TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Flugsystemdynamik

Inclusion of Physical Components in Flight Control Systems Optimization

Dipl.-Ing. Univ. Christopher Thorsten Schropp

Vollständiger Abdruck der von der Fakultät für Maschinenwesen der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender:	apl. Prof. Dr.-Ing. habil. Christian W. M. Breitsamter
Prüfer der Dissertation:	1. Prof. Dr.-Ing. Florian Holzapfel
	2. Prof. Dr. Markus Zimmermann

Die Dissertation wurde am 27.05.2019 bei der Technischen Universität München eingereicht und durch die Fakultät für Maschinenwesen am 29.01.2020 angenommen.

Abstract

Today's development processes for mechatronic products, such as modern flight control systems, are noncontinuous and characterized by tool-chain gaps. Overcoming these gaps requires time, funds, and manual labor and results in additional sources of error and unautomated design loops. The effects are particularly significant in the aviation industry, with its long and expensive development cycles. As such, although this work is tailored to flight control systems, the outcomes can be used to a limited extent in the development processes of any mechatronic product.

This thesis presents a method for bidirectional data transfer between CAD software tools and *Simscape Multibody*, a toolbox from *MathWorks' Simulink*, which allows for *multibody simulation* (MBS). By automatically creating MBS models on the basis of previously transferred CAD model data and changing CAD parameters from *MATLAB*, the approach closes the targeted design loop and enables the entire process to be automated. Bridging this specific tool-chain gap allows the entire functionality of the *MathWorks* suite to be applied to CAD models. As such, CAD model parameters can be included in *MATLAB* optimizations.

Furthermore, the research of this thesis covers the simultaneous optimization of hardware and software parameters. By additionally supplying requirements on the system as an input, the developed framework can only produce optimal system designs that adhere to the specifications. Such a highly automated design process results in much shorter and cheaper development cycles and cutting edge system designs.

Comprehensive MBS models of flight control systems, which are easily generated using the provided approach, allow one to evaluate the handling qualities, and ultimately the flight safety, by accounting for inertia loads that result from aircraft maneuvers. Apart from that, joints can be selected in the CAD tool to be automatically enhanced by models, which were implemented or developed in this thesis to simulate complex nonlinear effects like friction and free play.

Wide segments of the approach presented in this thesis are incorporated into a software product that is now being sold to industrial customers.

Zusammenfassung

Heutige Entwicklungsprozesse für mechatronische Produkte, wie beispielsweise moderne Flugsteuerungssysteme, sind durch viele Toolbrüche charakterisiert und damit nicht durchgängig. Die Überbrückung dieser Brüche erfordert Zeit, Geld und manuelle Arbeit, und resultiert in zusätzlichen Fehlerquellen sowie nicht automatisierten Design-Zyklen. In der Luftfahrtindustrie mit ihren langen und teuren Entwicklungszyklen sind diese Effekte besonders ausgeprägt. Obwohl sich diese Dissertation auf Flugsteuerungssysteme konzentriert, können die Methoden in begrenztem Umfang in den Entwicklungsprozessen eines jeden mechatronischen Produkts angewandt werden.

In dieser Dissertation wird eine Methode für den bidirektionalen Datentransfer zwischen CAD-Software-Tools und *Simscape Multibody* vorgestellt, einer Toolbox von *MathWorks' Simulink*, die Mehrkörpersimulation ermöglicht. Die automatische Erstellung von Mehrkörpersimulations-Modellen auf Basis zuvor übertragener CAD-Modelldaten und die Änderung von CAD-Parametern aus *MATLAB* schließt die anvisierte Konstruktionsschleife und ermöglicht die Automatisierung des gesamten Prozesses. Die Überbrückung dieser spezifischen Lücke in der Tool-Kette ermöglicht es, die gesamte Funktionalität der *MathWorks* Suite auf CAD-Modelle anzuwenden. Auf diese Weise können CAD-Modellparameter in *MATLAB* Optimierungen einbezogen werden.

Darüber hinaus umfasst die Forschung dieser Arbeit die gleichzeitige Optimierung von Hard- und Softwareparametern. Durch die zusätzliche Bereitstellung von Anforderungen an das System in Form von Randbedingungen kann das entwickelte Framework nur optimale Systemdesigns erzeugen, die den Spezifikationen entsprechen. Ein derart hochautomatisierter Designprozess führt zu viel kürzeren und kostengünstigeren Entwicklungszyklen und modernsten Systementwicklungen.

Hochdetaillierte Mehrkörpersimulations-Modelle von Flugsteuerungssystemen, die mit dem präsentierten Ansatz einfach zu erzeugen sind, ermöglichen es, die sogenannten *Handling Qualities* und letztlich die Flugsicherheit unter Berücksichtigung von Trägheitslasten mechanischer Komponenten, die durch Flugmanöver entstehen, zu bewerten. Darüber hinaus können im CAD-Tool Verbindungen ausgewählt werden, die automatisch durch Modelle ergänzt werden, welche in dieser Arbeit implementiert oder entwickelt wurden, um komplexe nichtlineare Effekte wie Reibung und freies Spiel zu simulieren.

Weite Teile des in dieser Dissertation vorgestellten Ansatzes sind in ein Softwareprodukt integriert, das nun an Industriekunden verkauft wird.

Danksagung

Diese Dissertation entstand zu größten Teilen während meiner Zeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Flugsystemdynamik der Technischen Universität München, der von Prof. Florian Holzapfel geleitet wird. Ihm gilt zuallererst mein Dank für das in mich gesetzte Vertrauen und damit für die Ermöglichung dieser Arbeit, sowie die stetige Unterstützung, auch über die Zeit am Lehrstuhl hinaus.

Darüber hinaus möchte ich mich insbesondere bei Prof. Markus Zimmermann für den Einsatz als Zweitgutachter, als auch für die Hilfsbereitschaft und den wertvollen Dialog bedanken. Bei Prof. Christian Breitsamter bedanke ich mich herzlich dafür, dass er sich bereiterklärt hat, den Vorsitz der Prüfungskommission zu übernehmen.

Weiterer Dank gilt den vielen Kollegen am Lehrstuhl für Flugsystemdynamik, für die zahlreichen, interessanten und fruchtbaren Gespräche. Extra hervorheben möchte ich hierbei Benedikt Grüter und Johannes Diepolder, die mich stets mit fachlichen Diskussionen und hilfreichen Hinweisen unterstützt haben.

Zuletzt möchte ich mich besonders bei meinen Eltern, meiner Schwester Vivian und meiner Freundin Lena bedanken, die mich über die lange Zeit fortwährend motiviert und bestärkt haben.

München, im April 2019 Christopher Schropp

Table of Contents

List of Figures

List of Tables

Acronyms

3D	three-dimensional
ACE	actuator control electronics
ADC	analog-to-digital converter
AFCS	automatic flight control system
AP	autopilot
API	application programming interface
BFS	Breadth First Search
CAA	component application architecture
CACE	computer aided control engineering
CAD	computer-aided design
CAE	computer-aided engineering
CAN	Controller Area Network
CFD	computational fluid dynamics
CG	center of gravity
CLI	command-line interface
CLS	control loading system
COM	component object model
CPT	Captain
CS-23	Certification Specifications 23
CS-25	Certification Specifications 25
DAE	differential-algebraic equation
DCM	direction cosine matrix
DLL	dynamic link library
DoF	degree of freedom
EASA	European Aviation Safety Agency
EBHA	electrical back-up hydraulic actuator
ECEF	Earth Centered Earth Fixed
EHA	electrohydrostatic actuator
EMA	electromechanical actuator
FAA	Federal Aviation Administration
FAR	Federal Aviation Regulations
FBW	fly-by-wire

FCC	flight control computer
FCS	flight control system
FDM	flight dynamics model
FEM	finite element method
FO	First Officer
FSD	Institute of Flight System Dynamics
GUI	graphical user interface
HMI	human-machine interface
IPOPT	Interior Point Optimizer
LUGRE	Lund-Grenoble
LVDT	Linear Variable Differential Transformer
MBS	multibody simulation
MDO	multi-disciplinary design optimization
MTOW	maximum takeoff weight
NED	North East Down
OOP	object-orientated programming
OPV	optionally piloted vehicle
PFCS	primary flight control system
POH	Pilot's Operating Handbook
RS-232	Recommended Standard 232
RVDT	Rotary Variable Differential Transformer
SBD	simulation-based design
SFCS	secondary flight control system
TUM	Technische Universität München
UAV	unmanned aerial vehicle
UI	user interface

Symbols

B	Temperature coefficient of remanence
$C_{h,e}$	Hinge moment coefficient of the elevator
C_{th}	Thermal capacity
D^0	Central difference quotient
E	Young's modulus
F_B	Buckling load
F_C	Dynamic friction force
F_N	Normal force
F_S	Static friction force
F_V	Viscous friction force
F_{Pilot}	Pilot force
F_{Stick}	Stick force
F_{bf}	Breakout force
G	Gear ratio
H_e	Hinge moment of the elevator surface
I_p	Second polar moment of area
I_{min}	Minimal second moment of area
I	Second moment of area
J_{Cost}	Cost function
J_{zz}	Moment of inertia with respect to the z-axis
J	Moment of inertia
K	Effective length factor
L	Phase-to-phase inductance
M_R	Moment about the point R
P_{Pusher}	Stick pusher electrical power
P_{loss}	Power loss
R_{th}	Thermal resistance
R	Phase-to-phase resistance
S_e	Elevator wing area behind the hinge line
T_{Coil}	Motor coil temperature

