) \tag{6.1a}$$

$$s.t. \quad \forall x_a \in \Omega_a,\ \exists x_b \in \Omega_{\mathrm{ds},b},\ f(x_a, x_b) \leq f_c; x_a^{ub} - x_a^{lb} \geq x_{r,a}\,. \tag{6.1b}$$

As visualised in figure 6.1d, this problem statement can be infeasible, in which case no result is obtained. Similarly to Optimal Constraint Relaxation for Solution-Compensation Spaces, the idea is to reformulate problem statement 17 such that it is feasible for any arbitrarily strict requirements. Instead of optimising the size of an Solution Space, the goal is to optimise an Solution Space, which fulfils the robustness requirements exactly $x_a^{ub} - x_a^{lb} = x_{r,a}$. This leads to an Solution-Compensation Space of fixed size $\Omega_a = [x_{c,a} - \frac{1}{2}x_r; x_{c,a} + \frac{1}{2}x_r]$ and $\Omega_b = \Omega_{ds,b}$, $x_{c,a}$ is the centre of the Solution-Compensation Space in the early-decision dimensions. In case no such Solution-Compensation Space can be found, the performance constraints are relaxed. The new objective function minimises the amount of relaxation. The degrees of freedom (dofs) are the position of the Solution-Compensation Space centre in the early-decision dimensions $x_{c,a}$ as well as the relaxation of the performance constraints Δf_c. This leads to the following problem statement:

Problem Statement 17.

$$\underset{\Delta f_c, x_{c,a}}{\text{minimise}}\, g(\Delta f_c) \tag{6.2a}$$

$$s.t. \quad \Delta f_c \geq 0; \qquad \forall x_a \in \Omega_a \ \exists x_b \in \Omega_b, f(x_a, x_b) \leq f_c + \Delta f_c; \quad \Omega_a \subseteq \Omega_{ds,a}\ . \tag{6.2b}$$

$$\text{with } \Omega_a = [x_{c,a} - \frac{1}{2}x_{r,a};\ x_{c,a} + \frac{1}{2}x_{r,a}]\ .$$

Similarly to Optimal Constraint Relaxation for Solution Space, the target function $g(\Delta f_c)$ has to have the properties defined in subsection 5.1.3.

6.2. Linear Performance Constraints

In this thesis we introduce an algorithm to compute Optimal Constraint Relaxation for Solution-Compensation Spaces in systems with linear (or linearised) performance constraints. Linear constraints are defined in equation (4.3). Similarly to problem statement 17, an optimal set of relaxed constraints as well as an optimal position for the centre of the Solution-Compensation Space is sought. Since the late-decision parameter are always bounded by the design space $x_b \in \Omega_{ds,b}$, the optimal position is only computed for the early-decision parameters. Hence, the dof is $x_{c,a}$ rather than x_c:

Problem Statement 18.

$$\underset{\Delta f_c, x_{c,a}}{\text{minimise}}\, g(\Delta f_c) \tag{6.3a}$$

$$s.t. \quad \Delta f_c \geq 0; \qquad \forall x_a \in \Omega_a \ \exists x_b \in \Omega_b, Ax_a + Bx_b \leq f_c + \Delta f_c; \quad \Omega_a \subseteq \Omega_{ds,a}\ . \tag{6.3b}$$

$$\text{with } \Omega_a = [x_{c,a} - \frac{1}{2}x_{r,a};\ x_{c,a} + \frac{1}{2}x_{r,a}]\ .$$

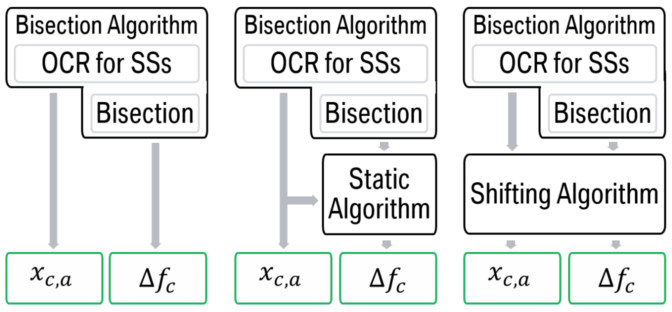


Figure 6.3 Overview of the algorithms to compute Optimal Constraint Relaxation for SCSs

In this section, the idea, the notation, and the computational effort for three different algorithms to compute Optimal Constraint Relaxation for Solution-Compensation Spaces are introduced. Each of them solves problem statement 18. Due to its efficiency, the Bisection Algorithm introduced in subsection 6.2.1 is used as a pre-optimisation for the Static and the Shifting Algorithm introduced in subsections 6.2.2 and 6.2.3.

6.2.1. Bisection Algorithm to compute Optimal Constraint Relaxation for Solution-Compensation Spaces

Idea

As indicated in figure 6.3, the Bisection Algorithm to compute Optimal Constraint Relaxation for Solution-Compensation Spaces consists of two steps. The idea is that in the first step we find a feasible set of relaxed performance constraints in the original non-projected system (Step (1) of algorithm 8). In the second step bisection is applied in order to iteratively scale the relaxed set to a point where it is only just sufficient in the projected system (Steps (2)-(5) of algorithm 8).

Details

In the following, the steps of the Bisection Algorithm to compute Optimal Constraint Relaxation for Solution-Compensation Spaces with linear performance constraints (see algorithm 8) are explained in detail.

Step (1). We know that with the original set of performance constraints problem statement 16 is infeasible, therefore Optimal Constraint Relaxation is needed. Problem statement 16 becomes feasible when a relaxed set of performance constraints is applied. In order to derive the relaxed set, Optimal Constraint Relaxation for Solution Spaces (see Section 5.2.1) is used to solve the following modified version of problem statement 13:

Problem Statement 19.

$$\underset{\Delta f_c, x_{c,a}, x_{c,b}}{\text{minimise}} \; g(\Delta f_c) \tag{6.4a}$$

$$s.t. \quad \Delta f_c \geq 0; \quad Ax_{c,a} + Bx_{c,b} \leq f_c + \Delta f_c - \frac{1}{2}|A|x_{r,a};$$

$$x_{c,a} \pm \frac{1}{2}x_{r,a} \in \Omega_{ds,a}; \quad x_{c,b} \in \Omega_{ds,b}, \text{ with } |A| = \begin{bmatrix} |a_{1,1}| & \dots & |a_{1,p}| \\ \vdots & \ddots & \vdots \\ |a_{m,1}| & \dots & |a_{m,p}| \end{bmatrix}. \tag{6.4b}$$

The goal is to derive Δf_c and $x_{c,a}$ for a box, which fulfils the robustness constraints with respect to the early-decision parameter $x_a^{ub} - x_a^{lb} \geq x_{r,a}$ and has an interval size of zero for all late-decision variables. This set of relaxed performance constraints was optimised with respect to problem statement 11 where a box for a specific value for each late-decision variable is computed. Hence, we can be sure that this set of relaxed constraints will also fulfil the constraints of problem statement 17. Since for problem statement 17 the existence of any feasible value for the late-decision variables is sufficient.

Step (2) - (5). Now that we found a set of constraints, which is feasible, we can try to improve that set, with respect to the target function $g(\Delta f_c)$. Therefore, we apply the Solution-Compensation Space method for linear constraints, which is introduced in section 4.2. We project the relaxed performance constraint by applying Fourier-Motzkin Elimination:

$$\forall x_a \in \Omega_a \exists x_b \in \Omega_b, Ax_a + Bx_b \leq f_c + \Delta f_c^i \xrightarrow{\text{FME}} \forall x_a \in \Omega_a, \tilde{A}^i x_a \leq \tilde{f}_c^i. \tag{6.5}$$

Unfortunately, the projected system refers to a specific Δf_c. If Δf_c changes FME needs to be applied again in order to determine a new projection. Hence, the index i is introduced. It indicates that $\tilde{A}^i$ and $\tilde{f}_c^i$ refer to Δf_c^i. With VT and linear constraints we are able to check whether problem statement 17 is feasible by examining the projected system:

$$x_a \in \Omega_{ds,a}, \tilde{A}^i x_a \leq \tilde{f}_c^i - \frac{1}{2}|\tilde{A}^i|x_{r,a} \tag{6.6}$$

If a solution for x_a exists, we know that the problem is feasible with a performance constraint relaxation of Δf_c^i. This idea can be applied in order to start an iterative procedure to determine the optimal relaxation. For performance reasons we do not change the entries of Δf_c arbitrarily. Instead we scale the entire vector by multiplying it with a scalar value $t \in \mathbb{R}$. This lets us optimise t via Bisection [48]. We know that if a certain t is feasible, the relaxation and hence all values for t that are higher are also feasible. If a certain t is not feasible, the relaxation is insufficient and hence all values that are lower are also not feasible. Figure 6.4 visualises how the optimal scaling factor for constraint relaxation t is derived. The two initial sets of constraints are shown in red and blue. The non-relaxed one $t = 0$ (red) is infeasible while the fully relaxed one $t = 1$ (blue) is feasible. We always test the middle point of the current interval. Hence, the first scaling factor, which is tested with respect to feasibility is $t = 0.5$. In the example

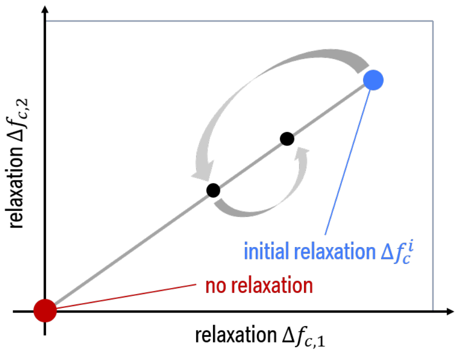

relaxation $\Delta f_{c,2}$
initial relaxation Δf_c^i
no relaxation
relaxation $\Delta f_{c,1}$

Figure 6.4 Example for the Bisection Algorithm.

shown $t = 0.5$ is infeasible. The next scaling factor tested is $t = 0.75$. This iterative process is executed a predefined number of times n. The upper boundary of the final interval (which is always feasible) is then taken as the final result.

Computational Effort

In order to estimate the computational effort of the Bisection Algorithm to compute Optimal Constraint Relaxation for Solution-Compensation Spaces (see algorithm 8) the different steps have to be considered. In Step (1) Optimal Constraint Relaxation for Solution Spaces with linear constraints is computed. Therefore, a quadratic programming problem with linear constraints has to be solved. As explained in section 5.2.1, the computational effort for this is low and grows with $\mathcal{O}(\sqrt{m})$. The feasibility check in Step (5) is computationally cheap as well, it can be implemented with just summation and multiplication operations. For the Bisection Algorithm the computationally dominant part is step (4) where the FME is computed. As explained in section 4.2.3, the FME is only recommended in combination with RR, in which case its computational effort can be estimated by $q \times m \times LP(d, s)$. Where q is the number of projected dimensions, m is the total number of performance constraints and s is the number of non-redundant performance constraints. $LP(d, s)$ represents the computational effort needed to solve an LPP of size d times s. Since the FME is computed once in each iteration, the computational effort for the Bisection Algorithm can be estimated by

$$n \times q \times m \times LP(d, s). \tag{6.7}$$

6.2.2. Static Algorithm to compute Optimal Constraint Relaxation for Solution-Compensation Spaces

Idea

As indicated in figure 6.3, the Static Algorithm to compute Optimal Constraint Relaxation for Solution-Compensation Spaces uses the very efficient Bisection Algorithm as a

Algorithm 8: Bisection Algorithm to compute OCR for Solution-Compensation Spaces with linear performance constraints

```
Data: Initial System with performance constraints $f(x) \leq f_c$, design space
      constraints $\Omega \subseteq \Omega_{ds}$, early- and late-decision variables $x = [x_a; x_b]$,
      robustness constraints for all early-decision variables $x_a^{\text{ub}} - x_a^{\text{lb}} \geq x_{r,a}$ and
      the target function $g(\Delta f_c)$.
Result: Position of the Solution Space $x_c$ and relaxation $\Delta f_c$ optimised with
        respect to the target function $g(\Delta f_c)$.
begin
    (1) Apply Optimal Constraint Relaxation for Solution Spaces to determine a
     $\Delta f_c^i$ optimised with respect to $g(\Delta f_c)$ for an Solution Space of size
     $\Omega = \Omega_a \times \Omega_b$, with $\Omega_a = [x_{c,a} - \frac{1}{2}x_{r,a}; x_{c,a} + \frac{1}{2}x_{r,a}]$ and $\Omega_b = x_{c,b}$.
    (2 )Set current best value $t = 1$, current test value $t_{\text{test}} = 1$, counter $j = 0$,
     current test value on feasible $k = 1$ (1: feasible/ 0: infeasible) and define
     number of iterations $n$.
    while Pre-defined number of iterations is not reached $j < n$ do
        Count the number of iteration steps: $j = j + 1$
        if Current test value is feasible $k == 1$ then
            (3) Modify the current test value for the scaling factor: $t_{\text{test}} = t_{\text{test}} - \frac{1}{2}^j$
        else
            (3) Modify the current test value for the scaling factor: $t_{\text{test}} = t_{\text{test}} + \frac{1}{2}^j$
        end
        (4) Apply FME to project the constraints relaxed by $t_{\text{test}}\Delta f_c$ (see equation
         (6.5)).
        if Projected system is feasible (see equation (6.6)) then
            (5) $k = 1$ and the best value is updated $t = t_{\text{test}}$.
        else
            (5) Projected system is infeasible $k = 0$.
        end
    end
end
```

pre-optimisation. The Static Algorithm got its name because it assumes a fixed position for the centre of the Solution Space in the early-decision variable space $x_{c,a}$. Therefore, $x_{c,a}$ is provided by the Bisection Algorithm. Due to the nature of the Bisection Algorithm it can only modify the scaling factor for the initial constraints relaxation Δf_c^i and cannot change the direction of the vector (see figure 6.4). The Static Algorithm explores different directions for Δf_c to further optimise the set of relaxed constraints with respect to the target function $g(\Delta f_c)$. As shown in figure 6.5(a), the information gained by the Bisection Algorithm can be used to significantly reduce the size of the search area for Δf_c.

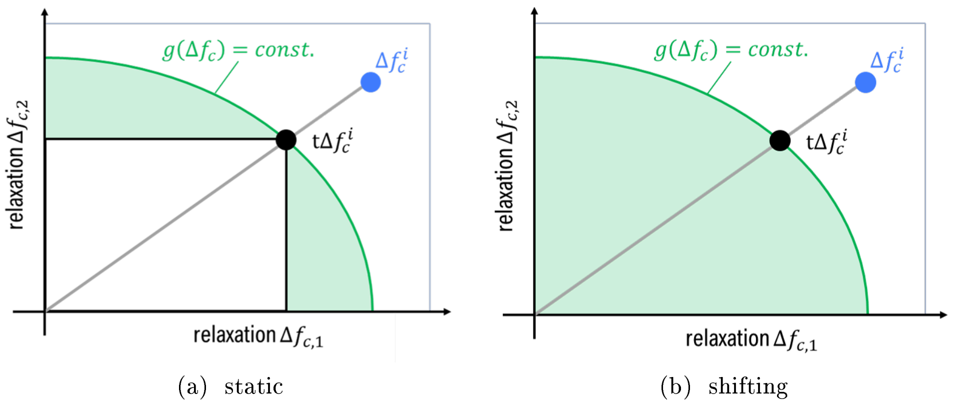


Figure 6.5 Example for the search area (green) of the Static and the Shifting Algorithm with respect to Δf_c.