T_{env}	Environment temperature
W	Aircraft weight
α_e	Elevator actuator deflection after the gearbox (positive according to "right-rand rule")
α_t	Angle of attack at the tail
α_{T_0}	Linear resistance temperature coefficient
$\alpha_{e,max}$	Maximum elevator actuator deflection
$\alpha_{e,min}$	Minimum elevator actuator deflection
$\boldsymbol{q}$	Quaternion
δ_e	Elevator control deflection
η_D	Dynamic viscosity
η_{Mech}	Mechanical efficiency
η_V	Viscous friction coefficient
η_{max}	Maximum elevator surface deflection (most downward deflection)
η_{min}	Minimum elevator surface deflection (most upward deflection)
η	Elevator surface deflection (positive for downward deflections)
$\mathbf{g}(\mathbf{z})$	Inequality constraints
$\mathbf{h}(\mathbf{z})$	Equality constraints
$\mathrm{T}_{\mathrm{Clutch}}$	Torque transmitted by a clutch
$\mathrm{T}_{\mathrm{HardStop}}$	Torque due to hard stops
T	Torque
μ_C	Dynamic friction coefficient
μ_S	Static friction coefficient
ν	Safety factor against buckling
$\overline{c}_e$	Mean aerodynamic chord of the elevator
$\overline{q}_t$	Dynamic pressure at the tail
$\ddot{\varphi}$	Motor acceleration
$\dot{Q}$	Heat flow rate
$\dot{\varphi}$	Motor speed
ρ	Density
σ_B	Buckling strength
σ_y	Yield strength
τ_{th}	Thermal time constant
υ	Safety against buckling
c	Damping coefficient
g	Gravitational acceleration
h	Spacing for gradient calculation
i	Current
k_e	Motor voltage constant
k_f	Friction coefficient

k_t	Motor torque constant
k_{CLS}	Spring stiffness of the control loading system spring
k	Spring stiffness
l_{bw}	Bob-weight lever length
l	Length
m_{bw}	Bob-weight mass
m_{cm}	Compensating mass
m_{el}	Mass of elevator surface
n_z	Load factor
q_s	Scalar part of quaternion
u_{cmd}	Voltage command
v_s	Stribeck velocity

1 Introduction

State-of-the-art development of modern aircraft involves many different engineering disciplines. These days advanced materials like carbon composites are gaining ground in aircraft manufacturing. Sophisticated flight control algorithms operate and fly large passenger airliners. Computers survey the pilot's inputs and prevent dangerous flight attitudes. *Unmanned aerial vehicles* (UAVs) like the *General Atomics MQ-9 Reaper* military drone but also toys like quadcopters can fly automatically due to modern flight control algorithms [Gen16]. Thanks to technological progress, flying today is safer than ever. Obviously, aircraft systems as well as their development become more and more complex.

Naturally, today's development processes have changed as well to meet the new requirements. Many different physical domains have to be considered during the development and design phase. Computers support in mastering complex tasks and have become absolutely essential. As such, construction is undertaken virtually in *computer-aided design* (CAD) software programs by modeling *three-dimensional* (3D) parts. *Finite element method* (FEM) analyses help to determine stress conditions inside these virtual parts under certain loading conditions. The design of optimal aerodynamic shapes is accomplished by *computational fluid dynamics* (CFD) simulations. Interaction of bodies and behavior of mechanical systems are assessed via *multibody simulations* (MBSs) and further physical modeling methods even allow to simulate electrical, hydraulic or thermal components. All these tools and methods enable engineers to find optimal solutions and allow to gain information in early development phases (e.g. virtual prototypes).

An essential part of any aircraft is the *flight control system* (FCS). As a part of this system, the *primary flight control system* (PFCS), including cockpit controls (e.g. sidestick or yoke), control surfaces, and engines, allow an aircraft to be flown safely. For a long time, FCSs were purely mechanical. These systems are characterized by the mechanical connection of the pilot's controls to the control surfaces via rods, bellcranks, and sometimes cables and pulleys. Whereas in almost every new commercial aviation aircraft conventional mechanical FCSs were replaced by *fly-by-wire* (FBW) systems over time, most aircraft in general aviation (smaller airplanes that are subject to the *Certification Specifications 23* (CS-23) ([Eur15]) of the *European Aviation Safety Agency* (EASA)) still use mechanical FCSs. Nevertheless, these mechanical FCSs are often supplemented with modern *autopilot* (AP) systems. The opportunity to delegate the constant task of correctively manipulating the aircraft's controls, or even fly entire routes automatically, are valuable benefits of APs. Using them, enables pilots to manage the actual flight instead of moving the controls. Naturally, this results in an enormous reduction of the pilot's workload, which is beneficial especially during emergencies [Fed09]. Modern AP systems for general aviation aircraft are usually represented by a simplex architecture, containing one *flight control computer* (FCC), and one actuator per pitch, roll, and yaw axis. Flight control algorithms running on the FCC calculate necessary surface deflections, which are sent to the corresponding actuators. These are linked to the mechanical system via clutches. Thus, in case of an AP system failure, the actuators can be decoupled from the mechanical FCS. Therefore, no redundancy is needed and the number of components and costs for certification is kept to a minimum.

On the other hand, FBW systems do not use any mechanical connection between the pilot's controls and the control surfaces. Pilot commands are sent electronically to the FCCs directly, which surveil and assess them with respect to flight control laws, and then command the actuators. These flight control laws prevent pilot commands that may result in hazardous flight states. In case of using an AP, pilot-based commands are no longer transmitted to the actuators, but rather commands are calculated by the AP algorithms. As there is no mechanical backup system, for the case of a FBW system error most of the critical components must be installed multiple times to reach a total loss probability of the FBW system of less than 10^{-9} [Eur16]. This is one of the reasons why most general aviation aircraft still use mechanical FCSs:

- Usually, general aviation aircraft have only a very limited payload. As FBW systems require components existing multiple times for redundancy, the already small payload would be reduced by the entire system significantly.
- Costs for developing a pure FBW system can easily exceed the costs of a small aircraft [HSPH13].
- Maintenance becomes more complex and more expensive, since more complex components need to be serviced.
- Due to complex hard- and software, certification processes becomes a lot more difficult and with that more expensive [HSPH13]. However, conventional certification specifications are currently revised in order to certify subsystems with less effort and costs.

Regardless of whether a mechanical or FBW FCS is designed, the physical components always have to be taken into account to represent the dynamic behavior of the systems properly. Instead of using complex and abstract mathematical models, a MBS qualifies perfectly to consider mechanical components easily, as the composition of MBS models is similar to the systems they represent and not abstracted to mathematical models on the modeling level.

However, the development of these models is often accompanied by a tool-chain gap that produces a significant amount of manual work and with that, of course, costs, even in today's development processes.

1.1 Background

In recent years, a popular general-purpose simulation tool for control engineering and model-based design, not only for aerospace purposes, is *MathWorks Simulink*® ([The16e]) [Arn04]. Additional toolboxes like *Simscape*™ ([The16d]) or *Simscape Multibody*™ [1] ([The16c]) provide an environment to rapidly create models of physical systems or MBS models of 3D mechanical systems for multi-domain simulation within *Simulink* [The16b]. Many methods such as optimizations, trim and linearization, or sensitivity analyses from *MathWorks MATLAB*® ([The16a]) can be applied to any model within this *MathWorks* environment. Today, *Simulink* combines tools for *computer aided control engineering* (CACE), MBS, multi-domain simulation and many others. The software qualifies perfectly to model FCSs with its different subsystems and to develop corresponding flight control algorithms. One complex model is built, whereas kinematics themselves can be modeled using *Simscape Multibody*, and subsystems of, for example, electrical or hydraulic nature can be modeled using *Simscape*. By simulating undesirable side effects like friction or backlash in the mechanical part, a high fidelity model is created that enables for the assessment of interaction of components and performance of the entire complex system. In fact, friction is one of the main causes of control problems [Hac15]. For these reasons, the widespread *MathWorks* suite is used for functional design in this thesis.

However, despite these many advantages, the use of *Simscape Multibody* in a development process causes a big software gap regarding the MBS model building. Building models by hand with CAD models as masters is common practice. Although design tools like *CATIA*® (*Dassault Systèmes*) or analyses tools like *MATLAB / Simulink* are well developed, they lack of integration with each other. There are only a few interfaces between some CAD systems and *Simscape Multibody*. These interfaces are most often unidirectional and offer very limited functionality (market overview follows in Chapter 3.1). There is actually no bidirectional interface to the most popular CAD software *CATIA V5* ([Das]) which is controllable from *MATLAB*.