Details

The problem statement, which is solved by the Static Algorithm relates to problem statement 19 with the premise that $x_{c,a}$ and $x_{c,b}$ are given and that two additional constraint are introduced to reduce the search area:

Problem Statement 20.

$$\underset{\Delta f_c}{\text{minimise}}\, g(\Delta f_c) \tag{6.8a}$$

$$s.t. \quad \Delta f_c \geq 0; \quad \exists x_b \in \Omega_{ds,b}, Ax_{c,a} + Bx_b \leq f_c + \Delta f_c - \frac{1}{2}|A|x_{r,a};$$

$$g(\Delta f_c) \leq g(t\Delta f_c^i); \quad \Delta f_{c,j} \geq t\Delta f_{c,j}^j, \text{for } j \in [1, ..., m]. \tag{6.8b}$$

The Bisection Algorithm result is used as starting point for the optimisation $\Delta f_c^{\text{start}} = t\Delta f_c^i$. The constraint $g(\Delta f_c) \leq g(t\Delta f_c^i)$ ensures that the area where the result is worse than the initial relaxation given by the bisection is not searched. For the target function used in this thesis $g(\Delta f_c) = \sum_{i=1}^{m} q_i(\Delta f_{c,i})^2$, this boundary is visualised in figure 6.5(a) by the green sphere. Note that this constraint is not necessary when gradient-based optimisation is applied, since the relaxation cannot become worse with respect to the target function anyway.

The constraint $\Delta f_{c,j} \geq t\Delta f_{c,j}^i$, for $j \in [1, ..., m]$ ensures that the area is searched where all entries of Δf_c are smaller than the entries of $t\Delta f_c^i$ is not searched. These relaxations cannot be feasible since the Bisection Algorithm would have found them otherwise. This boundary is visualised in figure 6.5(a) by the black box. Only the green area is searched in order to improve Δf_c.

In order to optimise Δf_c a gradient-based method such as interior point optimisation is recommended [80]. It can deal with non-linear constraints and is very efficient for quadratic target functions. The performance constraint $Ax_{c,a} + Bx_{c,b} \leq f_c + \Delta f_c - \frac{1}{2}|A|x_{r,a}$ needs to be checked via projection with FME, similar to steps (4) and (5) of the Bisection Algorithm 8. The violation c is then evaluated based on a modified version of equation (6.6):

$$c = -\tilde{f}_c^i + \frac{1}{2}|\tilde{A}^i|x_{r,a} + \tilde{A}^i x_{c,a}, \quad [x_{c,a} - \frac{1}{2}x_{r,a}; x_{c,a} + \frac{1}{2}x_{r,a}] \in \Omega_{ds,a}. \tag{6.9}$$

If $c \leq 0$ the constraint is fulfilled. If the projection yields no feasible result the constraint is maximally violated. In this case, c is set to an arbitrarily high number in our MATLAB© implementation. While the gradient of the quadratic target function can easily be derived in explicit form, the gradient of the constraints needs to be evaluated via finite differences. It cannot be determined explicitly since the parameters $\tilde{A}^i$ and $\tilde{f}_c^i$ depend on the current position of the Solution Space and can only be determined via projection.

Computational Effort

Similarly to the Bisection Algorithm, the computational effort is dominated by the LPPs computed during the FME projections. Since the Interior Point Algorithm usually has no predefined number of iterations it is hard to estimate the total computational effort. In each iteration the finite difference method is applied, which depending on whether the forward or the central method is applied needs an amount of function evaluations equal to the number of dimensions of the target function m or twice that amount $2m$. Usually the central method is recommended in order to get a better estimate of the gradient [79]. In addition, at least one new design point is evaluated. Sometimes multiple new design points are evaluated instead, in order to adapt the step length. Assuming n iterations are required and the central method is applied, the computational effort can be estimated by:

$$n \times (2m + 1) \times q \times m \times LP(d, s). \tag{6.10}$$

6.2.3. Shifting Algorithm to compute Optimal Constraint Relaxation for Solution-Compensation Spaces

Idea

Similarly to the Static Algorithm, the Shifting Algorithm to compute Optimal Constraint Relaxation for Solution-Compensation Spaces uses the Bisection Algorithm as a pre-optimisation (see figure 6.3). The Shifting Algorithm got its name because the position of the Solution Space centre in the early-decision variable space $x_{c,a}$ is an optimisation parameter; the position of the Solution Space is shifted during the optimisation. The Shifting Algorithm explores different directions for Δf_c while optimising the position of the Solution Space centre $x_{c,a}$ with respect to the target function $g(\Delta f_c)$. The starting value for both the box centre $x_{c,a}$ and the relaxation Δf_c is provided by the Bisection

Algorithm. As shown in figure 6.5(b), the information gained by the Bisection Algorithm can be used to reduce the size of the search area for Δf_c. Unlike the Static Algorithm no area inside the sphere can be excluded, since the $t\Delta f_c^i$ optimised by the Bisection Algorithm is only valid for that specific centre.

Details

The problem statement, which is solved by the Static Algorithm relates to problem statement 19 with an additional constraint, which is introduced to reduce the search area: $g(\Delta f_c) \leq g(t\Delta f_c^i)$. Note that for high-dimensional problems it can be beneficial to apply the Static Algorithm as an additional pre-optimisation algorithm to further reduce the search area $g(\Delta f_c) \leq g(\Delta f_c^{\text{static Algorithm}})$. In order to solve the optimisation problem for the Shifting Algorithm, gradient-based optimisation methods, such as interior point, are recommended [80]. Since the target function only depends on Δf_c, the gradient of the Solution Space position with respect to the target function is always zero. But according to equation (6.9) the position of the Solution Space $x_{c,a}$ influences whether the constraints function is fulfilled or not. Similarly to the Static Algorithm, the gradient of the constraint function is evaluated via finite differences. As a starting point for the optimisation the results of the Bisection Algorithm $x_{c,a} = x_{c,a}^i$ and $\Delta f_c^{\text{start}} = t\Delta f_c^i$ are used (if the Static Algorithm was used as a pre-optimiser, its results should be used as a starting point instead).

Computational Effort

Again, the computational effort is dominated by the LPPs computed during the projections done via FME. In order to estimate the computational effort a similar logic can be applied as with the Static Algorithm. The two main differences are:

- In order to compute the finite differences more designs need to be evaluated since the position of the Solution Space $x_{c,a}$ is an additional optimisation parameter. It adds an amount of dofs equal the number of early-decision variable dimensions p. Hence, for the central finite difference methods $2m + 2p + 1$ designs are evaluated in each iteration step of the Shifting Algorithm.
- As visualised in figure 6.5 the search area for Δf_c cannot be reduced within the sphere. This usually means that more iteration steps $n + \Delta n$ are needed for the optimisation algorithm to converge.

With these modifications of equation (6.10) the computational effort can be estimated by:

$$(n + \Delta n) \times (2m + 2p + 1) \times q \times m \times LP(d, s). \tag{6.11}$$

Chapter 7

Application and Comparison

In the following, the approaches proposed in this thesis are applied to the chassis design problem to demonstrate their applicability and to point out the differences between them, therefore the chassis design problem is described in detail. Based on the results from the application, the different methods are compared with respect to the size of the resulting Solution Spaces, the runtime, the accuracy, the fulfilment of the robustness constraints, the amount of relaxation required with respect to the performance constraints, and their applicability to non-linear problem statements.

7.1. Exemplary chassis design problem

The chassis design problem considered in this thesis was briefly introduced in section 3.1 as motivation for the aims and objectives. It is a typical engineering problem in the early stage of vehicle design. We assume that conceptional decisions with respect to the structure of the vehicle have already been made. Therefore, the genes of the car such as mass, rear axle load, and the height of the centre of gravity are fixed. Now chassis components need to be designed so that customer relevant properties with respect to vehicle dynamics are optimised. The chassis design dynamics problem is formulated in accordance with [74]. In the following, details about the design variables, the performance measures, their physical relations, the physical simulation model as well as the response-surface meta-model are described.

7.1.1. Design variables and vehicle parameters

The input space is created by eight design variables, which are listed in table 3.1. These variables refer to the tyres, suspension, bump-stop, and anti-roll bar, which all have a significant influence on the driving dynamics behaviour of a vehicle. The design variables μ_{max}, c_{arb}, and c_{bs} are directly linked to the component properties of the tyres, anti-roll bars, and bump stops. An axle including these components is shown in figure 7.1. The

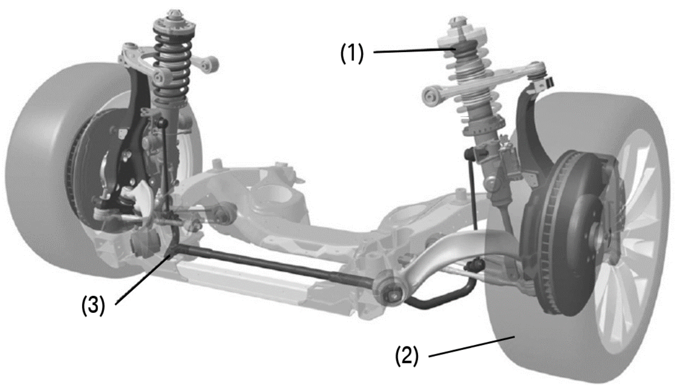


Figure 7.1 Front axle of a passenger vehicle [18] with the chassis components bump stop (1), tyre(2) and anti-roll bar (3)

remaining design variable h_{Ro} (roll centre height) depends on the roll centre (Ro), which is the current pole around which the vehicle body rotates relative to the wheel contact points without any elasticity due to kinematic laws. The front and rear axle each have one roll centre. When stationary, the roll centre is located in the transverse axis plane. Figure 7.2 visualises the roll centre height. For the construction of the roll centre Ro, the rotational poles of the wheels in relation to the road surface (wheel contact points W) and the turning points of the wheels in relation to the vehicle (transverse poles P) are used. The roll centre Ro is determined by the intersection of the two straight lines through W and P for the left and right half of the car. Its distance to the ground is

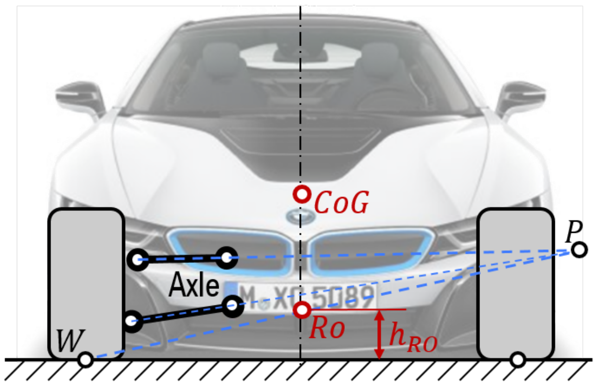


Figure 7.2 Centre of Gravity *CoG* and construction of the rolling centre *Ro* of a double wishbone axle when stationary.

the roll centre height h_{Ro} [71]. Tyres and axles usually have more design variables then just the maximum friction coefficient and the roll centre height. However, in this thesis, in order to keep this example design problem comprehensible, those design variables are kept fixed.

7.1.2. Performance measures

Three standardised driving manoeuvres, which allow for an objective and reproducible assessment of the performance of the vehicle, are regarded here:

- Quasi-Steady State Cornering (QSSC), see [35]: the vehicle follows a circular trajectory with a specified radius of 105 m while the velocity is increased so slowly, such that it can be estimated as being constant at any time step. The velocity is increased up to a point where the vehicle can no longer follow its specified trajectory.
- Ramp Steering (RAST): the vehicle maintains a constant velocity while cornering with an increasing steer angle until a certain lateral acceleration is reached.
- Sine with Dwell (SWD), see [34]: the vehicle performs a lane change manoeuvre with maximal lateral acceleration.

Different vehicle designs are simulated and assessed with respect to customer relevant properties, which are objectified by the performance measures $f(x)$ listed in table 3.2. All design goals are met when the performance measures are between their lower and upper bound, i.e., $f(x) \leq f_c$.

The methods considered in this thesis can be divided into two groups: computational algorithms for linear $Fx \leq f_c$ and non-linear performance constraints $f(x) \leq f_c$. In order to be able to apply both types of methods, a linearised function $f_{\text{lin}}(x) = Ax + b$ is considered in addition to the original non-linear vehicle performance measure $f(x)$.

Therefore, the original system output is linearised by a stepwise regression algorithm [14]. 10, 000 samples have been evaluated to derive the linear model. The coefficients of determination R^2 [50] for the linearised models are shown in table 7.1. They indicate how well each of the outputs can be linearised. Since the R^2 values are all close to a value of 1.0 it is assumed that the linearised models approximate the original models to a sufficient extent.

z_α	z_{a_y}	z_{F_z}	$z_{z_{RA}}$	$z_{z_{FA}}$	z_Φ	z_C	z_δ
0.9868	0.9347	0.9496	0.9676	0.9467	0.9855	0.9995	0.9947

Table 7.1 Coefficient of determination R^2 for the stepwise linearised models of the individual performance constraints. $N = 10,000$ samples have been used as training data for each of the models.

7.1.3. Physical relations between design variables and performance measures

In this section, the influence of the design variables (see table 3.1) on the performance measures (see table 3.2) is analysed. The qualitative influence of the design variables on the outputs is summarised in table 7.2. In order to elucidate these relations, the design variables are divided into two groups:

- Variables that influence the grip: The maximum friction coefficient μ_{max} is a measure for the force transmission between the tyre and the street. The friction coefficient is almost constant for low loads on the tyres. At higher loads, a lateral force saturation occurs and the lateral force decreases [32]. Hence, a high maximum friction coefficient in transversal direction $\mu_{max,Y}$ leads to a high maximum lateral acceleration z_{a_y} as well as a high maximum steering angle factor z_δ of the vehicle. Concurrently, the minimal vertical tyre force z_{F_z}, which is needed to keep the vehicle stable, decreases. The maximum friction coefficient in longitudinal direction $\mu_{max,X}$ has a similar effect on these performance measures but has a smaller impact. The self-steering gradient z_α is only influenced by $\mu_{max,Y}$. A higher grip in transversal direction leads to a lower self-steering gradient as well as a lower vertical displacement z_z of the front/rear part of the car.

- Variables that influence rolling behaviour: The roll centre height h_{Ro} influences the roll moment ratio while cornering. A high roll centre leads to jacking. The vehicle bends less on the outside than it bends on the inside, i.e. the centre of gravity is raised and thus the rolling moment is increased unfavourably. Changing the height of the rolling centre reduces this asymmetry [71]. The stiffness of the anti-roll bars c_{arb} as well as the stiffness of the bump stops c_{bs} influence the vertical stiffness of the respective axle. A higher vertical stiffness of the front axle compared to the rear axle leads to a higher roll moment ratio and vice versa. A higher roll moment ratio results in an increased self-steering gradient z_α, maximum steering angle factor z_δ, and vertical displacement of the front part of the car $z_{z_{\mathrm{FA}}}$, while the maximum lateral acceleration z_{a_y}, the minimum vertical tyre forces z_{F_z} as well as the vertical displacement $z_{z_{\mathrm{RA}}}$ of the rear part of the car decrease. Increasing the overall vertical stiffness leads to a lower roll angle z_Φ while leading to a higher amplitude of the frequency response when passing a one-sided road bump z_C since forces are carried over to the other side of the axle more easily.

	z_α	z_{a_y}	z_{F_z}	$z_{z_{FA}}$	$z_{z_{RA}}$	z_Φ	z_C	z_δ
$\mu_{\max,X}$	0	+	-	-	-	0	0	+
$\mu_{\max,Y}$	-	+	-	-	-	0	0	+
$h_{Ro,\mathrm{RA}}$	-	+	+	-	+	-	-	-
$h_{Ro,\mathrm{FA}}$	+	-	-	+	-	-	-	+
$c_{\mathrm{arb,RA}}$	-	+	+	-	-	-	+	-
$c_{\mathrm{arb,FA}}$	+	-	-	+	-	-	+	+
$c_{\mathrm{bs,RA}}$	-	+	+	-	+	-	+	-
$c_{\mathrm{bs,FA}}$	+	-	-	+	-	-	+	+

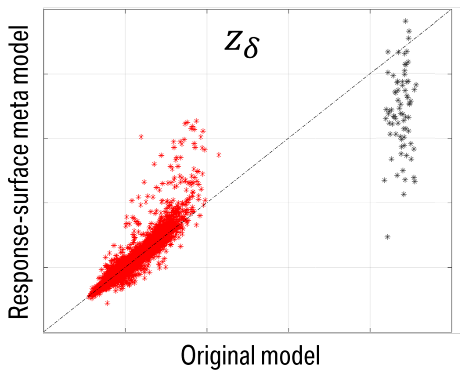
Table 7.2 Changes of the performance measures with respect to the design variables. +/−: the performance measure increases/decreases if the respective design variable increases. 0: the performance measure is not influenced by the respective design variable

7.1.4. Physical simulation model and Response-Surface Metamodel

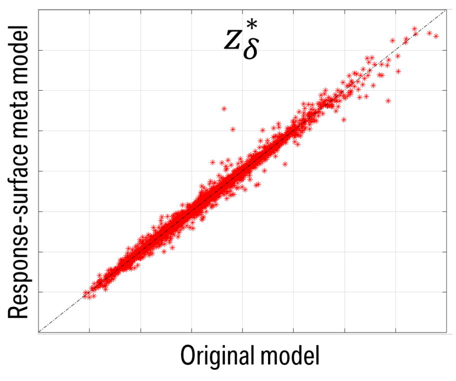
In order to simulate the manoeuvres introduced in subsection 7.1.2, a modified version [40] of the classical two-track model proposed in [32] is used. The model is implemented in MATLAB®. Tyres are modelled with the Pacejka's empirical Magical Formula [51]. Based on the simulation results of a high-fidelity simulation model ADAMS®, a look-up table for the interaction of the car body and the tyres is derived.

In the following, a Response-Surface Metamodel (RSM) is used to replace the actual expensive computer analyses of the modified two-track model. On the one hand, this facilitates the execution of non-gradient-based optimisation, since these methods usually need thousands of function evaluations. On the other hand, gradient-based optimisation is enabled since gradient information can easily be derived on RSMs [61]. On an Intel Xeon E5-2650 v2 @ 2.6 GHz processor a single design evaluation on the modified two-track model (simulating the QSSC, RAST, and the SWD manoeuvre) takes 28 s. On the RSM, the same evaluation takes 7.0×10^{-3} s.

The specific RSM used is a single layer ANN with a sigmoid basis function and 5 to 30 neurons. We tried to apply more than one hidden layer and found that the results where worse due to over-fitting. The numbers of neurons we tried were $5, 10, 15, \ldots, 30$ for each



(a)



(b)

Figure 7.3 Regression plots for the maximum steering angle factor trained with (a) the full data set z_δ and with (b) a filtered data set z_δ^* (the black sample points were discarded).

output. We always chose the variant with the best coefficient of determination R^2. It was trained with a classical back propagation algorithm [77] based on a dataset of $10,000$ sample designs. The evaluated designs were created by MC sampling in the design space Ω_{ds}. Only 90% of the samples were used during training. The rest was used to validate the model by computing the coefficient of determination R^2 [50]. As can be inferred from table 7.3, the RSM describes the dependence of the performance measures on the design variables almost perfectly. The only coefficient of determination lower then 0.99 is obtained for the maximum steering angle factor z_δ. Figure 7.3a shows the regression plot for z_δ. As can be inferred from the regression plot, the original model includes a jump in the performance measure. This is due to the fact that some designs will cause instability in the two-track model during the SWD manoeuvre. Sudden jumps in the output cannot be described appropriately by an ANN. In addition, we are not interested in these unstable outputs, since they do not describe the actual vehicle behaviour. Hence, these outputs are filtered before training the ANN. The filter discards all samples, which exceed a certain critical output value (black sample points in figure 7.3 (a)). The cluster of samples of these filtered training points leads to a coefficient of determination of 0.989. The associated regression plot is displayed in figure 7.3b. For the purpose of this thesis the newly trained model z_δ^* is used for all applications.

z_α	z_{a_y}	z_{F_z}	z_{zRA}	z_{zFA}	z_Φ	z_C	z_δ / z_δ^*
1.000	0.998	0.999	0.999	0.999	0.999	0.997	0.836 / 0.989

Table 7.3 Coefficient of determination R^2 for the RSM. $10,000$ samples have been used as training data for each of the models. The filtered Set for z_δ^* contained $9,925$ samples.

7.2. Solution-Compensation Spaces

In the following, the Basic Projection Algorithm (see subsection 4.2.2), the FME Algorithm with and without RR (see subsection 4.2.3) and the Stochastic Solution-Compensation Space Algorithm (see subsection 4.3.2) are applied to the industrial chassis design problem. The different approaches are compared with respect to computational effort, the size and accuracy of the derived Solution-Compensation Spaces as well as the scope of possible applications.

7.2.1. Application of Solution-Compensation Spaces

The vehicle dynamics system considered in this thesis consists of eight design variables $x \in \mathbb{R}^d$, $d = 8$, eight performance measures as well as eight performance constraints $f(x) \leq f_c \in \mathbb{R}^m$, $m = 8$. In the following, technical problem statement 1 (see subsection 3.1.2) is considered. The goal is to derive a set-based design $\Omega = I_1 \times \ldots \times I_d$, which fulfils both robustness constraints $I \geq I_{\min}$ and performance constraints $\forall x \in \Omega, f(x) \leq f_c$. In subsection 3.1.3, we showed that state of the art algorithms do not solve this technical problem statement. The box-shaped Solution Spaces that were computed using state of the art algorithms are too small for the application. Larger intervals are required to encompass the deviation between desired and realised design variable values. Therefore, Solution-Compensation Spaces are computed.

Solution-Compensation Spaces solve a modified problem statement. The performance constraints are fulfilled such that for each combination of early-decision parameters inside the Solution-Compensation Space, a combination of late-decision parameters exists, which leads to a good design:

$$\forall x_a, \exists x_b,\ f(x_a, x_b) \leq f_c.$$

Since technical problem statement 1 specifically states that the design variables of both the anti-roll bar and the bump can be modified in the later stages of the development process, these variables are categorised as late-decision variables x_b. For the design of the tyres and the suspensions in an early-development phase, a large Solution Space is sought. Hence, the corresponding variables are considered as early-decision variables x_a. An overview of all early- and late-decision parameters is given in table 7.4.

$\mu_{max,X}$	$\mu_{max,Y}$	$z_{rc,RA}$	$z_{rc,FA}$	$c_{\text{arb},RA}$	$c_{\text{arb},FA}$	$c_{bs,RA}$	$c_{bs,FA}$
early	early	early	early	late	late	late	late
$x_{a,1}$	$x_{a,2}$	$x_{a,3}$	$x_{a,4}$	$x_{b,1}$	$x_{b,2}$	$x_{b,3}$	$x_{b,4}$

Table 7.4 Early- and late-decision variables

When we compare the application of Solution-Compensation Spaces to the application of classical Solution Spaces with pre-optimised values for the late-decision variables (see subsection 3.1), we lose the information about which values the late-decision variables will assume until all values for the early-decision variables have been chosen. In exchange, we gain increased interval sizes for all early-decision variables, which provides the much desired increased robustness.

Since the result of the Basic Projection Algorithm as well as the FME Algorithm is a projected system and not an Solution-Compensation Space, a post-processing step is necessary. Therefore, either Vertex Tracking (VT) or the Stochastic Solution Space Algorithm (SSA) are applied. These algorithms are state of the art (see section 2.1). Both algorithms maximise the size of the Solution-Compensation Space. While VT guarantees that only good designs are included, SSA estimates the fraction of good designs in the box by evaluating an MC sample. Hence, only probabilistic statements on the fraction of good designs are possible.

Application of the Basic Projection Algorithm

The Basic Projection Algorithm is only applicable to the chassis design problem with linearised performance constraints. Fortunately, as shown in subsection 7.1.2, it is possible to linearise the constraint functions sufficiently well. The computational effort to execute the Basic Projection Algorithm is dominated by the computation of all possible vertices. The chassis design problem has eight input dimensions and eight performance constraints, this leads to

$$\Lambda_c = \begin{pmatrix} 2 \times d + m \\ d \end{pmatrix} = \begin{pmatrix} 2 \times 8 + 8 \\ 8 \end{pmatrix} = 735,471$$

vertices, which need to be calculated when applying the Basic Projection Algorithm. Computing those took 42 seconds on an Intel Xeon E5-2650 v2 @ 2.6 GHz processor. The obtained early-decision-variable space after projection is shown in figure 7.4(a)-(d). The black box shows the resulting SCS. For an easier comparison, the dashed black boxes show the required size from scenario 1/2 (see table 3.3). The values of all early-decision variables not shown in the plot are randomly chosen within the Design Space (DS). For each green design point in figure 7.4 there exists at least one combination of late-decision variables such that all design goals are reached. For each differently coloured design point the opposite is true. Unfortunately, the colour cannot be linked to a single performance constraint since the projected constraints are generally the result of multiple intersecting constraints from the original system. Since the results are shown in the projected system, the late-decision parameter dimensions do not exist and hence are not displayed. The results for the Basic Projection Algorithm match the results of the FME since in both cases the linear system is projected and then a classical SS optimiser is applied.

	m_t	m_{t1}	m_{t2}	m_{t3}	m_{t4}
FME without RR	24	144	2.07×10^4	2.96×10^6	4.30×10^8
FME with RR	24	24	52	19	11

Table 7.5 Number of constraints after each projection step for the chassis design problem when applying the FME with and without RR.

Application of the Fourier-Motzkin Elimination Algorithm

The FME Algorithm is only applicable to the chassis design problem with linearised performance constraints and can be applied with or without RR.

The computational effort to execute FME *without RR* is dominated by the derivation of the projected constraints. The projection of the four late-decision variable dimensions (q=4), when applying FME without RR, requires the computation of up to

$$m_{r_4} = \sum_{k=1}^{q} \left(\frac{2 \times d + m}{2} \right)^{2^k} = \sum_{k=1}^{4} \left(\frac{2 \times 8 + 8}{2} \right)^{2^k} = 432,988,560$$

constraints. As described in subsection 4.2.3, this is a worst case estimation. Unfortunately, this estimation is relatively close to the actual number of constraints, which are derived. Computing these constraints took over 7 hours on an Intel Xeon E5-2650 v2 @ 2.6 GHz processor.

The computational effort to execute FME *with RR* is dominated by the solving of the LPPs contained in the Clarkson Algorithm (see algorithm 2). The projection of the four late-decision variable dimensions, when applying FME with RR, requires a total of 1064 LPPs to be solved. The final system consists of $m_{t4} = 11$ constraints. Computing those took 0.06 seconds on an Intel Xeon E5-2650 v2 @ 2.6 GHz processor.

Table 7.5 shows the number of constraints considered after each projection step when applying FME with and without RR. The resulting Solution-Compensation Space is shown in figure 7.4 (a)-(d).

Application of the Stochastic Solution-Compensation Space Algorithm

The Stochastic Solution-Compensation Space Algorithm is the only Solution-Compensation Space algorithm introduced in this thesis, which is applicable to the original chassis design problem with non-linear performance constraints.

The computational effort to execute the Stochastic Solution-Compensation Space Algorithm is dominated by the evaluation of the performance functions. Before the computation starts, the number of iteration steps v_1, number of samples per candidate box N_1, maximum number of iterations for the particle swarm optimisation v_2, and number of particles N_2 are chosen. According to table 4.1, each number has an influence on the resulting Solution-Compensation Space. We ran three different configurations. These

Table 7.6 Parametrisation of the Stochastic SCS Algorithm and the resulting duration of the computation.

Parametrisation	v_1^{PhaseI}	$v_1^{PhaseII}$	N_1	v_2	N_2	runtime
1	100	100	100	100	20	337 s
2	200	100	100	200	40	495 s
3	50	100	100	50	10	212 s

configurations and their computation time on an Intel Xeon E5-2650 v2 @ 2.6 GHz processor are shown in table 4.1. Parametrisation 1 is the standard configuration, while parametrisation 2 increases and parametrisation 3 decreases the computational effort. N_1 and $v_1^{PhaseII}$ are kept constant in order to ensure a sufficient confidence level.

In order to properly evaluate the accuracy of the Solution-Compensation Space algorithm, it is executed on the linearised system. Therefore, the result can be validated in the projected system derived by the FME algorithm. The resulting Solution-Compensation Spaces are shown in figure 7.4 (e)-(j).

7.2.2. Comparison of different methods to compute Solution-Compensation Spaces

In the following, the different methods to compute SCS are compared with respect to computational effort and size accuracy. In the end, a conclusion is drawn, which algorithm fits the chassis design problem best.

Computational Effort

In table 7.7, an overview of the different algorithms is given. Note that the runtime for the Basic Projection and the FME algorithms includes the computation of the Solution-Compensation Space. Both, the Vertex Tracking and the Stochastic Solution Space Algorithm, have a low computational effort and scale well with the number of dimensions. The computation of the Solution-Compensation Space in the projected system with the Vertex Tracking Algorithm took 3.19 s and the computation with the Stochastic Solution Space Algorithm took 0.53 s.

The computation time for the FME Algorithm without redundancy removal is the longest. This is due to the fact that its computational effort increases double exponentially with the number of projected dimensions. If only one dimension is projected, it is a valid choice. If more dimensions are projected, the additional RR Algorithm is recommended. For this example, FME with RR is the fastest algorithm taking less than one second. The Basic Projection algorithm took 42 s. Its computational effort grows exponentially with the total number of dimensions. Hence, it is only recommended for low dimensional problems. Note that even in low dimensions, FME with RR is often faster

Table 7.7 Overview of the results of the SCS algorithms with respect to runtime, size, accuracy, fulfilment of the requirements from scenario 1/2, and whether an application to systems with non-linear performance constraints is possible.

	Basic Projection		FME without RR		FME with RR		Stochastic SCS Algorithm		
	VT	SSA	VT	SSA	VT	SSA	1	2	3
runtime	45s	43s	7.0h	7.0h	3.3s	0.6s	337s	495s	212s
size $\frac{\mu(\Omega)}{\mu(\Omega_1)}$	6.00	7.65	6.00	7.65	6.00	7.65	7.96	8.84	4.02
accuracy	100%	99.8%	100%	99.8%	100%	99.8%	99.1%	98.5%	99.8%
scenario 1	x	✓	x	✓	x	✓	✓	✓	x
scenario 2	x	x	x	x	x	x	x	x	x
non-linear	x	x	x	x	x	x	✓	✓	✓

than Basic Projection. The computational effort of the stochastic Solution-Compensation Space algorithm does not scale with the number of dimensions but is highly dependent on its parameter configuration and the computational effort of a single function evaluation. Even though we are working with an RSM, which takes 7.0×10^{-3} s to evaluate a single design, the total computation time is comparatively high with ≥ 212 s. Considering its computational effort, the stochastic Solution-Compensation Space Algorithm is mostly recommended in combination with RSMs.