As part of this thesis, an integral method for a bidirectional data transfer is developed that allows for an interaction between selected CAD tools and *MATLAB / Simscape Multibody*. The method enables to automatically convert even complex CAD models that fulfill certain modeling rules to *Simscape Multibody* models within a few seconds. The bidirectionality provides for the transfer of geometrical modifications back to the CAD

[1]Before *MATLAB R2016a* known as *SimMechanics*

system. Thus, it is possible to apply *MATLAB* analysis methods and functions such as optimization algorithms on even CAD parameters.

A consequence of closing the software gap this way is a more continuous development process with enhanced methods and functionality. This process provides an environment that allows for a better and more competitive cutting edge design. Although this applies to development processes in many industries, this thesis focuses on the aerospace industry.

1.2 History

Computers and specialized software tools have played an increasingly important role in aircraft development in recent decades. The first commercial airliner designed through a CAD process in the early '90s is the *Boeing 777* [WL13].

In these times, there was no comprehensive software for developing mechatronic products like *Simulink* today. There was a software tool for every single engineering discipline such as CACE, CAD, MBS, FEM, or CFD amongst others. But over time, new mechatronic products required more sophisticated controllers. Before then, linear state-space models were exported from MBS to CACE tools, but an increasing controller complexity has now generated demand for more tightly coupled interfaces between these tools. Attempts to transfer full MBS models to CACE tools produced poor performance since the CACE solvers were not designed for *differential-algebraic equations* (DAEs), which typically are used to represent physical systems. Therefore, the time integration of the full system including controllers was undertaken in MBS tools [VKV04, Arn04]. Today *Simulink*, as a state-of-the-art software tool, uses modern solvers that solve differential and algebraic parts simultaneously, and it is even capable of handling multiple local solvers in one model [The16b].

Software tools from other disciplines like CAD and MBS have not been highly integrated with each other. Very often the MBS model is still built and modified (in iterations) by hand on the basis of CAD models or drawings [JNSM16]. Engineers key in numerical values such as inertia tensors or change CAD parameters manually up to the present day.

In the case of *Simscape Multibody*, *MathWorks* provides an interface named *Simscape Multibody Link*. Unfortunately *Simscape Multibody Link* with its limited functionality satisfies only a fraction of the needs of a continuous development process. Although it is not closing the software gap completely, it is a first step in the right direction by reducing manual work.

Nevertheless, the software gap between CAD and MBS tools and especially between the very popular tools *CATIA V5* and *Simscape Multibody* still exists. Even if the conversion of CAD to MBS models itself might be automated in some very few cases, there is still a software gap that causes a significant amount of manual work due to development processes that are dominated by many iterations and interdisciplinary design changes.

1.3 State-of-the-Art

During today's development processes of mechatronic systems, virtual prototypes in form of extended MBS models arise that allow for the estimation of system behavior and performance (depending on the modeling depth) and a validation of formalized requirements, for instance, without any part of the systems even existing.

Common MBS software tools used nowadays in the industry or research for the development of these models are *Adams* (*MSC Software Corporation*), *SIMPACK* (*Dassault Systemes Deutschland GmbH*), or as already introduced *Simscape Multibody* (*MathWorks*). Although all tools are MBS programs, their functional spectrum and consequently their area of application differs considerably. While *Adams* and *SIMPACK* are standalone programs dedicated to highly detailed and sophisticated MBS that can even represent flexible bodies, *Simscape Multibody* covers only classical MBS, but in form of a Simulink toolbox that integrates seamlessly into the *MathWorks* world. This means that all *MathWorks* functionality can be applied to the MBS models and is even available for developing entire virtual prototypes (e.g. including controllers, finite-state automatons), which allows for a more continuous tool-chain. Therefore, this thesis focuses on the *MathWorks* tool-chain and *Simscape Multibody* as MBS software tool.

Virtual prototypes of mechanical or mechatronic systems are developed in complex, multi-staged, and iterative design processes that require a broad range of engineering expertise. Discontinuities in the tool-chain cause manual work and prolong the design process. At the same time the process is prone to error by many manual actions.

Such a process is shown in Figure 1.1. Each step is represented by a box, where a red background indicates a step that requires manual action and a green background indicates an automated step. Obviously, this process is characterized by many manual steps and tool-chain gaps. It should be noted that the same gap between CAD and *Simscape Multibody* shows up multiple times. In the following each step of this process chain is discussed using the exemplary design of a FCS. The process assumes that no interface and primarily *CATIA V5* as CAD software is used, as leading aircraft manufacturers use this software.

Requirements Specification

First and foremost requirements regarding the system are gathered and specified [VDI93]. Quite often requirement management tools like *IBM Rational DOORS* or *Siemens POLARION* are used. In the case of FCSs for general aviation aircraft, there are mandatory requirements due to certification regulations as well as requirements that are defined by the aircraft manufacturer. The manufacturer itself might define a requirement regarding the very limited installation space of the control stick, for instance:

The control stick must have a total length of between 400 mm and 600 mm.

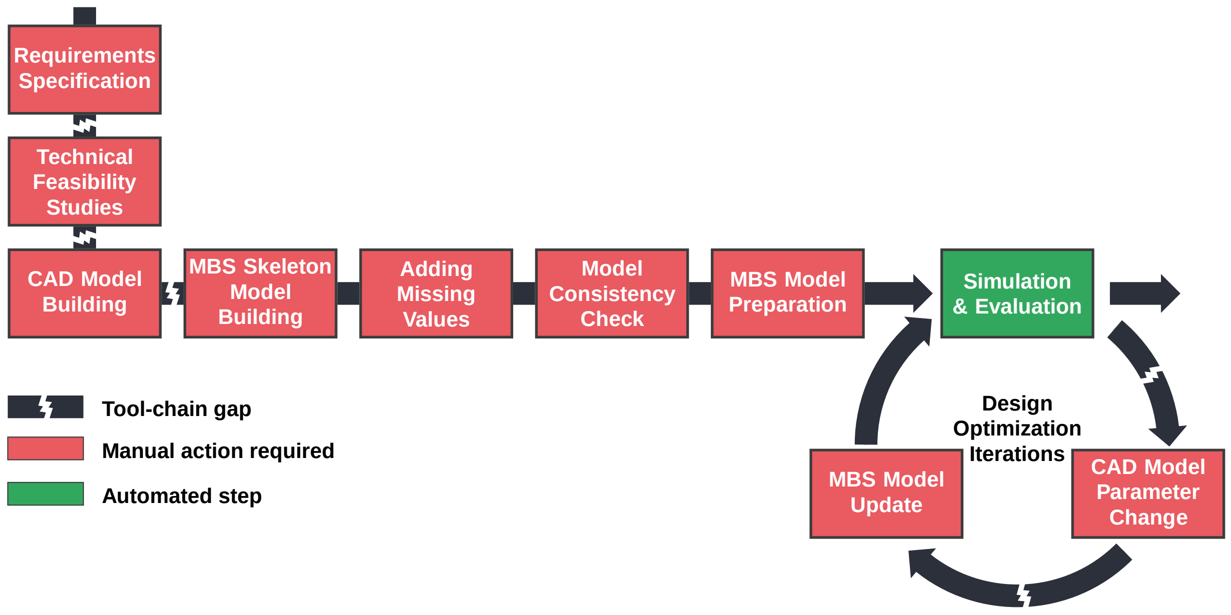


Figure 1.1: *Common development process of mechatronic systems.*

Since this is a design requirement, only the CAD system is necessary to design a system that meets it. On the contrary, there are mandatory requirements from CS-23 like the following:

> ***CS 23.155 Elevator control force in manoeuvres***
>
> *The elevator control force needed to achieve the positive limit maneuvering load factor may not be less than - [...] for stick controls, $W/14N$ (where W is the maximum weight in kg) [...] or $66 \cdot 8N$ (15 lbf), whichever is greater, except that it need not to be greater than 155 N (35 lbf).*

Designing a system that satisfies requirements like this, clearly shows the need of both software tools, the CAD system and the MBS software.

Technical Feasibility Studies

The technical feasibility studies aim to show that the system concept works by making basic analytic calculations and rough estimates on the system parameters [BRIA14, VDI93]. Static calculations allow for estimation of expected performance values or determination of dimensions or ratios of components like levers, so that a first probable technical solution can be identified [CE08]. This is particularly valuable for the following step.

CAD Model Building

After specifying all the system requirements and the technical feasibility studies, the design phase begins [Ver98] and a CAD model is usually built. Engineers use CAD tools to form single 3D parts, primarily with regards to design requirements (such as dimensions or masses). If a FCS is modeled, parts are created for each component like control sticks, rods, and bellcranks. At this stage, it is reasonable to use parameters for lengths, diameters, or positions that might change later as such a parametrized model can be changed easily afterwards. Then these parts are assembled to products. Therefore, kinematic constraints are created between parts, and these constraints position the parts either in relation to each other or in space. At the same time they define the relative motion of parts, since every kinematic constraint blocks one or more *degrees of freedom* (DoFs) between them. In the end, the combination of a part pair's unblocked DoFs specifies the type of joint that links them to each other. Since it is more intuitive to define joints rather than constraints between parts, most of the popular CAD tools provide such a functionality and create corresponding constraints automatically after a joint type is chosen. The resulting CAD model represents a 3D visualization, using solids that have a specific density and inertia. Moreover, if constraints are set not only for a proper positioning but with respect to motion, the CAD model contains valid kinematic information as well.