Size and accuracy of the Solution-Compensation Spaces

In order to estimate the accuracy of the different algorithms, an MC sample with 10, 000 points is computed in the early-decision dimensions of the final Solution-Compensation Space. Each point is evaluated using the projected system derived with FME. The results are shown in table 7.7.

When we consider the size of an Solution-Compensation Space, generally two things are important: the size of the interval of each individual early-decision variable and the total size of the Solution-Compensation Space $\mu(\Omega)$. Whether the results fulfil the robustness requirements of scenario 1/2 is shown in table 7.7. Details about the size of the intervals are given in table 7.8. The total size of the Solution-Compensation Space represents the target function for the optimisation algorithm and is maximised. The total sizes divided by the size of the required Solution-Compensation Space from scenario 1 $\mu(\Omega_1)$ are shown in table 7.7.

The results obtained from the Basic Projection and the FME Algorithm are identical. This was to be expected since both algorithms simply project a linear system.

When Vertex Tracking is applied after the projection, an Solution-Compensation Space with 100% accuracy is obtained. Unfortunately, the resulting Solution-Compensation Space does not fulfil the robustness requirements. This is visualised in figure 7.4 (a)/(b). The dashed boxes show the requirements from Scenario 1 and 2. Note that these requirements are identical for the roll centre height. Hence, only one dashed box is shown.

When the stochastic Solution Space Algorithm is applied after the projection, the accuracy is below 100%. Therefore, as shown in figure 7.4, the size of the Solution Space is larger. In our test case, the resulting Solution-Compensation Space fulfils all robustness requirements and has an accuracy of 99.8%. The location of the resulting Solution Space is similar to the one derived with VT, only the intervals are enlarged due to less strict requirements with respect to the performance constraints.

The results from the Stochastic Solution-Compensation Space Algorithm are hugely dependent on the parametrisation of the algorithm: parametrisation 1 (see figure 7.4 (e)-(f)) finds an Solution-Compensation Space, which fulfils the robustness constraints and has an accuracy of 99.1%. Parametrisation 2 yields a slightly larger Solution-Compensation Space but only has an accuracy of 98.5%. The decreased accuracy and the increased size are likely to be a result of the additional computational effort put into the PSO, which lowers the likelihood that individual samples are wrongly labelled as bad design (false negatives). Hence, in parametrisation 2, the same candidate Solution-Compensation Space is more likely to be accepted than in parametrisation 1 because 100/100 are good designs. In both parametrisation 1 and 2 the iterations in Phase II stopped because an Solution-Compensation Space with 100/100 good MC samples was found and not because 100 iteration steps where actually reached. Parametrisation 3 yields by far the smallest Solution-Compensation Space. We can assume that in Phase I not enough iterations were executed.

Conclusion

Depending on the problem statement, a different algorithm to compute Solution-Compensation Spaces is needed. For most applications with linear constraints, the FME Algorithm with RR is recommended. It requires low computational effort compared to the Basic Projection Algorithm and the FME Algorithm without RR. Furthermore, it yields an exact projection of the system. Whether the Vertex Tracking or the Stochastic Solution Space Algorithm should be applied after the projection depends on the accuracy and the size, which is required. Vertex Tracking always has 100% good designs but the Stochastic Solution Space Algorithm generally yields better results with respect to the size measure.

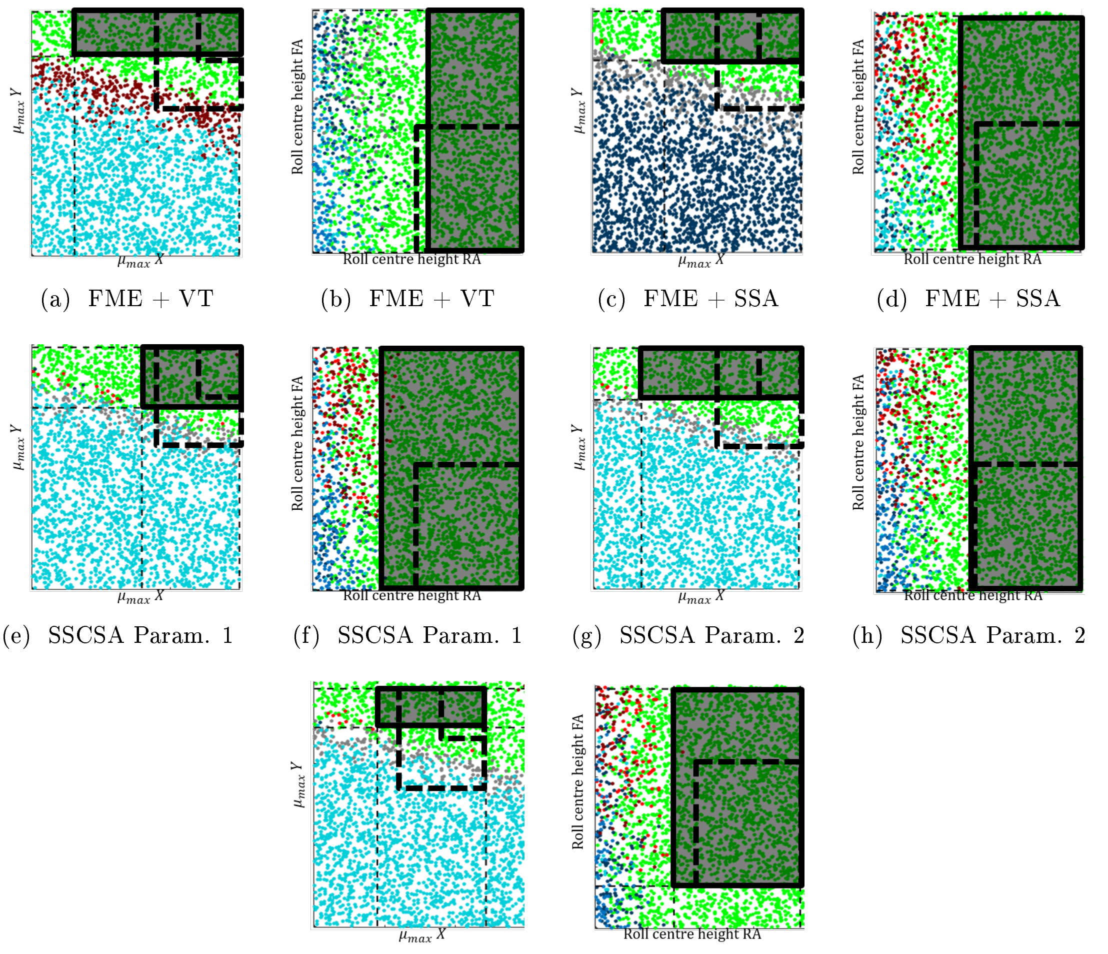
The main advantage of the Stochastic Solution-Compensation Space Algorithm is that it is applicable to problem statements with non-linear constraints. Unfortunately, it is mostly applicable to problems where a single function evaluation is computationally cheap.

For the chassis design problem discussed here, the Stochastic Solution-Compensation Space Algorithm with parametrisation 1 is a good choice. With 337 s the runtime is acceptable. It yields a result, which fulfils all robustness requirements of scenario 1 and most importantly it is applicable to non-linear performance constraints. Even though the

Table 7.8 Interval sizes (normalised with respect to requirements of scenario 1) and the resulting volume of the SCS for $\mu_{\max}$ and h_{Ro}.

	$\mu_{\max,X}$	$\mu_{\max,Y}$	$h_{\mathrm{Ro,RA}}$	$h_{\mathrm{Ro,FA}}$
scenario 1 (premium)	1.00	1.00	1.00	1.00
scenario 2 (standard)	2.00	2.00	1.00	1.00
Basic Projection or FME with Vertex Tracking	3.97	0.84	0.90	2.00
Basic Projection or FME with the stochastic Solution Space Algorithm	3.29	1.01	1.18	1.96
stochastic SCS Opt. 1	2.36	1.24	1.36	2.00
stochastic SCS Opt. 2	3.81	1.07	1.09	1.99
stochastic SCS Opt. 3	2.57	0.80	1.20	1.63

linearised performance constraints approximate the actual constraints very well, a small error cannot be prevented. In case a faster solution is needed, the FME Algorithm with RR plus the Stochastic Solution Space Algorithm is a good choice. It yields a result, which fulfils all robustness requirements of scenario 1 with a runtime of only 0.6 s and accuracy of 99.8%. Of course a minor error due to linearisation has to be accepted.



(a) FME + VT (b) FME + VT (c) FME + SSA (d) FME + SSA

(e) SSCSA Param. 1 (f) SSCSA Param. 1 (g) SSCSA Param. 2 (h) SSCSA Param. 2

(i) SSCSA Param. 3 (j) SSCSA Param. 3

Figure 7.4 (a)-(d) SCS (black box) optimised either with Vertex Tracking (VT) or the Stochastic Solution-Compensation Space Algorithm (SSCSA) and for reference an SCS, which fulfils the robustness requirements (dashed box). (e)-(f) SCS optimised with the Stochastic Solution-Compensation Space Algorithm (SSCSA) with the 3 different parametrisation from table 7.6.

7.3. Optimal Constraint Relaxation for Solution Spaces

In the following, the algorithm to compute Optimal Constraint Relaxation for Solution Spaces with linear performance constraints (see Section 5.2.1) as well as the algorithm for non-linear performance constraints (see Section 5.3) are applied to the industrial chassis design problem introduced in section 7.1. Therefore, the determination of weighting factors q_i to evaluate the relaxation of different performance constraints according to the target function (5.4) is crucial. Different approaches on how to determine the weighting factors were introduced in section 5.4. The different Optimal Constraint Relaxation for Solution Spaces approaches are compared with respect to computational effort, accuracy, and possible applications.

7.3.1. Application of Optimal Constraint Relaxation for Solution Spaces

In the following, technical problem statement 2 (see subsection 3.1.2) is considered. The goal is to derive a set-based design $\Omega = I_1 \times \ldots \times I_d$ and a set of relaxed performance constraints $f(x) \leq f_c + \Delta f_c$. The set-based design Ω is such that it fulfils the robustness constraints $I \geq I_{\text{min}}$ and the relaxed performance constraints $\forall x \in \Omega, f(x) \leq f_c + \Delta f_c$. The set of relaxed performance constraints has to be optimal in the sense that each constraint is relaxed by as little as possible. In subsection 3.1.3, we introduced a basic approach to solve this problem statement. Unfortunately, it is computationally expensive, does not work for arbitrary non-linear performance constraints, and is probably not optimal concerning minimal constraint relaxation. A more advanced algorithm is needed that is able to optimise the constraint relaxation and does not scale exponentially with the number of dimensions. Therefore, Optimal Constraint Relaxation for Solution Spaces is computed.

In order to be able to compare the results of different approaches, a target function is introduced, which evaluates the amount of constraint relaxation:

$$g(\Delta f_c) = \sum_{i=1}^{m} q_i (\Delta f_{c,i})^2. \tag{7.1}$$

Application of Optimal Constraint Relaxation for Solution Spaces with linear performance constraints

The Basic Projection Algorithm is only applicable to the chassis design problem with linearised performance constraints. The computational effort to execute Optimal Constraint Relaxation for Solution Spaces with linear performance constraints is dominated by the computation of the Interior Point algorithm, which is computationally cheap for quadratic target functions. Initially, four different configurations are computed: scenario 1/2 with generic and RI-based weighting factors. In scenario 1 the robustness constraints for the design of the high performance tyres are set. A relatively small Solution Space is sufficient here. Similarly, scenario 2 sets the robustness constraints for the design of the standard tyre. Here, a larger Solution Space is required (defined in table 3.3). The

RI	$\Delta\text{RI}z_\alpha$	$\Delta\text{RI}z_{a_y}$	$\Delta\text{RI}z_{F_z}$	$\Delta\text{RI}z_{z_{RA}}$	$\Delta\text{RI}z_{z_{FA}}$	$\Delta\text{RI}z_\Phi$	$\Delta\text{RI}z_C$	$\Delta\text{RI}z_\delta$
	0.13	0.50	200	2.50×10^{-4}	2.50×10^{-4}	1.57×10^{-2}	1	0.5
gen	$f_c z_\alpha$	$f_{c,z_{a_y}}$	$f_{c,z_{F_z}}$	$f_{c,z_{z_{RA}}}$	$f_{c,z_{z_{FA}}}$	f_{c,z_Φ}	f_{c,z_C}	f_{c,z_δ}
	0.36	9.1	560	9.00×10^{-3}	9.00×10^{-3}	5.50×10^{-2}	2.90	4.5

Table 7.9 Overview of all ΔRI and f_c for the chassis design problem in SI units. The weighting factors are either generic (gen) or according to RI.

results of Optimal Constraint Relaxation for Solution Spaces are shown in figure 7.6. The different colour points represent distinct chassis designs. Green designs fulfil all relaxed constraints, the other colours depict which constraints are violated (see the colour-code in table 3.2). As can be seen, the optimal position of the Solution Space changes with different target functions and different robustness requirements. Especially the permissible intervals for the tyre parameter $\mu_{\max X}$ shift depending on whether the constraint concerning maximum lateral acceleration z_{a_y} (🟪 colour) or minimum vertical tyre force z_{F_z} (⬛ colour) is relaxed by a larger margin. Table 7.10 includes the margins by which the constraints are relaxed and the target function values for the different configurations. Table 7.9 includes the vectors f_c and ΔRI, which were used to determine the weighting factors for the target functions (7.1) according to section 5.4.

Application of Optimal Constraint Relaxation for Solution Spaces with non-linear performance constraints

Optimal Constraint Relaxation for Solution Spaces with non-linear performance constraints is applied to the original chassis design problem. Similarly to the application with linear performance constraints, scenario 1/2 and two different sets of weighting factors are considered (see table 7.9). The results are visualised in figure 7.7. The position of the resulting Solution Space varies more than in the linear approach. The buffer spring stiffness at the front axles assumes values within the whole range of its design space. For reference, the results from the basic approach (introduced in subsection 3.1.3) are visualised in figure 3.3.

As described in section 5.3, Optimal Constraint Relaxation for Solution Spaces can be applied to any problem statement with arbitrary non-linear performance constraints. But the accuracy of the result can only be guaranteed if all constraints are monotonic. Unfortunately, that is not the case for the non-linear chassis design problem. Instead, we validate the resulting Solution Space by evaluating 10,000 MC sample designs. An overview of the runtime, size, accuracy, number of iterations, constraint relaxation, and target function values for all non-linear approaches is given in table 7.11. The accuracy is 100% for all approaches. As was to be expected, the target values obtained by Optimal Constraint Relaxation for Solution Spaces are superior to the basic approach due to the use of an appropriate target function.

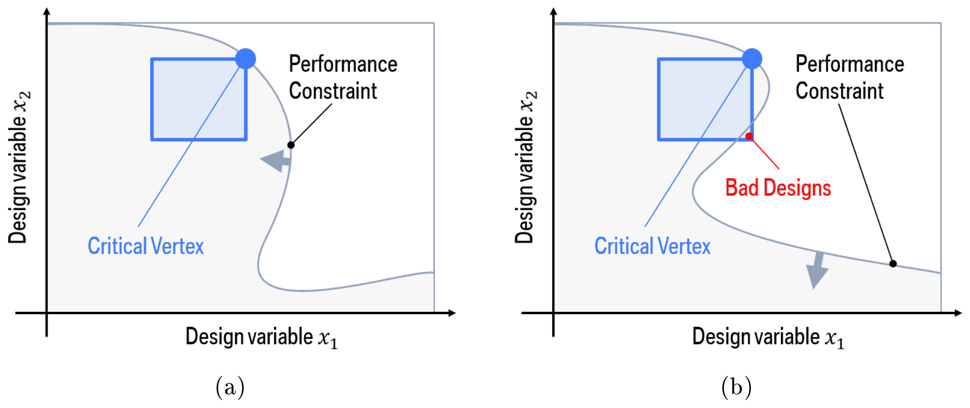


Figure 7.5 (a) Example for a constraint, which is globally non-monotonic, but is monotonic in a sufficiently large area around the critical vertex. (b) Example for a constraint, which is globally non-monotonic and is not monotonic in a sufficiently large area around the critical vertex to ensure 100% accuracy.

7.3.2. Comparison of different methods to compute Optimal Constraint Relaxation for Solution Spaces

In the following, the different methods to compute Optimal Constraint Relaxation for Solution Spaces are compared with respect to computational effort, performance, and accuracy. In the end, a conclusion is drawn which algorithm fits the chassis design problem best. In tables 7.10 and 7.11, an overview of the results of the different algorithms is given. These results are used as a basis for the following analysis.

Computational Effort

The runtime for the linear systems is significantly lower than for non-linear systems. In the linear case, the optimisation problem needs to be solved only once and no linearisation is needed. Here, the computational effort mainly scales with the number of constraints $\mathcal{O}(\sqrt{m})$. Hence, the method is applicable for high-dimensional linear problems. The computation time of Optimal Constraint Relaxation for Solution Spaces with non-linear performance constraints is highly dependent on the computational effort needed to execute a linearisation. Since the non-linear chassis design problem is computed on an RSM, a single function evaluation is cheap and the first derivative is given in explicit form, which makes the linearisation relatively cheap. No finite differences need to be computed. As long as the linearisation is computationally cheap, Optimal Constraint Relaxation is applicable for non-linear high-dimensional problems. It outperforms the basic approach in 8 dimensions. With an increasing number of dimensions, Optimal Constraint Relaxation will outperform the basic approach even more significantly since the basic approach considers all vertices and hence its computational effort scales exponentially $\mathcal{O}(2^d)$.

Accuracy and performance with respect to the target function

For the chassis design problem the accuracy of all the tested algorithms is 100%. This implies that a sufficiently large area around the critical vertices exists in which the non-linear performance constraints behave monotonously. An example for such a constraint is shown in figure 7.5(a). The required size of the area depends on the pre-defined box size. If the box size is decreased, the required area, in which the constraint needs to behave monotonously, becomes smaller. Figure 7.5(b) shows a constraint, which is not monotonic in a sufficiently large area around the critical vertex to ensure feasibility for the entire Solution Space. Hence the accuracy is $< 100\%$.

As expected, the target values reached with Optimal Constraint Relaxation are generally better than the target values reached with the basic approach. When comparing Optimal Constraint Relaxation for the linear and non-linear system, we see that the target values for the linear problem are superior. In each instance, the iterative process for the non-linear system terminated because the position of the Solution Space did not change significantly any more. For the RI-based target function the difference with respect to the target values for the linear and non-linear approach is small enough ($5.58 \times 10^{-3}/6.71 \times 10^{-3}$ and $6.95 \times 10^{-2}/8.11 \times 10^{-2}$) that we can assume it is mostly a result of the linearisation error. This proposition is further confirmed by figures 7.6(e)-(h)/(m)-(p) and 7.7(e)-(h)/(m)-(p) where we can see that the positioning of the Solution Spaces is relatively similar. For the generic target function, this is not the case. The substantial differences in the target values ($3.60 \times 10^{-5}/1.90 \times 10^{-4}$ and $4.66 \times 10^{-4}/1.37 \times 10^{-3}$) and the significantly different positioning of the Solution Spaces are visualised in figures 7.6(a)-(d)/(i)-(l) and 7.7(a)-(d)/(i)-(l) indicating that the non-linear algorithm did not find the global minimum, but got stuck in a local minimum.

Conclusion

Depending on the problem statement, a different algorithm to compute Optimal Constraint Relaxation for Solution Spaces is needed. For most applications with linear constraints, the linear approach (problem statement 13) with the generic target function is recommended. It requires low computational effort, always converges towards the global optimum and no additional gradient information is required. In case additional gradient information is available, case-specific weighting factors (as introduced in subsection 5.4.2) are recommended.

In case the system has non-linear performance constraints, only the iterative Optimal Constraint Relaxation algorithm (alg. 7) is applicable. It is computationally more expensive than the linear algorithm. Generally, a large number of linearisations is necessary. Hence, the algorithm is mostly applicable to systems where the linearisation is computationally cheap. Note that the linearisation is computationally cheap if function evaluations are cheap. 100% accuracy as well as convergence towards the global optimum of the iterative algorithm can only be guaranteed if all constraints are monotonic (as defined in (5.19) and (5.18)).

For the chassis design problem discussed here, Optimal Constraint Relaxation for Solution Spaces with non-linear performance constraints and RI-based weighting factors are recommended. With a runtime of less than one second, the algorithm yields a result, which has 100% accuracy and is likely to be very close to the global optimum. At the same time, the linearisation error is minimised since, instead of a global stepwise linearisation, each constraint is linearised around its critical vertex. The final result is then displayed in the original non-linear system. In addition, the available expert knowledge is included in the final result by applying RI-based instead of generic weighting factors.

7.4. Optimal Constraint Relaxation for Solution-Compensation Spaces

In the following, the algorithm to compute Optimal Constraint Relaxation for Solution-Compensation Spaces with linear performance constraints (see Section 6.2) is applied to the linearised industrial chassis design problem introduced in section 7.1. Therefore, the weighting factors q_i to evaluate the relaxation of different performance constraints were chosen according to the generic approach introduced in section 5.4. The different algorithms to compute Optimal Constraint Relaxation for Solution-Compensation Spaces are compared with respect to computational effort and target value.

7.4.1. Application of Optimal Constraint Relaxation for Solution-Compensation Spaces

Technical problem statements 1 and 2 (see subsection 3.1.2) are considered. In order to further decrease the amount of relaxation needed compared to Optimal Constraint Relaxation for Solution Spaces, we determine a subset of early- and late-decision parameters (according to table 7.4). The goal here, is to derive a set-based design for the early-decision parameters $\Omega_a = I_1 \times \ldots \times I_p$ and a set of relaxed performance constraints $f(x_a, x_b) \leq f_c + \Delta f_c$ with the following properties:

- For each combination of early-decision parameters $x_a \in \Omega_a$ a combination of late decision parameters exists such that the relaxed constraints are fulfilled: $\forall x \in \Omega, \exists x_b \in \Omega_{b,\text{ds}}, f(x_a, x_b) \leq f_c + \Delta f_c$
- The robustness constraints are fulfilled exactly $I_i = I_{i,\min}$ for $i = 1, \ldots, p$.
- The set of relaxed performance constraints has to be optimal in the sense that each constraint is relaxed by as little as possible; minimise $g(\Delta f_c)$.

Therefore, Optimal Constraint Relaxation for Solution-Compensation Spaces is computed. This approach combines the advantages of Solution-Compensation Spaces and Optimal Constraint Relaxation for Solution Spaces, since it uses late-decision parameters to compensate larger early-decision variable intervals while also being able to compute a solution for any arbitrarily strict performance and robustness constraints.

In section 7.3 we used two different sets of weighting factors in order to show that different types of weighting factors can be applied depending on the problem statement.

linear	scenario 1		scenario 2	
	OCR gen	OCR RI	OCR gen	OCR RI
runtime [s]	0.031	0.025	0.027	0.026
size $\frac{\mu(\Omega)}{\mu(\Omega_1)}$ [−]	1.00	1.00	4.00	4.00
Δz_α $\left[\text{rad}/\frac{\text{m}}{\text{s}^2}\right]$	0	0	0	0
Δz_{a_y} $\left[\text{m}/\text{s}^2\right]$	4.96×10^{-2}	2.34×10^{-2}	0.159	8.87×10^{-2}
Δz_{F_z} [N]	0.423	8.40	2.43	26.9
$\Delta z_{z_{RA}}$ [m]	0	0	0	0
$\Delta z_{z_{FA}}$ [m]	0	0	0	0
Δz_Φ [rad]	0	6.34×10^{-5}	0	3.24×10^{-5}
Δz_C [−]	0	0	3.21×10^{-3}	0
Δz_δ [−]	1.07×10^{-2}	2.00×10^{-2}	5.31×10^{-2}	7.08×10^{-2}
$g_{\text{gen}}(\Delta f_c)$	3.60×10^{-5}	2.53×10^{-4}	4.66×10^{-4}	2.64×10^{-3}
$g_{\text{RI}}(\Delta f_c)$	1.03×10^{-2}	5.58×10^{-3}	0.113	6.95×10^{-2}

Table 7.10 Linearised chassis design problem. Overview of the results of OCR for Solution Spaces for the linearised chassis design problem with respect to runtime (on an Intel Xeon E5-2650 v2 @ 2.6 GHz processor), size, constraint relaxation, and value of the target function from scenario 1/2.

non-linear	scenario 1			scenario 2		
	OCR gen	OCR RI	Basic	OCR gen	OCR RI	Basic
runtime [s]	0.613	0.621	7.47	0.428	0.553	7.50
size $\frac{\mu(\Omega)}{\mu(\Omega_1)}$ [-]	1.00	1.00	1.00	4.00	4.00	4.00
accuracy	100%	100%	100%	100%	100%	100%
Iterative Loops	8	6	-	6	7	-
Δz_α $\left[\text{rad}/\frac{\text{m}}{\text{s}^2}\right]$	0	0	0	0	0	0
Δz_{a_y} $\left[\text{m/s}^2\right]$	0.107	2.67×10^{-2}	0	0.225	0.103	0.359
Δz_{F_z} [N]	1.56	9.38	0	4.46	31.0	10.6
$\Delta z_{z_{RA}}$ [m]	0	0	2.00×10^{-4}	5.43×10^{-5}	0	2.00×10^{-4}
$\Delta z_{z_{FA}}$ [m]	1.55×10^{-5}	3.22×10^{-7}	0	0	2.76×10^{-6}	0
Δz_Φ [rad]	3.14×10^{-4}	2.01×10^{-4}	7.15×10^{-4}	5.26×10^{-4}	6.68×10^{-4}	2.02×10^{-3}
Δz_C [-]	0	1.23×10^{-3}	0	0	0	0
Δz_δ [-]	1.23×10^{-2}	1.93×10^{-2}	6.00×10^{-2}	0.107	5.63×10^{-2}	0.110
$g_{\text{gen}}(\Delta f_c)$	1.90×10^{-4}	3.21×10^{-4}	8.41×10^{-4}	1.37×10^{-3}	3.50×10^{-3}	4.35×10^{-3}
$g_{\text{RI}}(\Delta f_c)$	5.10×10^{-2}	6.71×10^{-3}	0.656	0.297	8.11×10^{-2}	1.22

Table 7.11 Non-linear chassis design problem. Overview of the results of OCR for Solution Spaces for the non-linear chassis design problem with respect to runtime (on an Intel Xeon E5-2650 v2 @ 2.6 GHz processor), size, constraint relaxation and target function value from scenario 1/2.

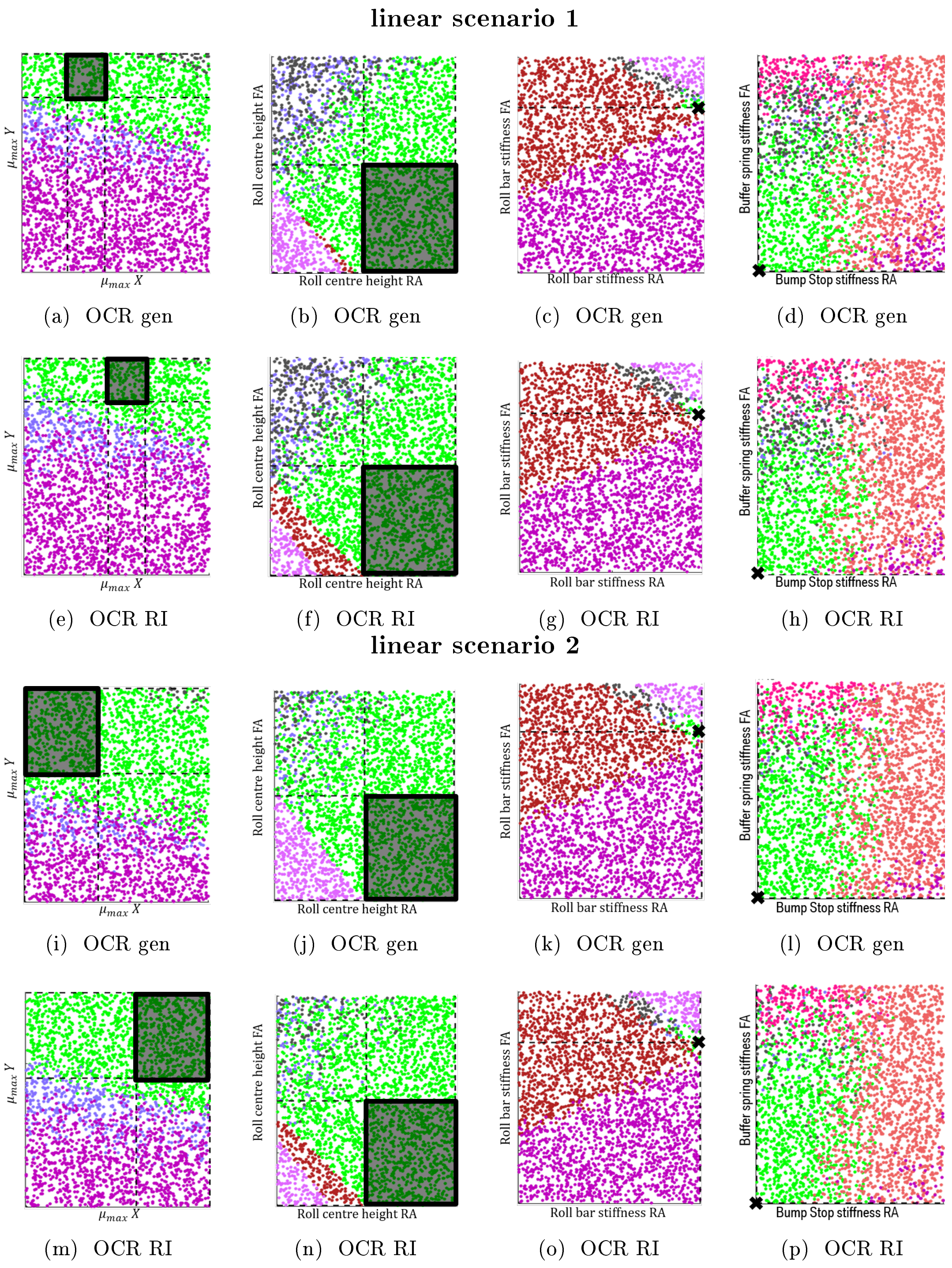


Figure 7.6 Linearised chassis design problem. (a)-(h)/(i)-(p) an Solution Space (black box) that fulfils the robustness requirements for scenario 1/2 (see table 3.3) while fulfilling the relaxed performance constraints for scenario 1/2 (see table 3.5). The constraints are relaxed with respect to the target function (5.4). The weighting factors are either generic (gen) or according to RI.