MBS Skeleton Model Building

CAD models are often used as masters to build a MBS model [Fis07, SPLK01, RW99]. The following steps are undertaken in a MBS software, such as *Simscape Multibody*. For every part in the CAD model, a *Solid* block is dragged manually from the *Simscape Multibody* library to the model. The same applies to joints. Finally, *Rigid Transform* blocks are added to the model in the same manner, which represent a transformation between two coordinate frames and are necessary for positioning and orientating solids as well as joints to each other.

Furthermore, this step involves structuring the model and connecting blocks that belong together. Naming components in the CAD and MBS tool equally facilitates a better orientation and helps to accomplish this step.

The result is a structured MBS model in which blocks are interconnected correctly but which still lacks a lot of information. Depending on the particular block, different data such as masses, inertias, distances, or the like is needed. This data is added in the next step of the process.

Adding Missing Values

In order to obtain reasonable results with MBS, each solid's mass, *center of gravity* (CG), and inertia tensor as well as positions and orientations have to be entered. The entire data is available as they are generated and computed by the CAD system. In the case of solids, their masses, CGs, and inertia tensors have to be manually transferred accurately. Accuracy is very important here since this data can influence the system behavior considerably. For instance, it affects a FCS's performance when accelerating the system from steady state or during flight maneuvers.

Entering positions and orientations is more challenging. *Simscape Multibody* requires a consecutive modeling style, meaning that solids and joints can not be positioned and orientated with respect to the global coordinate frame, but only to the one of the predecessor block. Consequently, relative positioning and orientating data is needed. However, usually CAD systems provide data with respect to the origin, but not to other parts. Either measurement tools have to be used to gain this information or it has to be calculated by hand. Especially when simulating overconstrained mechanisms or ones containing kinematically closed loops, this data has to be very accurate. Otherwise, solvers are not able to solve the equations of motion, describing the kinematic behavior of a mechanism.

Depending on the model's complexity, this and the previous step may take from a few minutes up to several days or even weeks.

Model Consistency Check

Having transferred, calculated, and entered all these data, the consistency / data integrity of the *Simscape Multibody* model has to be verified. Typing errors or other corrupted data could lead to unrealistic and therefore useless simulation results.

MBS Model Preparation

At this point, the CAD data is completely translated into a MBS model, but the kinematics are so far only under the influence of gravity. Therefore, preparing the MBS model afterwards is essential for the simulation. The implementation or rather integration of control algorithms takes place. External forces are applied on the mechanism. Values, important for later analyses (such as positions, velocities, accelerations, and forces), are logged in the form of time series and saved to variables.

In the case of designing a FCS, a *flight dynamics model* (FDM) or aerodynamic forces might be added. Furthermore enhanced physical models of components, such as electrodynamic models of actuators or clutches, can be added as part of the MBS model preparation. After preparatory measures, the model is ready for simulation.

Simulation and Design Optimization Iterations

This section covers the *Simulation & Evaluation* step as well as the following loop *Design Optimization Iterations*, shown in Figure 1.1.

First, the simulation is started, and the solver begins with the numerical solving of the DAEs. Once all time steps are calculated, results and measurements can be evaluated. Modern MBS tools provide a 3D visualization that allows for the visual assessment of the kinematic motion at this point.

Analyzing the measurements and the 3D motion visualization, gives some indication on the system behavior and whether it can meet the requirements specified in the first step. At this point, systems usually show an unexpected behavior, the performance may be insufficient, or measurements may not be within given limits. For example, the elevator control force in maneuvers of the FCS fails to reach the required value (*CS 23.155*).

In order to improve the system behavior, parts may require design changes or controllers may need a gain adjustment. To increase the elevator control force in the given example, the gear ratio between control stick and elevator surface may be inappropriate and needs to be lowered. This could be done by shortening the stick itself and thereby decreasing the lever arm.

However, when parts are changed, their specific data such as masses, CGs, and inertia tensors change. Furthermore, positions and orientations of subsequent parts and joints may change as well. To calculate the changed data in such cases, the CAD system is necessary again. If a parametrized CAD model was built, very probably the control stick's length is represented via a parameter and is changed quickly. But whether parametrized or not, the length of the stick has to be changed, and that is undertaken in the CAD software. The gear ratio can, of course, also be changed somewhere between the control stick and the control surfaces.

Afterwards, design changes and their consequences have to be transferred manually to the MBS tool again. Depending on the extent of the design change, under some circumstances this involves a recalculation of the values of relative positions and orientations.

Having this new data entered into the MBS model, the entire process starts again with a new run of the simulation of the changed system design.

In addition to this complex and discontinuous process, the tool chain in its current state has weaknesses regarding the optimization of FCS. As shown, a *multi-disciplinary design optimization* (MDO) can only be performed manually, as updates of the CAD, but also of the MBS model must be made manually. Beyond that, the *Simscape Multibody* blocks representing joints inherently provide no opportunity to easily consider complex non-linearities such as friction or backlash that might affect the system behavior significantly. The same applies to the consideration of buckling which is according to [Fed12] clearly unacceptable for push-pull rod systems.

1.4 Motivation

As seen in the previous chapter, the state-of-the-art process contains many manual steps requiring a lot of manual actions, making it unnecessarily time-consuming, prone to error, and expensive. Closing the tool-chain gap between CAD tools and especially *Simscape Multibody* in a way that a more integral development process is formed entails many advantages in technical as well as economical terms.

Figure 1.2 shows a part of the original state-of-the-art process and below, an equivalent MBS model building process using a bidirectional, enhanced interface between the CAD and MBS tool. This interface automatically transfers and computes relevant data and builds a MBS model. Furthermore, preparation of the MBS model can be partly automated by automatically adding models for friction, backlash, or buckling to the MBS model, and thereby considering non-ideal joints and solids.

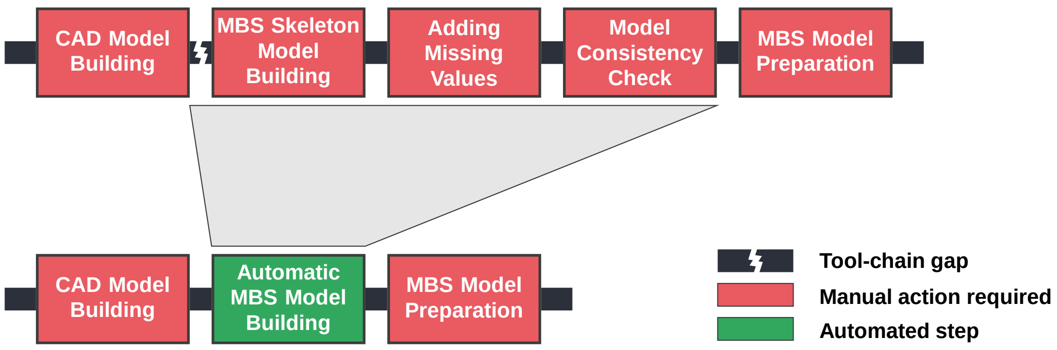


Figure 1.2: *MBS model building process using automated methods.*

This leads to several advantages, primarily a significant reduction of manual work and consequently much shorter durations of the model building process which is accomplished within only a fraction of the original time. In addition, the number of error sources decreases. Thus, there are no mistakes, such as typing errors, transposed digits, or calculation errors. Shorter model building times and fewer sources of error subsequently result in lower development costs and more reliable simulations.

A close of this tool-chain gap also applies the advantages mentioned above to the *Design Optimization* process loop shown in Figure 1.3. In this part of the process, bidirec-

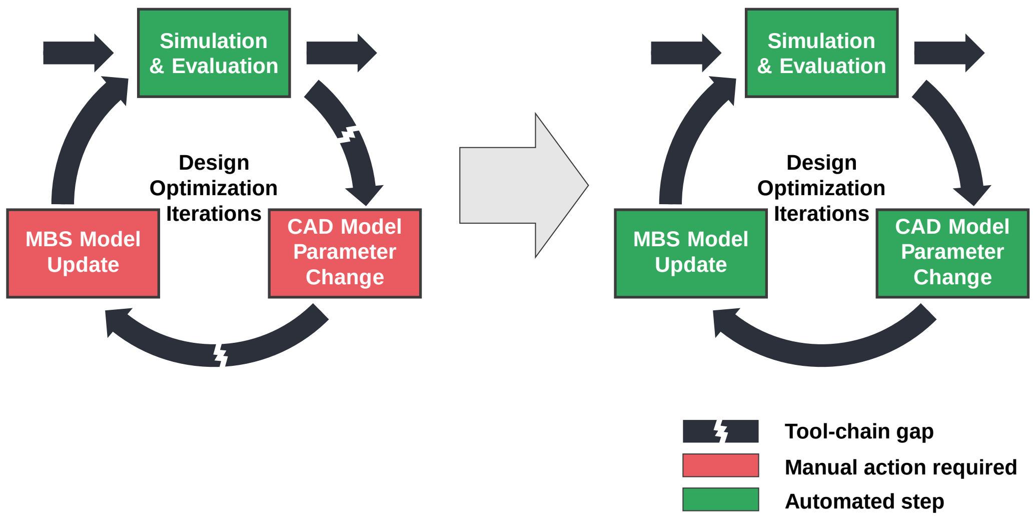


Figure 1.3: *Automated CAD design optimization loop.*

tionality is essential as the possibility of operations in both directions allows to automate the step *CAD Model Parameter Change* in Figure 1.3. By this means, the loop over both tools is closed. This enables the use of optimization algorithms that optimize even CAD parameters with respect to results of the MBS, for example.