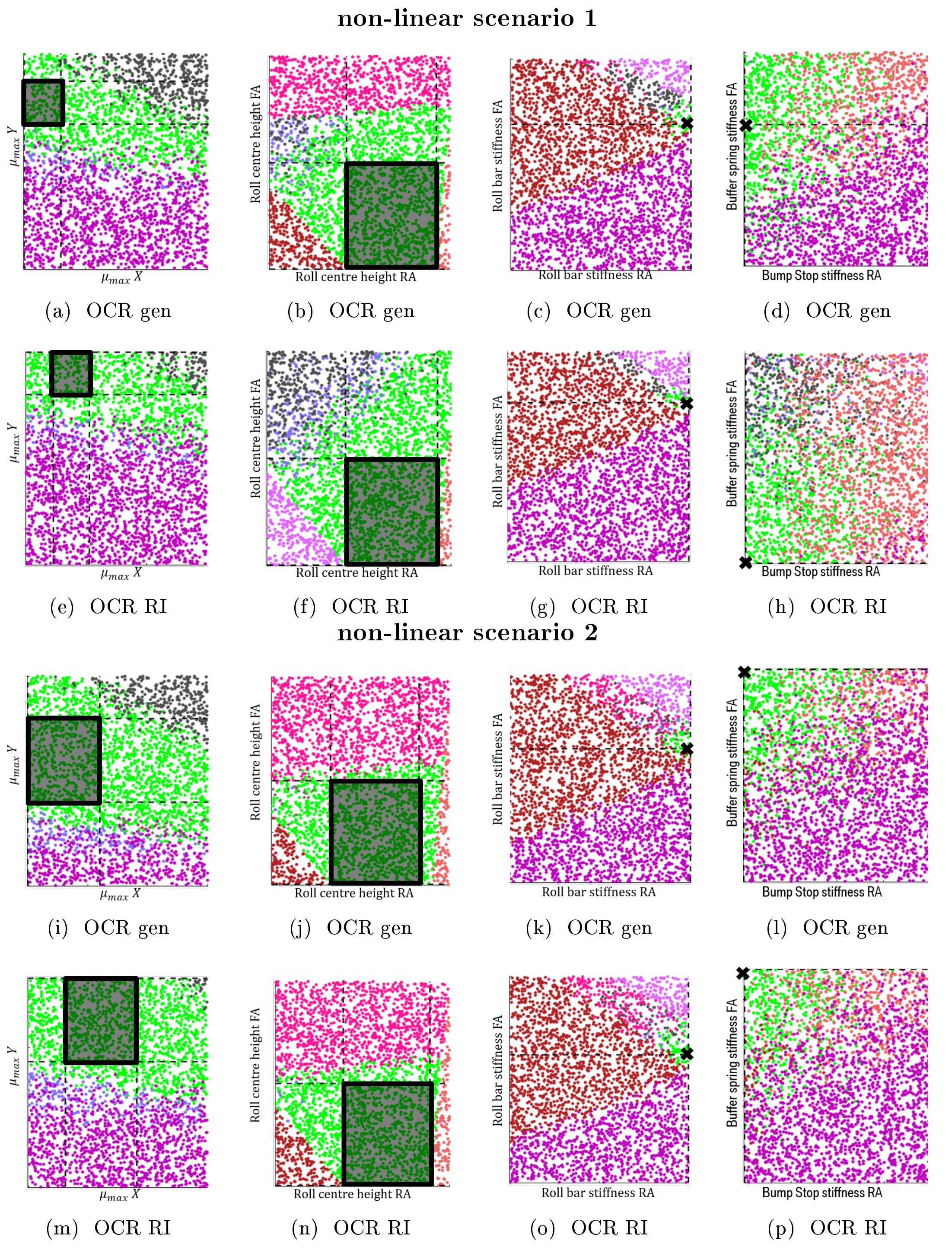


Figure 7.7 Non-linear chassis design problem. (a)-(h)/(i)-(p) an Solution Space (black box) that fulfils the robustness requirements for scenario 1/2 (see table 3.3) while fulfilling the relaxed performance constraints for scenario 1/2 (see table 3.5). The constraints are relaxed with respect to the target function (5.4). The weighting factors are either generic (gen) or according to RI.

In order to decrease the complexity in this section we only compute Optimal Constraint Relaxation for SCS with generic weighting factors.

7.4.2. Comparison of different algorithms to compute Optimal Constraint Relaxation for Solution-Compensation Spaces

In the following, the Bisection Algorithm, the Static Algorithm, and the Shifting Algorithm to compute Optimal Constraint Relaxation for Solution-Compensation Spaces are compared with respect to computational effort and target value. In the end, a conclusion is drawn which algorithm fits the chassis design problem best. In tables 7.12 and 7.13 an overview of the results for scenario 1 and 2 of the different algorithms is given. These results are used as a basis for the following analysis.

Computational Effort

The main computational effort for all the algorithms comes from the projection step. Depending on the algorithm, the system needs to project multiple times in each iteration step. Formulas to estimate the computational effort for the different algorithms are given in section 6.2.

The runtime for the Bisection Algorithm is by far the lowest of the three algorithms introduced. This was to be expected since the Bisection Algorithm is used as a pre-optimisation in the other algorithms. Table 7.12 shows that for scenario 1 the Static Algorithm is the slowest. This is unexpected since the number of dofs is lower and the search area is smaller compared to those of the Shifting Algorithm. In order to restrict the search area we introduced an additional constraint. Unfortunately, this configuration led to an increased number of iteration steps for scenario 1. Table 7.13 shows that in scenario 2 this is different. Here we can see that the Shifting Algorithm takes a longer time as well as more iteration steps to converge.

Comparing the runtime of any Optimal Constraint Relaxation for Solution-Compensation Spaces algorithm ($50 - 2500$ s) to the Solution-Compensation Space algorithms ($1 - 45$ s) or Optimal Constraint Relaxation for Solution Space algorithms (0.03 s), we can see that it is significantly higher. This was to be expected since for Solution-Compensation Space only a single projection is required, while for Optimal Constraint Relaxation for Solution Spaces no projection is required at all.

Performance with respect to the target function

Figures 7.8 (a)-(f)/ (g)-(l) visualise the results of Optimal Constraint Relaxation for Solution-Compensation Spaces for scenario 1/2 (high performance/ standard tyre, see table 3.3). Only the early-decision parameter dimensions are visualised. The black boxes are resulting Solution-Compensation Spaces. The constraints are relaxed according to the results denoted in tables 7.12, 7.13. Note that green points visualise good designs while any other colour represents bad designs. The results for the Bisection Algorithm and the Static Algorithm look similar since the resulting Solution-Compensation Space is at the exact same position. The difference is that the constraints are relaxed slightly differently. The shifting algorithm changes the position of the Solution-Compensation Spaces with respect to a single variable, the maximum friction coefficient in the tyre longitudinal axle $\mu_{\max,X}$. In both scenarios it increases the value for the friction coefficient, which enables it to lower the necessary relaxation for the maximum lateral acceleration z_{a_y}. In scenario 1 this also enables it to lower the maximum steering angle factor z_δ to zero. In scenario 2 it enables significantly lower relaxation of the performance with respect to z_δ as well as the minimal vertical tyre force z_{F_z}. Note that even though in scenario 2 the target values for $Z_{z_{\mathrm{RA}}}$ and $Z_{z_{\mathrm{RA}}}$ are relaxed as a result of the Shifting Algorithm, the overall value for the target function $g_gen(\Delta F_c)$ is smaller compared to the other algorithms.

When we compare the results of Optimal Constraint Relaxation for Solution Spaces (table 7.10) and Optimal Constraint Relaxation for Solution-Compensation Spaces (tables 7.12, 7.13) we can see that the necessary amount of relaxation with respect to the performance constraints is decreased significantly. For problem statement 1, the value for the target function with generic weight factors lies at 2.05×10^{-7} or lower for Optimal Constraint Relaxation for Solution-Compensation Spaces, while the best result obtained with Optimal Constraint Relaxation for Solution Spaces was 3.60×10^{-5}. On the one hand, this was to be expected, due to the existence of late-decision variables in Optimal Constraint Relaxation for Solution-Compensation Spaces. On the other hand, the magnitude of the difference is surprisingly high. Comparing the best results of both methods, the value of the target function can be decreased by a factor of 9.0×10^{-5}. Since the Bisection Algorithm decreases the relaxation by scaling Δf_c, we can see that its results are 7.54% / 72.6% of the result obtained with Optimal Constraint Relaxation for Solution Spaces for scenario 1 / 2.

Note that we do not consider scenario 1 (high performance tyre, see table 3.3) any more. This is due to the fact that we already showed that (see section 7.2) for scenario 1 the Solution-Compensation Space approach is sufficient to fulfil all non-relaxed performance requirements as well as the robustness requirements. Hence, when we apply Optimal Constraint Relaxation for Solution-Compensation Spaces, the result is that no relaxation $\Delta f_c = 0$ is required.

Conclusion

Depending on the problem statement, a different algorithm to compute Optimal Constraint Relaxation for Solution-Compensation Spaces is needed. For applications with linear constraints and low computational capacity, the Bisection Algorithm with generic weighting factors is usually recommended. It is very efficient, can be applied to any problem with linear performance constraints and offers results, which are, compared to Optimal Constraint Relaxation for Solution Spaces, always superior with respect to minimizing the target function. For applications with linear constraints and high computational capacity the Shifting Algorithm is recommended. It has by far the best results of all newly introduced Optimal Constraint Relaxation algorithms with respect to minimising the target function. The price you pay is a high computational effort. Note that even though the static algorithm seems like a bad choice now, for future applications the performance of the Static Algorithm could be improved by providing the analytical gradient of the additional constraint (box inside the sphere see figure 6.5) to the optimisation algorithm.

For the chassis design problem discussed here, the Shifting Algorithm to compute Optimal Constraint Relaxation for Solution-Compensation Spaces with linear performance constraints is recommended. With a runtime of about one hour, the algorithm takes quite long but yields an Solution-Compensation Space, which is also optimised with respect to its position. Hence, the Shifting Algorithm yields significantly better results with respect to the value of the target function. For the chassis design it is of utmost importance to decrease all relaxation of the target values to their absolute minimum, since decreasing a target value means decreasing the quality of the vehicle.

Note that even though we only introduced algorithms to compute Optimal Constraint Relaxation for Solution-Compensation Space for systems with linear constraints it is possible to apply any of these algorithms to non-linear problems by linearising the system in each iteration step similarly to the idea introduced for Optimal Constraint Relaxation for Solution Space. Since this idea does not work for arbitrary non-linear systems, it is important to check the accuracy of the result. This can be done by Monte Carlo sampling.

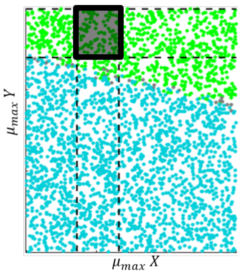
generic weighting	scenario 1		
	Bisection A.	Static A.	Shifting A.
runtime [s]	52.5	1952.8	386.8
iteration steps	15	15+578	15+100
size $\frac{\mu(\Omega)}{\mu(\Omega_1)}$ $[-]$	1.00	1.00	1.00
Δz_{α} $\left[\mathrm{rad}/\frac{\mathrm{m}}{\mathrm{s}^2}\right]$	x	x	x
Δz_{a_y} $\left[\mathrm{m}/\mathrm{s}^2\right]$	3.70×10^{-3}	3.90×10^{-3}	x
Δz_{F_z} [N]	3.19×10^{-2}	3.19×10^{-2}	3.19×10^{-2}
$\Delta z_{z_{RA}}$ [m]	x	x	x
$\Delta z_{z_{FA}}$ [m]	x	x	x
Δz_{Φ} [rad]	x	x	x
Δz_{C} $[-]$	x	x	x
Δz_{δ} $[-]$	8.08×10^{-4}	2.42×10^{-4}	x
$g_{\mathrm{gen}}(\Delta f_c)$	2.05×10^{-7}	1.92×10^{-7}	3.24×10^{-9}

Table 7.12 Linearised chassis design problem. Scenario 1. Generic weighting factors. Overview of the results of OCR for SCSs with generic weighting factors for the linearised chassis design problem with respect to runtime (on an Intel Xeon E5-2650 v2 @ 2.6 GHz processor), size, constraint relaxation, and value of the target function from scenario 1.

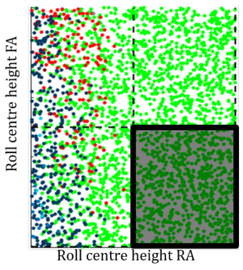
generic weighting	scenario 2		
	Bisection A.	Static A.	Shifting A.
runtime [s]	54.0	2474.9	3762.8
iteration steps	15	15+719	15+1470
size $\frac{\mu(\Omega)}{\mu(\Omega_1)}$ [$-$]	1.00	1.00	1.00
Δz_α $\left[\text{rad}/\frac{\text{m}}{\text{s}^2}\right]$	x	x	x
Δz_{a_y} $\left[\text{m}/\text{s}^2\right]$	0.116	0.120	5.69×10^{-2}
Δz_{F_z} [N]	1.77	1.77	0.176
$\Delta z_{z_{RA}}$ [m]	x	x	2.16×10^{-7}
$\Delta z_{z_{FA}}$ [m]	x	x	2.42×10^{-7}
Δz_Φ [rad]	x	9.93×10^{-6}	x
Δz_C [$-$]	2.33×10^{-3}	x	1.54×10^{-5}
Δz_δ [$-$]	3.86×10^{-2}	1.32×10^{-2}	5.80×10^{-3}
$g_{\text{gen}}(\Delta f_c)$	2.46×10^{-4}	1.91×10^{-4}	4.09×10^{-5}

Table 7.13 Linearised chassis design problem. Scenario 2. Generic weighting factors. Overview of the results of OCR for SCSs with generic weighting factors for the linearised chassis design problem with respect to runtime (on an Intel Xeon E5-2650 v2 @ 2.6 GHz processor), size, constraint relaxation, and value of the target function from scenario 2.

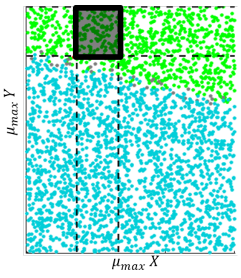
linear scenario 1



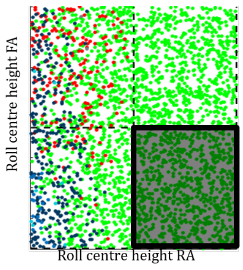
(a) OCR gen



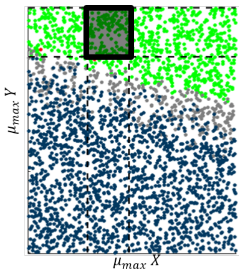
(b) OCR gen



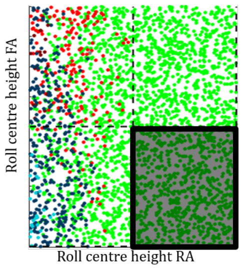
(c) Static A. gen



(d) Static A. gen

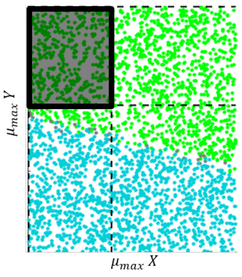


(e) Shifting A. gen

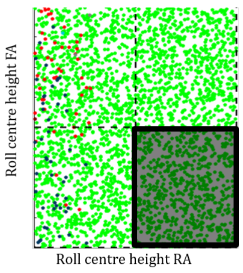


(f) Shifting A. gen

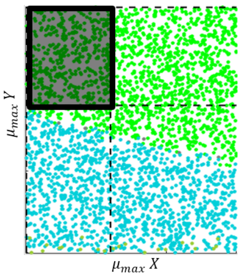
linear scenario 2



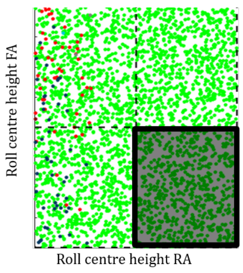
(g) OCR gen



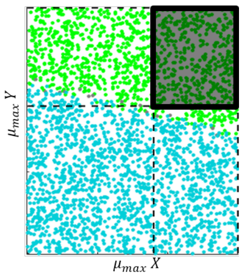
(h) OCR gen



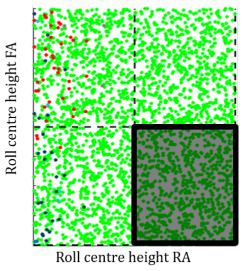
(i) Static A. gen



(j) Static A. gen



(k) Shifting A. gen



(l) Shifting A. gen

Figure 7.8 Linear chassis design problem. (a)-(b) & (g)-(h)/(c)-(d) & (i)-(j)/(e)-(f) & (k)-(l) The early-variable design spaces of SCSs (black boxes) optimised by the Bisection/Static/Shifting Algorithm. These SCSs fulfil the robustness requirements for scenario 1 & 2 (see table 3.3) while fulfilling the relaxed performance constraints for scenario 1 (see table 7.12). The constraints are relaxed with respect to the target function (5.4). The weighting factors are generic (gen).

Chapter 8
Conclusion

The following chapter refers back to the stated aims and research questions and provides a critical reflection on what has been achieved. In addition, the challenges concerning the newly introduced methods to compute set-based designs, which where encountered when applying these to the chassis design problem, are discussed. In the end, an outlook is given in which the approaches introduced in this thesis as well as the key findings, are summarised, the main conclusions are drawn and ideas for future research work are given.

8.1. Critical Reflection

In chapter 1 the necessity for providing larger Solution Spaces in early development stages of complex products was presented. A literature review was conducted in chapter 2 and aims and research questions were derived in chapter 3. In order to solve the research questions, new methods to compute set-based designs were introduced in chapters 4, 5, and 6. In chapter 7 the newly developed methods were applied to a chassis design problem, which was not solvable with state of the art methods. In the following, we review the research questions and discuss to which extend they were fulfilled.

Aim 1: Provide a method, which computes box-shaped Solution Spaces with significantly increased interval sizes (compared to classical Solution Spaces [83]) for crucial design variables. Solution-Compensation Spaces were introduced in chapter 4 as a new method to derive box-shaped designs. Solution-Compensation Spaces build on the fact that in most industrial applications not all parameters are designed simultaneously. If that is the case, the variables, which are designed later in the process (late-decision variables) can be used to compensate for larger permissible interval sizes. We validated the newly introduced method by applying it to an industrial example the chassis design problem (see section 7.2). We increased all early-variable interval sizes compared to the classical Solution Space approach and more than doubled the volume of the resulting box (see tables 3.4, 7.8). Hence, aim 1 is fulfilled: a new method with the specified properties was developed. With the caveat that the new method is only applicable to sequential design processes.

Aim 2: Provide a method, which computes a minimal set of changes, which need to be applied to the performance constraints such that any Solution Space problem statement with robustness constraints becomes feasible. Optimal Constraint Relaxation for Solution Spaces was introduced in chapter 5 as a new method to derive a minimal set of changes to the performance constraints to make any Solution Space problem statement feasible. The target function, which is minimised is the weighed sum of all relaxations with respect to the performance goals. We introduced an algorithm for problem statements with linear- and monotonous non-linear performance functions. The newly introduced method always converges towards the global optimum. Hence, aim 2 is fulfilled: a new method with the specified properties was developed. With the caveat that the new method is not applicable to arbitrarily non-linear performance functions. In addition, the result of the algorithm is significantly dependent on the chosen target function.

Aim 3: Provide a method, which combines the advantages of Solution-Compensation Spaces with the advantages of Optimal Constraint Relaxation for Solution Spaces. Thus making any Solution Space problem statement with robustness constraints feasible while deploying the increasing flexibility gained from applying Solution-Compensation Spaces. Optimal Constraint Relaxation for Solution-Compensation Spaces was introduced in chapter 6. It works similarly to Optimal Constraint Relaxation for Solution Spaces, with the difference that it uses late-decision variables to further decrease the required relaxation. Hence, Optimal Constraint Relaxation for Solution-Compensation Spaces is only applicable to a sequential design processes. We validated the newly introduced method by applying it to the chassis design problem. Compared to Optimal Constraint Relaxation for Solution Spaces we decreased the relaxation according to the target function (with generic weighting factors, see 5.4) by 99.9% for scenario 1 and by 91.2% for scenario 2 (see tables 7.10, 7.12, and 7.13). Hence, aim 3 is fulfilled: a new method with the specified properties was developed. With the caveat that the new method is only applicable to sequential design processes with linear (or linearised) performance functions. In addition, the result of the algorithm is significantly dependent on the chosen target function.

Note that both Optimal Constraint Relaxation for Solution Spaces and Optimal Constraint Relaxation for Solution-Compensation Spaces are also applicable if no robustness constraints are required. In this case the result will be a single feasible design and not a set-based design.

Summary: It is shown that the methods introduced eliminate the deficiencies of the state of the art algorithms pointed out in chapter 3 and hence, fulfil the objectives of this thesis. Which approach is recommended depends on the problem statement.

Figure 8.1 gives a recommendation on which Solution Space approach to apply depending on the type of design problem. In case that all variables of the considered system are designed simultaneously, the classical Solution Space approach is recommended (see chapter 2). In case the variables of the considered system are designed sequentially, the Solution-Compensation Space approach is applicable (see chapter 4). It enables larger intervals for the early-decision variables and is hence recommended over the classical Solution Space approach. In case no solution is found or the resulting Solution Space/Solution-Compensation Space is too small, Optimal Constraint Relaxation for Solution Spaces/Solution-Compensation Spaces can be applied to determine a set of relaxed constraints such that the problem becomes feasible (see chapter 5 and 6). Optimal Constraint Relaxation for Solution-Compensation Spaces yields solutions with less relaxation compared to Optimal Constraint Relaxation for Solution Spaces and is hence recommended in the sequential design process.

8.2. Outlook

Even though we covered a lot of the deficiencies of state of the art methods concerning Solution Spaces many questions remain unanswered, which can be addressed in the future:

- How can Solution-Compensation Spaces be optimised if a minimal interval size for certain parameters is required (robustness constraints)? Currently, we are only able to optimise the Solution-Compensation Space according to a size measure (e.g. volume).

- Is it possible to compute Optimal Constraint Relaxation for Solution Spaces/Solution-Compensation Spaces for arbitrarily non-linear constraint functions? So far monotone functions are required for Optimal Constraint Relaxation for Solution Spaces

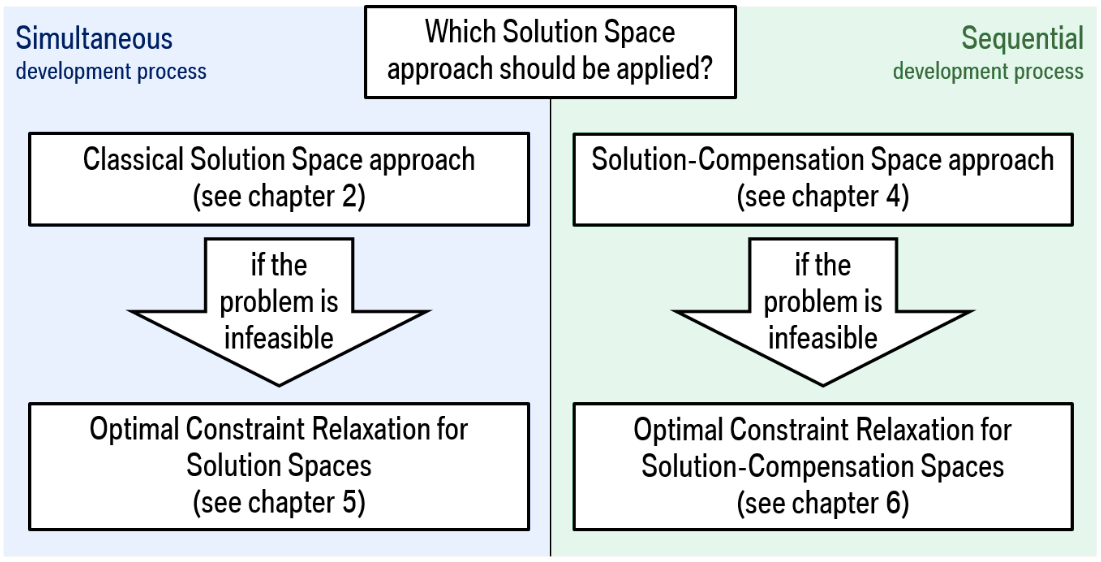

Simultaneous
development process
Which Solution Space approach should be applied?
Sequential
development process
Classical Solution Space approach (see chapter 2)
Solution-Compensation Space approach (see chapter 4)
if the problem is infeasible
if the problem is infeasible
Optimal Constraint Relaxation for Solution Spaces (see chapter 5)
Optimal Constraint Relaxation for Solution-Compensation Spaces (see chapter 6)

Figure 8.1 Recommendation on which Solution Space approach to apply depending on the design problem.

and linear functions are required for Optimal Constraint Relaxation for Solution-Compensation Spaces.

- How can we modify the target function in order to improve the result of Optimal Constraint Relaxation for Solution Spaces/Solution-Compensation Spaces? User defined parametrisation was introduced in 5.4. One idea would be to train a machine learning algorithm based on user evaluation.
- In which industrial applications can Solution-Compensation Spaces and Optimal Constraint Relaxation be beneficial, apart from chassis design?

As shown in chapter 7, the methods introduced in this thesis, bring great advantages with respect to robustness and flexibility during the chassis design process. From a mathematical standpoint these methods can be applied to any development process where design parameters shall be optimised and objective requirements can be formulated. I firmly believe that in the future Solution-Compensation Spaces and Optimal Constraint Relaxation will be applied in different fields of the vehicle industry such as crash design and engine mount systems design. Even applications in other industries where complex products are designed, such as aircraft or aerospace industry are possible.

List of acronyms

ANN	Artificial Neural Network
DS	Design Space
FME	Fourier-Motzkin Elimination
FTC	Functional Tyre Characteristic
LPP	Linear Programming Problem
MC	Monte-Carlo
NINF	Number of Infeasibilities
OCR	Optimal Constraint Relaxation
PSO	Particle Swarm Optimisation
QSSC	Quasi-Steady State Cornering
RAST	Ramp Steering
RI	Rating Index for vehicle dynamics
RR	Redundancy Removal
RSM	Response-Surface Metamodel
SCS	Solution-Compensation Space
SINF	Sum of Infeasibilities
SLVF	Sum of the lengths of the feasibility vectors
SS	Solution Space
SSA	Stochastic Solution Space Algorithm
SSCSA	Stochastic Solution-Compensation Space Algorithm
SVM	Support Vector Machine
SWD	Sine with Dwell
VT	Vertex Tracking
bad design	design that does not fulfil all performance requirements
dof	degrees of freedom
good design	design that fulfils all performance requirements

List of variables

A	System matrix A of the early-decision variable space. $A \in \mathbb{R}^{m \times p}$. See equation (4.3)
B	System matrix B of the late-decision variable space. $B \in \mathbb{R}^{m \times q}$. See equation (4.3)
d	Total number of variable dimensions. (early and late-decision variable dimensions $d = p + q$)
$Delta f_c$	Relaxation of the performance constraints $f(x) \leq f_c + Delta f_c$.
f_c	Minimal threshold to fulfil the performance requirement: $f(x) \leq f_c$.
G	Extended system matrix A and B including the design space constraints. $G \in \mathbb{R}^{m_t \times d}$. See equation (4.5)
g_c	Extended constraints vector f_c including design space constraints. $g_c \in \mathbb{R}^{m_t}$. See equation (4.5)
I	Identity Matrix
m	Number of performance constraints.
m_t	Total number of constraints. $m_t = m + 2d$. (performance and design space constraints)
m_{ti}	Number of constraints after eliminating a total of i dimensions with the FME.
m_b	Number of back facing constraints in in the direction $-e_i$.
m_f	Number of front facing constraints in in the direction $-e_i$.
m_p	Number of parallel constraints in the direction $-e_i$.
p	Number of early-decision variable dimensions.
q	Number of late-decision variable dimensions.
x_c	Centre of the box-shaped Solution Space

List of Figures

List of Tables

Bibliography

[1] M. L. Ajspur and R. M. Jensen. “Using Fourier-Motzkin-Elimination to Derive Capacity Models of Container Vessels”. In: *IT University Technical Report Series* (2017).

[2] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. “The quickhull algorithm for convex hulls”. In: *ACM Transactions on Mathematical Software* 22.4 (1996), pp. 469–483. ISSN: 00983500. DOI: 10.1145/235815.235821.

[3] I. A. Basheer and M. Hajmeer. “Artificial neural networks: fundamentals, computing, design, and application”. In: *Journal of Microbiological Methods* 43.1 (2000), pp. 3–31. DOI: 10.1016/S0167-7012(00)00201-3.

[4] M. R. Bonyadi and Z. Michalewicz. “Particle swarm optimization for single objective continuous space problems: a review”. In: *ECJ Evolutionary Computation Journal* 25.1 (2017), pp. 1–54.

[5] S. Boyd and L. Vandenberghe. *Convex optimization.* Cambridge University Press, 2004. ISBN: 978-0521833783.

[6] B. Chazelle. “An optimal convex hull algorithm in any fixed dimension”. In: *Discrete & Computational Geometry* 10.4 (1993), pp. 377–409. DOI: 10.1007/BF02573985.

[7] J. W. Chinneck. *Feasibility and Infeasibility in Optimization.* Vol. 118. 2008. ISBN: 978-0-387-74931-0. DOI: 10.1007/978-0-387-74932-7.

[8] J. W. Chinneck. “The constraint consensus method for finding approximately feasible points in nonlinear programs”. In: *INFORMS Journal on Computing* 16.3 (2004), pp. 255–265. DOI: 10.1287/ijoc.1030.0046.

[9] K. L. Clarkson. “More output-sensitive geometric algorithms”. In: *Proc. 35th Annual Symposium on Foundations of Computer Science.* IEEE. 1994, pp. 695–702. DOI: 10.1109/SFCS.1994.365723.

[10] I. Cuevas Salazar, F. Duddeck, and L. Song. “Small overlap assessment for early design phases based on vehicle kinematics”. In: *International Journal of Crashworthiness* (2019), pp. 1–16. DOI: 10.1080/13588265.2018.1514689.

[11] G. B. Dantzig and B. C. Eaves. *The Basic.* Stanford University Press, 2003, p. 255.

[12] S. Das. *A brief note on estimates of binomial coefficients.* http://page.mi.fu-berlin.de/shagnik/notes/binomials.pdf. Accessed: 2019-05-09.