In the example of the control stick from Chapter 1.3, an optimization algorithm could be used to find the optimal length after preparations and implementation of some additional data. The algorithm running in *MATLAB* changes the control stick length in the CAD tool. Afterwards, the MBS model is updated and the simulation is started. The optimization algorithm decides, depending on the measurements of the simulation, how the stick length is changed. This is done until the length is within the design requirements of the aircraft manufacturer and at the same time satisfies the elevator control stick force requirement *CS 23.155*.

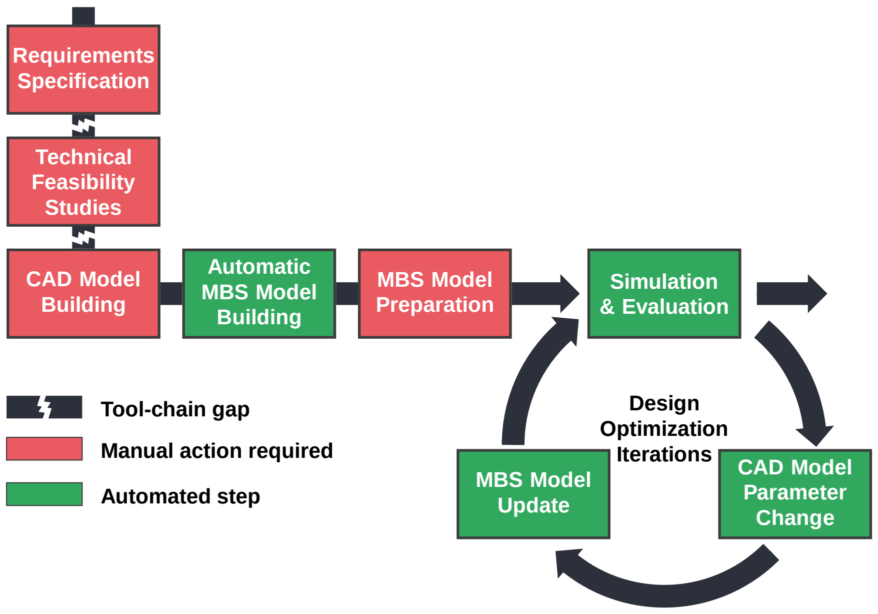

Requirements Specification
Technical Feasibility Studies
CAD Model Building
Automatic MBS Model Building
MBS Model Preparation
Simulation & Evaluation
Design Optimization Iterations
MBS Model Update
CAD Model Parameter Change
Tool-chain gap
Manual action required
Automated step

Figure 1.4: *Entire development process caused by using a bidirectional interface.*

The resulting comprehensive process is shown in Figure 1.4. As a result of shortening this process, knowledge of the system behavior is available at much earlier stages of the development, and furthermore products require fewer development periods in total. Additionally, since *Simscape Multibody* is a toolbox of the *MathWorks* world, it allows for the application of many other analysis methods on the entire system, such as sensitivity analysis or trim and linearization, which are common methods during development and construction. As an overall consequence, this provides for competitive advantage.

All the advantages mentioned have positive effects on development processes in any industry sector. However, especially the more complex the system is and the longer development periods are, the more the positive effects impact the development. Design cycles in the aviation sector are usually up to several months or years for components - whole airplanes sometimes take even decades until they are entirely developed. Using methods especially tailored to the development of FCSs, which allow for the simultaneous optimization of mechanical and software parameters to find optimal system designs, automatically monitor buckling in selected parts, consider friction and backlash in selected joints, verify simulations against requirements, or consider inertia loads due to flight can help to gain detailed information at early development stages and to shorten overall development times.

1.5 Literature Review

In present product development processes, simulation is an important tool and plays an increasing role [MKLC09]. As [MMGG12] states, using simulations as early as possible during the system design phase is a paradigm called *simulation-based design* (SBD). In such a process, simulations of any kind are used for design evaluation and verification in order to eliminate poor designs as soon as possible. For efficiency reasons, there arises the need for a tight coupling of design in CAD tools and the different analysis disciplines and their tools. The general SBD approach and its implementation and integration of computational tools is described in [Bos98]. Current research focuses on the coupling of CAD and analysis tools, which mostly provide FEM or CFD methods.

In [LAT15], a method for integrating CAD tools and software for FEM is presented. Their approach allows CAD models to be modified automatically on the basis of an FEM analysis. However, the method described in this thesis focuses on the coupling of CAD and MBS software to complete the corresponding design loop. The authors of [GM11] claim that, to date, research puts little emphasis on completing the design loop and the automation of the entire process. Although the need for the integration of CAD and behavioral modeling was identified almost two decades ago, this statement is still valid today.

An SBD environment with a new modeling paradigm to combine CAD and MBS models is introduced in [SPK00]. The core of the system is a central design database, which stores function, behavior, product structure, and CAD data for the current design. This database can then be modified, through several *graphical user interfaces* (GUIs), to access, for instance, CAD or behavioral data. However, this framework is not integrated into common and commercial CAD software tools and thus does not provide any means of modifying the geometry of individual components.

An interface with integration into the widely accepted CAD tool *SolidWorks* is presented in [EBPF03]. The software automatically converts CAD models into the *Modelica* language, which is a non-proprietary, object-oriented, equation-based language for convenient modeling of complex physical systems [Mod]. This set of *Modelica* components can then be simulated in a simulation environment like *Dymola*, for example. The visualization takes place in either the CAD software or in a dedicated visualizer. Nevertheless, there is no bidirectional data exchange that allows for modifying CAD components. A very similar approach targeting the CAD software *CATIA* but showing the same drawbacks is presented in [BSMC18].

In [CJ06], an integrated tool environment to determine a mechanism design with optimal kinematic and dynamic performance is presented. The proposed environment supports *SolidWorks* and *Pro/ENGINEER* [2]. The dynamic simulation *DADS* (Dynamic Analysis and Design System) is included, and *DOT* (Design Optimization Tool) is employed for a batch mode optimization. This approach closes a loop and allows a design to automatically be optimized on the basis of dynamic simulation results. However, apart from mechanical, other physical domains are not considered.

The approach presented in this thesis associates the design and the analysis process by integrating common CAD tools and *MATLAB / Simulink*. The method integrates seamlessly into *MATLAB* and provides access to its entire functionality, which includes physical modeling (multi-domain) and optimization algorithms. This implies that there is no need for external optimizers, although they are supported by the proposed method. The approach allows one to automatically derive *Simscape Multibody* MBS models from CAD models. The tight coupling to the CAD software tools is realized by using their *application programming interfaces* (APIs) and allows for a bidirectional data exchange. This system completes the design loop, even through to the controller design, and allows a high degree of automation.

Such a deeply integrated environment enables the designing of large heterogeneous systems involving all kinds of design variables, including geometric, physical, or logic ones. At least a subset influences overall behavior and performance of the system and decides whether system requirements are satisfied. The holistic determination of appropriate design variable sets is subject to the current research regarding MDO.

There are several systems engineering approaches that address multidisciplinary design. A very popular one is the V-model presented in [Has06], which reduces complexity in system development by disregarding design details in the top-level requirements. Subsequently, the details are considered stepwise by cascading requirements down. Another approach proposed in [PBFG07] structures systems on the functional level, which decomposes them and allows one to solve sub-problems. However, such concepts are too vague and do not help design on a quantitative basis.

In [ML13], several architectures for MDO are presented and classified. Yet, the focus does not lie specifically on the particular CAD and MBS software tools. In [MGM06], methods for the optimization of complex mechanical systems with application to vehicle engineering are treated. However, the approach of this thesis is very specific to the software tools that are used and which interfaces are available.

The author of [Bet10] presents optimization methods and discusses how they can be combined with discretization techniques for DAEs in order to solve optimal control problems. However, the methods are not suitable for MDO problems in the context of systems design.

Another multidisciplinary approach to develop complex systems in a top-down manner

[2]Now called *PTC Creo Parametric*

is presented in [PW, KMPJ03]. Top-level system design targets are cascaded down to lower levels by splitting the problem into to subproblems of subsystems and components. Nevertheless, complex coordinate strategies are required to ensure that subsystems are compatible with each other.

The methodology proposed in [ZKN+17] allows the determination of a solution space for a large system that is subject to uncertainty, for instance a system in early development stages. A solution space consists of permissible ranges for each design variable in which the overall system requirements are always satisfied. The primary goal is to maximize the solution space in order to provide flexibility for the subsequent implementation and make the system robust to design variable changes (while still satisfying the requirements). In [ZMRMZ17], extensions to this methodology are presented that enable one to consider varying design variables, additional constraints, and a balance between physical and logic design variables. However, the approaches described in [ZKN+17, ZMRMZ17] require a computationally expensive top-down target cascading using separate surrogate models to identify feasible solution spaces.

In contrast, the method of this thesis incorporates a sort of top-down method by optimizing the design variables and searching within the feasible space for the best solution. Yet, the simulation, which is used in the context of MDO, belongs to the bottom-up methods. Intermediate levels, like in the previously described approaches, are not required.

1.6 Purpose Statement

The goal of this thesis is to close one of the major tool-chain gaps within the development of FCSs by designing and implementing a method for a bidirectional data transfer between CAD systems and *Simscape Multibody* in order to reach a more continuous and integral development process. In fact, the developed method is not limited to this particular aerospace problem but can be applied in any mechatronic product development process. That way, physical components can easily be included in analyses (like trimming or linearizing for the design of controllers) or in further processes such as optimization or requirement validation.

Although the techniques of this thesis can be applied to many industries, this thesis relates intentionally to aviation. *Simscape Multibody* is chosen as the MBS tool, as it is part of the *MathWorks* world and its many toolboxes which allow the development of comprehensive virtual prototypes. Consequently, *MATLAB* is used to implement the method of this thesis to achieve a seamless integration into this environment.

The implementation of the method is modular and encapsulated so that any CAD software that provides the required information over a *component object model* (COM) server can be used after a CAD system specific *dynamic link library* (DLL) is created. These DLLs are loaded into the interface's *MATLAB* framework to enable communication to the corresponding CAD software.

This thesis concentrates on the CAD software *CATIA V5* (R20) as this tool is very common in especially the aviation industry. Furthermore, there is no bidirectional interface between *CATIA V5* and *Simscape Multibody*, as mentioned before.

Regarding the automatic MBS model building, aspects from graph theory are used to automate reasonable positioning of programmatically added blocks. Moreover, CAD data is fully parametrized within the *Simscape Multibody* model. This allows for quick CAD parameter changes and MBS model updates as long as topology has not changed. The development of a method providing for the optimization of mechanical and software parameters takes the demand for an integral MDO into account as well.

Furthermore, the advantages of automatic MBS model building can be used for adding automatically models for friction, backlash, or buckling together with their parametrization. Therefore the LuGre friction model and a model for simulating backlash are implemented to enable the method presented in this thesis to add them automatically to selected joints. Euler's model for buckling is also implemented to estimate the risk of buckling for selected parts.

In the application examples, the developed method is used to translate CAD models of FCSs of general and commercial aviation aircraft to *Simscape Multibody* models. CAD parameters of the mechanical FCS of the general aviation aircraft are optimized in order to minimize nonlinearities in the transfer function. Electrodynamic actuator models are introduced to determine a suitable configuration regarding supply voltage and gearbox ratio. The entire FCS is accelerated using data from a FDM to assess inertia loads in flight and to determine handling qualities. The model of the commercial aircraft FCS is used for a model-based requirement validation. An optimal system design in terms of mechanical and software parameters which meets specified formalized requirements is determined with the utilization of the developed optimization framework.

1.7 Contribution

The following key topics of this thesis and especially their combination represent the main contributions to the state-of-the-art. The next paragraphs give an overview and introduce each of them briefly.

Integral Method for a Bidirectional Data Transfer Between *Simscape Multibody* and 3D-CAD Software Tools

The method developed in this thesis provides for a bidirectional data transfer between *Simscape Multibody* and 3D-CAD software. Thereby, the encapsulated implementation is modular in order to easily connect further CAD systems via the COM interface. The connection to *CATIA V5* is used as a sample implementation. The method allows a fully automated conversion of CAD models into *Simscape Multibody* models, where no manual mapping or similar is required. The bidirectional approach presented in this thesis controls the CAD program, which means that also CAD parameters can be changed from *MATLAB*.

Simultaneous Optimization of Mechanical and Software Parameters

Mechanical parameters of the CAD model as well as software parameters are optimized simultaneously in order to find an optimal system design of a FBW control system of a commercial aviation aircraft. Thereby, requirements are introduced in form of inequality constraints to converge only to solutions, which are compliant with the requirements. For this purpose, a framework is developed, which utilizes the method for the bidirectional data transfer.

Incorporation of CAD Models in the Automated Model-Based Requirement Validation

The method of this thesis is applied to a CAD model of a commercial aviation aircraft FCS. The automatically generated *Simscape Multibody* model is used for the automated model-based requirement validation. Requirements are formalized and test cases verify the compliance of the model.

Integrated Consideration of Complex Non-Linear Effects

To consider non-ideal joints in *Simscape Multibody* models, mathematical descriptions for friction and backlash are introduced. The method presented in this thesis incorporates models which represent these effects and allows to automatically consider them in selected joints. The advanced LuGre friction model ([COAL95], [Ols96]) is implemented to take a wide range of important friction effects into account (hysteresis, frictional lag, pre-sliding displacement, or stick-slip).

Parallel Strength Estimation of Components Endangered by Buckling

For slim (hollow) cylinders, a the second moment of area is used to estimate the risk of buckling during the simulation by applying Euler's column formula. The developed method automatically adds blocks for the buckling analysis to user selected parts and provides an approach for approximating the second moment of area from the moment of inertia.

Evaluation of Handling Qualities with Comprehensive MBS Models

Inertia loads due to aircraft maneuvers are included in the MBS model of a mechanical FCS of a general aviation aircraft. The comprehensive model allows to evaluate handling qualities and thus flight safety. The stick force per g characteristic of the FCS is assessed and subsequently adjusted with a bob-weight to meet the corresponding requirement of the CS-23.

1.8 Outline

After this introductory chapter, the thesis begins with a discussion of typical components of FCSs in Chapter 2. Thereby, important physical effects that are related to these components are considered and mathematical models describing these effects are introduced.

In Chapter 3, a market study is conducted, which clearly shows the demand of an integral method providing a bidirectional data transfer between *CATIA V5* and *Simscape Multibody*. The specification as well as the implementation of essential features of the method developed in this thesis are presented. In addition, the structure of optimization framework for the simultaneous optimization of CAD and software parameters is described. Implemented strategies for a time-efficient optimization conclude the chapter.

The application of the presented method to a mechanical FCS of a general aviation aircraft is subject of Chapter 4. After a conversion of the CAD model to a *Simscape Multibody* model, CAD parameters are optimized in order to minimize nonlinearities in the transfer function. Afterwards, an advanced actuator model is attached and friction is considered in joints to determine actuator loads due to aerodynamic forces. By introducing backlash and a FDM, the inertia loads due to maneuvers are considered and handling qualities of the aircraft are assessed and modified. Finally, the risk of buckling for the push-pull rods is estimated.

Chapter 5 focuses on the model-based requirement validation for a FCS of a FBW commercial aviation aircraft. The automatically generated *Simscape Multibody* model is verified against previously formalized requirements. Afterwards, an optimal system design is determined. Mechanical and software parameters of the model are simultaneously optimized with respect to requirements.

A summary of the thesis and an outlook for potential future work is given in Chapter 6 which concludes this thesis.

2

Physical Components of Flight Control Systems

This chapter introduces common components of FCSs. As such, it covers their typical field of application and physical relations that are important especially for the design of FCSs. Most of these physical relations are developed and derived within this chapter but initially used later in the application examples. Subject matter are components affecting the FCS directly, such as rods, bellcranks, joints, cables, actuators (including the controller), clutches, or sensors. However, avionic components for navigation, communication, display, flight guidance or FCCs are not considered.

2.1 Rods and Bellcranks

Mechanical FCSs can be divided into two types: Push-pull rod systems and cable systems. This chapter deals with rod systems, while cable systems are covered in Section 2.3. In a rod system, push-pull rods connect the cockpit controls with the flight control surfaces, and bellcranks link these rods to each other.

Push-pull rods are usually mounted via hinge bearings on both sides, so force transmission can only occur along the longitudinal axis in the form of a normal force. Bellcranks can be used to introduce a lever to increase or decrease forces. In addition to transmitting forces from one rod to another, they also permit forces to be rerouted along another direction. Thus, it is possible to install rod systems following the shape of the airplane's fuselage or to bypass obstacles. If a force is introduced into a mechanical FCS via the control stick, the flux of the force runs over the rod system and transmits it to the levers that are linked to the control surfaces.

2.1.1 Nonlinearities in Rod Systems

The use of bellcranks makes the system's behavior nonlinear, meaning that the ratios of output to input forces (each in axial direction of the rods) and displacements are not

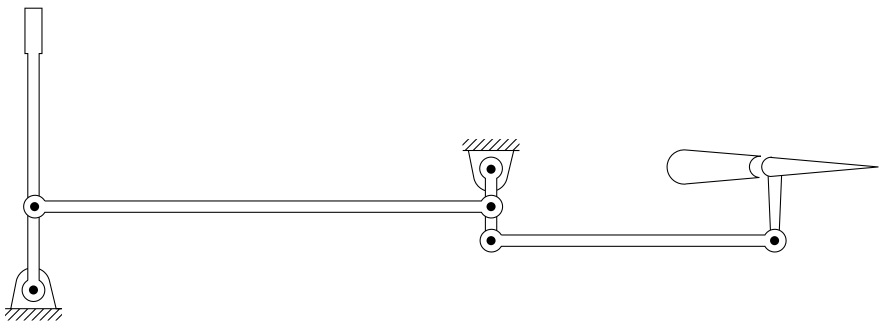


Figure 2.1: *Mechanical FCS using a push-pull rod system.*

constant. As rods are tilting when the system is actuated, effective lever arms change.