[13] M. Daub and F. Duddeck. “Complex systems design under non-reducible lack-of-knowledge uncertainties”. In: *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems: Part B. Mechanical Engineering* (2019).

[14] N. R. Draper and H. Smith. *Applied regression analysis.* Vol. 326. John Wiley & Sons, 2014. ISBN: 978-0471170822.

[15] F. Duddeck and E. Wehrle. “Recent advances on surrogate modeling for robustness assessment of structures with respect to crashworthiness requirements”. In: *10th European LS-DYNA conference, Würzburg, Germany* (2015).

[16] M. Eichstetter. *Design of vehicle system dynamics using solution spaces.* PhD thesis, Technische Universität Berlin, Berlin, Germany, 2017.

[17] M. Eichstetter. “Solution spaces for damper design in vehicle dynamics”. In: *5th International Munich Chassis Symposium.* Springer. 2014, pp. 107–132. DOI: 10.1007/978-3-658-05978-1_10.

[18] M. Eichstetter, S. Müller, and M. Zimmermann. “Product Family Design With Solution Spaces”. In: *Journal of Mechanical Design* 137.12 (2015). ISSN: 1050-0472. DOI: 10.1115/1.4031637.

[19] S. Erschen, F. Duddeck, M. Gerdts, and M. Zimmermann. “On the optimal decomposition of high-dimensional solution spaces of complex systems”. In: *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems* 4.2 (2017), p. 021008. DOI: 10.1115/1.4037485.

[20] S. Erschen. *Optimal Decomposition of High-Dimensional Solution Spaces for Chassis Design.* PhD thesis, Technische Universität München, Munich, Germany, 2018.

[21] S. Erschen, F. Duddeck, and M. Zimmermann. “Robust Design using classical optimization - Computation of Solution Spaces with application to chassis design”. In: *Proc. Appl. Math. Mech.* 15.1 (2015), pp. 565–566. DOI: 10.1002/pamm.201510272.

[22] J. Fender, F. Duddeck, and M. Zimmermann. “Direct computation of solution spaces”. In: *Struct. Multidisc. Optim.* 55.5 (2017), pp. 1787–1796. ISSN: 1615-147X. DOI: 10.1007/s00158-016-1615-y.

[23] J. Fender, F. Duddeck, and M. Zimmermann. “On the calibration of simplified vehicle crash models”. In: *Struct. Multidisc. Optim.* 49 (2013). DOI: 10.1007/s00158-013-0977-7.

[24] J. Fender, L. Graff, H. Harbrecht, and M. Zimmermann. “Identifying key parameters for design improvement in high-dimensional systems with uncertainty”. In: *Journal of Mechanical Design* 136.4 (2014). DOI: 10.1115/1.4026647.

[25] W. W. Finch and A. C. Ward. “Quantified Relations: A Class of Predicate Logic Design Constraints Among Sets of manufacturing, Operating, and Other Variations”. In: *Proceedings of the 8th International Conference on Design Theory and Methodology, Irvine, California.* 1996.

[26] J. Fox. *Applied regression analysis and generalized linear models.* Sage Publications, 2015. ISBN: 978-1452205663.

[27] K. Fukuda. *Lecture notes on Polyhedral Computation.* ETH Zürich, Switzerland. 2014.

[28] K. Fukuda, B. Gärtner, and M. Szedlák. “Combinatorial redundancy detection”. In: *Annals of Operations Research* (2014), pp. 1–19. DOI: 10.4230/LIPIcs.SOCG.2015.315.

[29] G. Fung, S. Sandilya, and R. B. Rao. “Rule Extraction from Linear Support Vector Machines”. In: *Proc 11th ACM SIGKDD int. Conf. on Knowledge Discovery in Data Mining, Chicago, Illinois, USA* (2005), pp. 32–40. DOI: 10.1145/1081870.1081878.

[30] L. Graff. “A stochastic algorithm for the identification of solution spaces in high-dimensional design spaces”. PhD thesis. University of Basel, Basel, Germany, 2013.

[31] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. “Support vector machines”. In: *IEEE Intelligent Systems and their applications* 13.4 (1998), pp. 18–28. DOI: 10.1109/5254.708428.

[32] B. Heißing, M. Ersoy, and S. Gies. “Fahrdynamik”. In: *Fahrwerkhandbuch*. Wiesbaden: Vieweg+Teubner, 2011, pp. 37–154. DOI: 10.1007/978-3-8348-8168-7_2.

[33] J.-L. Imbert. “Fourier’s Elimination: Which to Choose?” In: *PPCP* (1993), pp. 117–129.

[34] International Organization for Standardization. “ISO 19365:2016 - Passenger cars – Validation of vehicle dynamic simulation – Sine with dwell stability control testing”. In: *Passenger cars. Caravans and light trailers* (2016).

[35] International Organization for Standardization. “ISO 4138:2012 - Passenger cars – Steady-state circular driving behaviour – Open-loop test methods”. In: *Passenger cars. Caravans and light trailers* (2012).

[36] L. Kettner and E. Welzl. “Contour edge analysis for polyhedron projections”. In: *Geometric Modeling: Theory and Practice*. Springer, 1997, pp. 379–394. DOI: 10.1007/978-3-642-60607-6_25.

[37] N. Khan, S. Inayatullah, M. Imtiaz, and F. H. Khan. “New artificial-free phase 1 simplex method”. In: *arXiv preprint arXiv:1304.2107* (2013).

[38] H. M. Kim, N. F. Michelena, P. Y. Papalambros, and T. Jiang. “Target cascading in optimal system design”. In: *Journal of Mechanical Design* 125.3 (2003), pp. 474–480. DOI: 10.1115/1.1582501.

[39] S. Königs and M. Zimmermann. “Resolving conflicts of goals in complex design processes – application to the design of engine mount systems”. In: *7th International Munich Chassis Symposium*. Springer Fachmedien Wiesbaden, 2017. DOI: 10.1007/978-3-658-14219-3_14.

[40] P. Kvasnicka, G. Prokop, M. Dorle, A. Rettinger, and H. Stahl. “Dürchgangige Simulationsumgebung zur Entwicklung und Absicherung von fahrdynamischen Regelsystemen”. In: *VDI Berichte* 1967.1 (2006), p. 387.

[41] V. A. Lange, J. Fender, and F. Duddeck. “Relaxing high-dimensional constraints in the direct solution space method for early phase development”. In: *Optimization and Engineering* (2018), pp. 1–29. DOI: 10.1007/s11081-018-9381-x.

[42] V. A. Lange, J. Fender, L. Song, and F. Duddeck. “Early phase modeling of frontal impacts for crashworthiness: From lumped mass–spring models to Deformation Space Models”. In: *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering* (2018). DOI: 10.1177/0954407018814034.

[43] C. Larman and V. R. Basili. “Iterative and incremental developments: a brief history”. In: *Computer* 36.6 (2003), pp. 47–56. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1204375.

[44] M. Lehar and M. Zimmermann. “An inexpensive estimate of failure probability for high-dimensional systems with uncertainty q”. In: *Structural Safety* 36-37 (2012), pp. 32–38. ISSN: 0167-4730. DOI: 10.1016/j.strusafe.2011.10.001.

[45] U. Lindemann. *Handbuch Produktentwicklung.* Carl Hanser Verlag GmbH Co KG, 2016. ISBN: 978-3446445185.

[46] I. J. Lustig, R. E. Marsten, and D. F. Shanno. “Interior point methods for linear programming: Computational state of the art”. In: *ORSA Journal on Computing* 6.1 (1994), pp. 1–14. DOI: 10.1287/ijoc.6.1.1.

[47] S. Marsland. *Machine learning: an algorithmic perspective.* Chapman and Hall/CRC, 2011. ISBN: 978-1439889213.

[48] *Math World, Bisection.* http://mathworld.wolfram.com/Bisection.html. Accessed: 2019-05-09.

[49] M. Münster, M. Lehner, and D. Rixen. “Vehicle steering design using solution spaces for decoupled dynamical subsystems”. In: *International Conference on Noise and Vibration Engineering, ISMA, Leuven, Belgium.* 2014.

[50] N. J. Nagelkerke. “A note on a general definition of the coefficient of determination”. In: *Biometrika* 78.3 (1991), pp. 691–692.

[51] H. Pacejka. *Tire and vehicle dynamics.* Elsevier, 2005. ISBN: 978-0080543338.

[52] J. H. Panchal, M. G. Fernández, J. K. Allen, C. J. Paredis, and F. Mistree. “An interval-based focalization method for decision-making in decentralized, multi-functional design”. In: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference.* American Society of Mechanical Engineers. 2005, pp. 413–426. DOI: 10.1115/DETC2005-85322.

[53] C. Paredis, J. Aughenbaugh, R. Malak, and S. Rekuc. “Set-based design: a decisiontheoretic perspective”. In: *Proc. frontiers in design & simulation research 2006 workshop.* 2006, pp. 1–25.

[54] K. Pohl. *Requirements engineering: fundamentals, principles, and techniques.* Springer, Berlin Heidelberg, Germany, 2010. ISBN: 978-3-642-12577-5.

[55] A. J. Qureshi, J.-Y. Dantan, Jérôme, and R. Bigot. "Set-based design of mechanical systems with design robustness integrated". In: *International Journal of Product Development* 19.1-3 (2015), pp. 64–89. DOI: 10.1504/IJPD.2014.060037.

[56] S. J. Rekuc, J. M. Aughenbaugh, M. Bruns, and C. J. Paredis. "Eliminating design alternatives based on imprecise information". In: *SAE Transactions* (2006), pp. 208–220. DOI: 10.4271/2006-01-0272.

[57] C. M. Rocco, J. A. Moreno, and N. Carrasquero. "Robust design using a hybrid-cellular-evolutionary and interval-arithmetic approach: a reliability application". In: *Reliability Engineering & System Safety* 79.2 (2003), pp. 149–159. ISSN: 09518320. DOI: 10.1016/S0951-8320(02)00226-0.

[58] A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998. ISBN: 978-0471982326.

[59] R. Shao. "Developing a methodology to increase communality for chassis components within the vehicle dynamics design". Master Thesis. Karlsruhe Institute of Technology, Germany, 2017.

[60] Y. Shi and R. Eberhart. "A modified particle swarm optimizer". In: *Evolutionary Computation Proceedings, World Congress on Computational Intelligence, Anchorage, AK, USA.* IEEE. 1998, pp. 69–73. DOI: 10.1109/ICEC.1998.699146.

[61] T. W. Simpson, J. Poplinski, P. N. Koch, and J. K. Allen. "Metamodels for computer-based engineering design: survey and recommendations". In: *Engineering with Computers* 17.2 (2001), pp. 129–150. DOI: 10.1007/PL00007198.

[62] D. J. Singer, N. Doerry, and M. E. Buckley. "What Is Set-Based Design?" In: *Naval Engineers Journal* 121.4 (2009), pp. 31–43. DOI: 10.1111/j.1559-3584.2009.00226.x.

[63] D. K. Sobek II, A. C. Ward, and J. K. Liker. "Toyota's principles of set-based concurrent engineering". In: *MIT Sloan Management Review* 40.2 (1999), p. 67.

[64] L. Song. "Commonality Design of Vehicle Architectures Concerning Crashworthiness Using Solution Spaces." PhD thesis. Technische Universität München, Munich, Germany, 2019.

[65] L. Song, M. Pabst, F. Duddeck, and J. Fender. "A simplified model for barrier–vehicle interaction in a rear crash for early phase development and solution spaces". In: *International Journal of Crashworthiness* 23.5 (2018), pp. 507–520. DOI: 10.1080/13588265.2017.1350091.

[66] R. Storn and K. Price. "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces". In: *Journal of Global Optimization* 11.4 (1997), pp. 341–359. DOI: 10.1023/A:1008202821328.

[67] F. Stutz. "Computation of Optimized Tolerances for Uncertainty in Systems Design with Early and Late Decision Variables". Master Thesis. Faculty of Science of the University of Basel, Switzerland, 2017.

[68] R. E. Swaney and I. E. Grossmann. "An index for operational flexibility in chemical process design. Part I: Formulation and theory". In: *AIChE Journal* 31.4 (1985), pp. 621–630. DOI: 10.1002/aic.690310412.

[69] M. Szedlak. "Redundancy in Linear Systems: Combinatorics, Algorithms and Analysis". PhD thesis. ETH Zurich, Switzerland, 2017.

[70] I. C. Trelea. "The particle swarm optimization algorithm: convergence analysis and parameter selection". In: *Information processing letters* 85.6 (2003), pp. 317–325.

[71] M. Trzesniowski. *Fahrwerk.* Springer, 2017. DOI: https://doi.org/10.1007/978-3-658-15545-2.

[72] F. Vallentin and A. Gundert. *Das Eliminationsverfahren von Fourier und Motzkin.* http://www.mi.uni-koeln.de/opt/wp-content/uploads/2014/07/fourier-motzkin.pdf. University of Cologne, Germany, Accessed: 2019-05-09. 2014.

[73] M. Vogt, F. Duddeck, H. Harbrecht, F. Stutz, M. Wahle, and M. Zimmermann. "Computing solution-compensation spaces using an enhanced Fourier-Motzkin algorithm". In: *PAMM Proceedings in Applied Mathematics and Mechanics* (2018). DOI: 10.1002/pamm.201800103.

[74] M. Vogt, F. Duddeck, M. Wahle, and M. Zimmermann. "Optimising Tolerance to Uncertainty in Systems Design with Early- and Late-Decision Variables". In: *IMA Journal of Management Mathematics dpy003* (2018). DOI: 10.1093/imaman/dpy003.

[75] A. C. Ward. *A theory of quantitative inference applied to a mechanical design compiler.* http://hdl.handle.net/1721.1/6977. Accessed: 2019-05-09. 1989.

[76] A. C. Ward and D. K. Sobek II. *Lean product and process development.* Lean Enterprise Institute, 2014. ISBN: 978-1934109434.

[77] P. J. Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560. DOI: 10.1109/5.58337.

[78] A. Wieland and C. M. Wallenburg. "Dealing with supply chain risks: Linking risk management practices and strategies to performance". In: *International Journal of Physical Distribution & Logistics Management* 42.10 (2012), pp. 887–905. DOI: 10.1108/09600031211281411.

[79] P. Wilmott, S. Howson, S. Howison, and J. Dewynne. *The mathematics of financial derivatives: a student introduction.* Cambridge University Press, 1995. ISBN: 978-0521497893.

[80] Y. Zhang. "A review of: Interior Point Algorithms: Theory and Analysis". In: *IIE Transactions* 31.3 (1999), pp. 275–276. DOI: 10.1080/07408179908969827.

[81] M. Zimmermann. *Lecture notes on methods of product development.* Technische Universität München, Munich, Germany. 2019.

[82] M. Zimmermann, S. Königs, J. Fender, C. Niemeyer, C. Zeherbauer, R. Vitale, and M. Wahle. “On the design of large systems subject to uncertainty”. In: *Journal of Engineering Design* 28.4 (2017), pp. 233–254. DOI: `10.1080/09544828.2017.1303664`.

[83] M. Zimmermann and J. E. von Hoessle. “Computing solution spaces for robust design”. In: *Int. J. Numer. Meth. Engng* 94 (2013), pp. 290–307. DOI: `10.1002/nme.4450`.

[84] M. Zimmermann and M. Wahle. “Solution Spaces for Vehicle Concepts and Architectures”. In: 24th Aachen Colloquium Automobile and Engine Technology, Aachen, Germany, 2015.