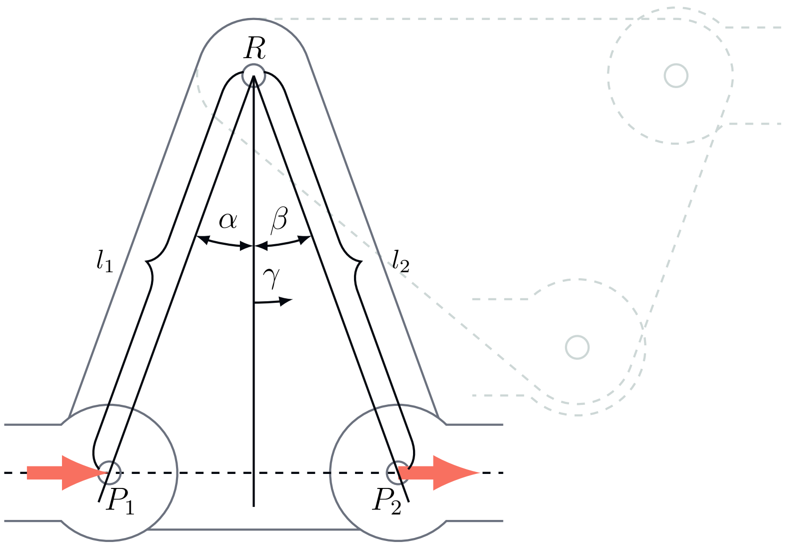


Figure 2.2: *Nonlinear transmission of forces and displacements.*

Taking the bellcrank shown in Figure 2.2, a force F_1 introduced on P_1 in an axial direction results in a moment M_R about point R:

$$M_R = l_1 \cos(\gamma - \alpha) F_1 \tag{2.1}$$

where γ is the rotation angle of the bellcrank. The force F_2 in the axial direction of the subsequent rod, which is caused by a moment M_R and thus by F_1, is calculated as

$$F_2 = \cos(\gamma + \beta)\frac{M_R}{l_2} \tag{2.2}$$

With Equations 2.1 and 2.2, the inherent constant angles α and β, and the constant lengths $l_1 \neq 0$ and $l_2 \neq 0$, the force F_2 can be described as a function of F_1 and γ:

$$F_2(F_1, \gamma) = \cos(\gamma - \alpha)\cos(\gamma + \beta)\frac{l_1}{l_2}F_1 \tag{2.3}$$

This function becomes zero for $\gamma = \frac{\pi}{2} + \alpha$, $\gamma = \frac{\pi}{2} - \beta$, and $F_1 = 0$. In the γ-related zeros, a mechanism is in a so-called kneeling state (see grey colored illustration in Figure 2.2). For FCSs, this can be a catastrophic failure. Therefore, it must be a design maxim for an FCS to never reach these states.

For a symmetrical bellcrank with $\alpha = \beta$ and $l_1 = l_2$, Figure 2.3 shows qualitatively the resulting outputs normalized to the constant input over the bellcrank rotation angle γ. It also shows that α and β not only define the kneeling states and thus the maximum range of the bellcrank's rotation, but together with l_1 and l_2, they also influence the gear ratio.

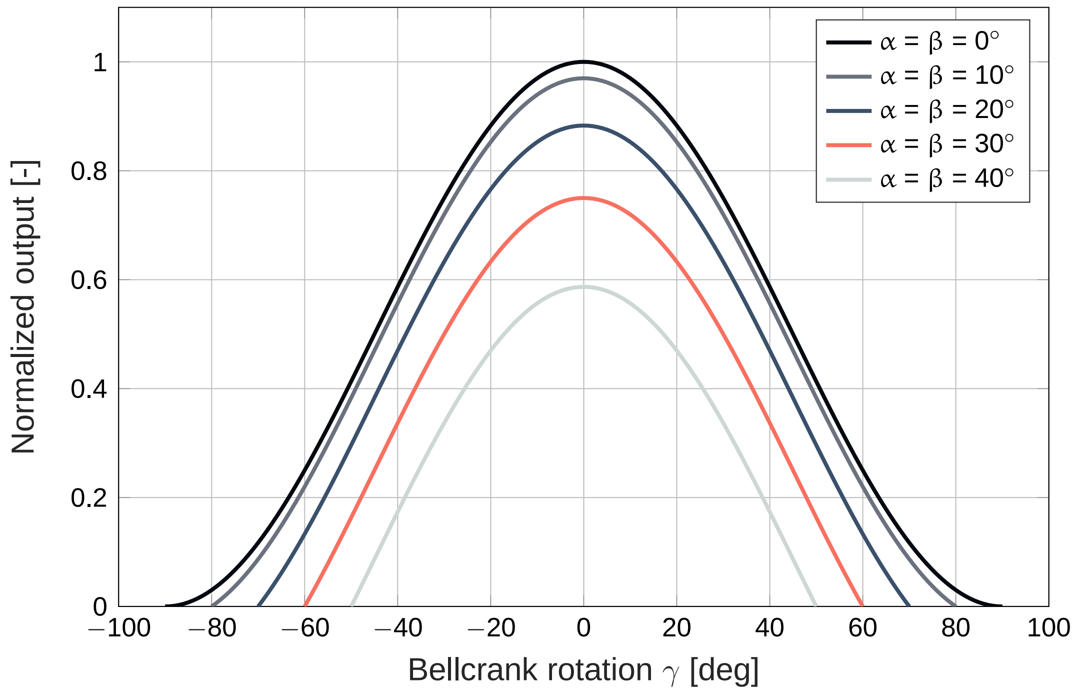


Figure 2.3: *Normalized outputs over the bellcrank rotation angle.*

Having multiple consecutive bellcranks means these nonlinearities can accumulate. As a result, there are system states where the ratio of inputs to outputs changes significantly with only small displacements. Since on one hand flight control algorithms usually command surface deflections and on the other hand there are nonlinearities in the mechanism, which links actuators and control surfaces, an analysis of such a behavior is important for the development of controllers.

Usually, FCSs are designed in a way that nonlinearities are small with the control stick in a central position. The more the control stick is displaced from this central position, the greater the nonlinearities become. Using MBS allows the degree of the system's nonlinearity to be easily determined.

2.1.2 Buckling Resistance

As mentioned, besides introducing nonlinearity, installing bellcranks involves several advantages. In addition to this, an occasional installation of bellcranks even reduces the risk of buckling. Rods of a FCS must not buckle under any circumstances [Fed12]. Risk of buckling exists especially in the case of slim parts that are exposed to high compressive forces. If they reach a certain level, the part loses stability, which causes an unpredictable deformation in lateral direction and involves a significant loss of load-carrying capacity. As rods are very slim parts, it is likely that there is a risk of buckling in mechanical FCSs. Buckled rods are not able to transmit forces properly and endanger a safe flight, hence it is important to consider buckling during the design phase. As classical MBS methods as well as *Simscape Multibody* always treat parts as rigid solids, buckling does not occur in the simulation inherently. Therefore, a separate buckling model, such as EULER's buckling model, is implemented in this thesis.

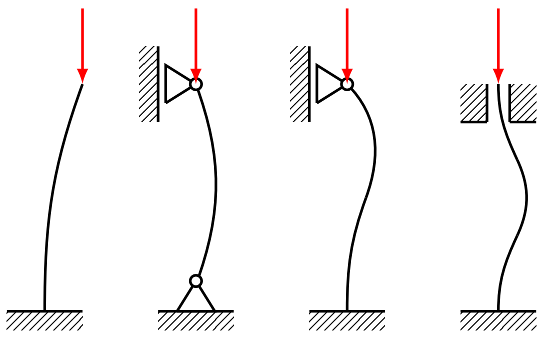


Figure 2.4: *The different* EULER *buckling modes according to [RS5].*

Analyzing Buckling with Euler's Column Formula

Buckling is a complex phenomenon, but there are methods to estimate limit loads. To analyze the buckling resistance, the EULER's equation for determining the buckling force can be applied:

$$F_B = \frac{\pi^2 E I_{min}}{(KL)^2} \tag{2.4}$$

where E is Young's modulus, I_{min} is the minimal second moment of area, and L represents column's unsupported length. The effective length factor K depends on the bearing of the column. With both ends of the rods hinged, $K = 1$ (EULER's second case). Further, safety against buckling is defined by υ, where $F_{N,max}$ is the actual maximum allowed compressive force:

$$F_B = \upsilon \cdot F_{N,max} \tag{2.5}$$

Related to structures, a safety factor of 1.5 is specified in [Eur15] and [Eur16], which also applies to safety against buckling. According to [BB15], formula 2.4 is valid, as long as Hooke's law applies, and consequentially as long as the buckling strength $\sigma_B \leq \sigma_y$ (yield strength). The buckling strength σ_B is defined as:

$$\sigma_B = \frac{F_B}{A} \tag{2.6}$$

The cross sectional area of the column, or in the case of a FCS, the cross sectional area of a rod is represented by A. Using Equations 2.4 to 2.6 allows one to make a rough estimate on buckling risk already during simulation. In summary, within this thesis, it is assumed that, upon fulfillment of Equation 2.7, no buckling occurs.

$$F \leq \frac{F_B}{\nu} \tag{2.7}$$

2.2 Joints

Mechanical joints link rods to each other, mount the FCS to the fuselage, and depending on the joints themselves, block certain DoFs. Different types of joints are used, such as prismatic, revolute, spherical or cylindrical ones. However, the most popular joint should be the hinge (revolute joint). It blocks five out of six DoFs and only allows a rotation around one axis. By nature, real joints are not ideal, as they are modeled within MBS or CAD software. There is a certain degree of backlash and friction that increases nonlinearity of the overall system. However, due to advanced methods of manufacturing, backlash can be reduced to a minimum. Further, certified lubricants reduce friction and wear, such as LPS 2®[LPS13].

If effects due to friction or backlash must be considered within a simulation, corresponding models have to be implemented. Since friction and backlash are important aspects of control systems, it makes sense to consider them especially when designing a FCS, as they might affect flight control algorithms and with that overall system performances negatively. Especially friction can lead to tracking errors or undesired stick-slip motion [COAL95]. Using a combination of an enhanced MBS model covering effects of friction and backlash together with a control algorithm, allows to assess the consequences of these inefficiencies on the system's behavior and to find better controller gains, for instance.

2.2.1 Friction

Tribology[1] is an important topic in current research and has still not been fully explored. Researchers are trying to develop precise tribological simulation models that cover the vast number of effects. The next sections present the most important types of friction, followed by an introduction of a suitable model and its simplifications. Here, a model is suitable in the sense of considering important and significant effects while simulating with

[1]Science of interacting surfaces in relative motion.

manageable computational costs. Thus, the aim is to have a balance between accuracy and computation time.

Friction is usually differentiated between external and internal friction. The latter is defined by dynamic viscosity, which plays a minor role only in fluids, such as lubricants in joints. The more important effects come from external friction, primarily static and dynamic friction, which basically arise when surfaces move relative to each other.

Static Friction

Technically, static friction plays an important role and occurs whenever two surfaces touch and do not move relative to each other at the same time. Static friction F_S is defined as

$$F_S = \mu_S F_N \tag{2.8}$$

Contrary to intuition, static friction is independent of the size of the contact area and only depends on the normal force F_N and the material-specific coefficient of static friction μ_S. As long as an external force $F_{ext} \leq F_S$, there is no relative movement.

Dynamic Friction (Coulomb Friction)

As soon as there is a relative motion ($v > 0$), dynamic friction occurs. It always acts against the direction of movement and is independent of the magnitude of velocity:

$$F_C = \mu_C F_N \tag{2.9}$$

Usually, the friction coefficient $\mu_C \leq \mu_S$ and consequently the dynamic friction $F_C \leq F_S$ with the same F_N. Due to the law of energy conservation, dynamic as well as viscous friction cause no loss of energy but rather a dissipation to thermal energy.

Viscous Friction

Viscous friction occurs in moving fluids, such as lubricated joints. In this case, surface areas and velocity have a major influence on the magnitude of viscous friction F_V, which is defined as

$$F_V = \eta_D A \frac{\Delta v}{\Delta y} \tag{2.10}$$

where η_D is the fluid's dynamic viscosity, A represents the surface area and $\frac{\Delta v}{\Delta y}$ is the velocity gradient. As most oils and lubricants are Newtonian fluids and clearance between adjacent parts of joints is small, causing only thin fluid layers, the velocity gradient can be assumed to be linear and constant [Sch10]. With this assumption made and y describing clearance and v relative velocity, the formula for F_V can be written as

$$F_V = \frac{\eta_D A}{y} |v| \tag{2.11}$$

Dynamic Friction Model LuGre

All three presented types of friction should be covered by a friction model. Static models that combine all three types create discontinuities and require case differentiation, both at the expenses of performance. Dynamic models mostly do not introduce these problems. Moreover, they are able to describe macroscopic effects, such as pre-sliding displacement or friction depending on increasing or decreasing velocity [Kub08].

Such a dynamic model is the *Lund-Grenoble* (LUGRE) model, which is described in [COAL95] and [Ols96]. It simulates many of the previously mentioned macroscopic effects and in addition does not require a case differentiation for $v = 0$. The basic assumption of the LUGRE model is that two surfaces contact through elastic bristles, visualized in Figure 2.5. According to [Ols96], the LUGRE model is based on the average

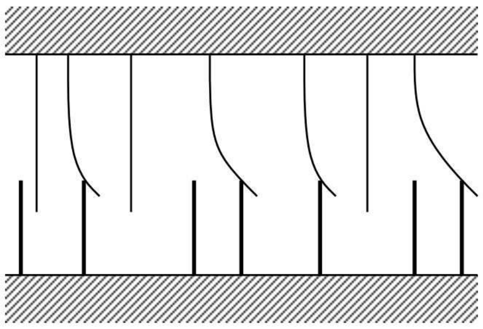


Figure 2.5: LuGre *model assumes friction as a deflection of elastic bristles. Lower bristles are shown as being rigid, for simplicity [Ols96].*

behaviour of the bristles. Therefore only one bristle, which behaves similar to a mechanical spring, describes the aggregated state of all bristles [Kub08]. Figure 2.6 shows an average

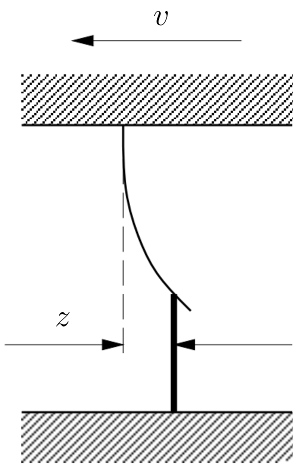


Figure 2.6: *The average deflection of bristles in figure 2.5 is represented by a single bristle [Ols96].*

deflection z of the bristles. This single bristle can never slip, as it aggregates all bristles. For motions with constant velocities, deflection reaches a steady-state value. Deflection z

is modeled by

$$\frac{\mathrm{d}z}{\mathrm{d}t} = \dot{z} = v - \frac{|v|}{g(v)}z \tag{2.12}$$

with v being the relative velocity between the two surfaces. The function g depends on many factors, such as material properties and lubrication. A parametrization of g that describes the STRIBECK effect, is

$$g(v) = \frac{1}{\sigma_0}\left(F_C + (F_S - F_C)\mathrm{e}^{-(v/v_s)^2}\right) \tag{2.13}$$

where v_s is the STRIBECK velocity, F_C can be derived with Equation 2.9, and F_S can be derived from Equation 2.8. Values for the STRIBECK velocity are chosen by [Ols96] in the range $v_s \approx [10^{-3}, 10^{-1}]$. In order not to underestimate friction force for small velocities, v_s should not exceed the lower bound of this range. The total friction force, generated from bristle bending and viscous friction, is described as

$$F_{fric} = \sigma_0 z + \sigma_1(v)\dot{z} + F_v(v) \tag{2.14}$$

which is equivalent to the equations of motion for a mass spring damper system, where σ_0 is stiffness, $\sigma_1(v)$ a velocity dependent damping coefficient and $F_v(v)$ a term proportional to velocity, representing the the friction force caused by the viscosity of the lubricant (see equation 2.11). In accordance with [Ols96], values for stiffness should be in the interval $\sigma_0 \approx [10^3, 10^5]$. The velocity dependent damping coefficient is given by

$$\sigma_1(v) = \sigma_1 \mathrm{e}^{-(v/v_d)^2} \tag{2.15}$$

where v_d defines the velocity interval around zero, where damping becomes active. Also, this value should not be chosen too small, as the motion may pass through zero too often without slowing down enough to a complete stop. Values for the damping velocity interval similar to $v_d \sim v_s$ are suggested. The authors of [Ols96] propose a relative damping of $\zeta = 1$. The damping coefficient is then given by

$$\sigma_1 = 2\sqrt{\sigma_0}. \tag{2.16}$$

2.2.2 Backlash

Backlash, also referred to as free play, is mostly an undesired effect, which allows a small movement of a part without transmitting a force to the following part. It is caused by gaps within joints and exists by nature in every joint connection. The only exception is a weld joint, where there is no relative motion at all. Accordingly, backlash only occurs in blocked DoFs, where immediate power transmission is desired. Backlash must be reduced to a minimum in FCSs, since tiny pilot or AP commands might be compensated by the gaps and do not lead to a surface deflection change. Free play also leads to discontinuities that can be problematic for flight control algorithms.

Backlash in Revolute Joints

In an FCS, backlash can originate in the revolute joints of the bellcranks, for example. Presuming there is a clear gap between pin and bushing, the pin could move uncontrolled in this space without transmitting any force or motion if the friction is disregarded (see Figure 2.7). This applies to the translational movement of the pin and, of course, to the rotary movement, which is not taken into account here.

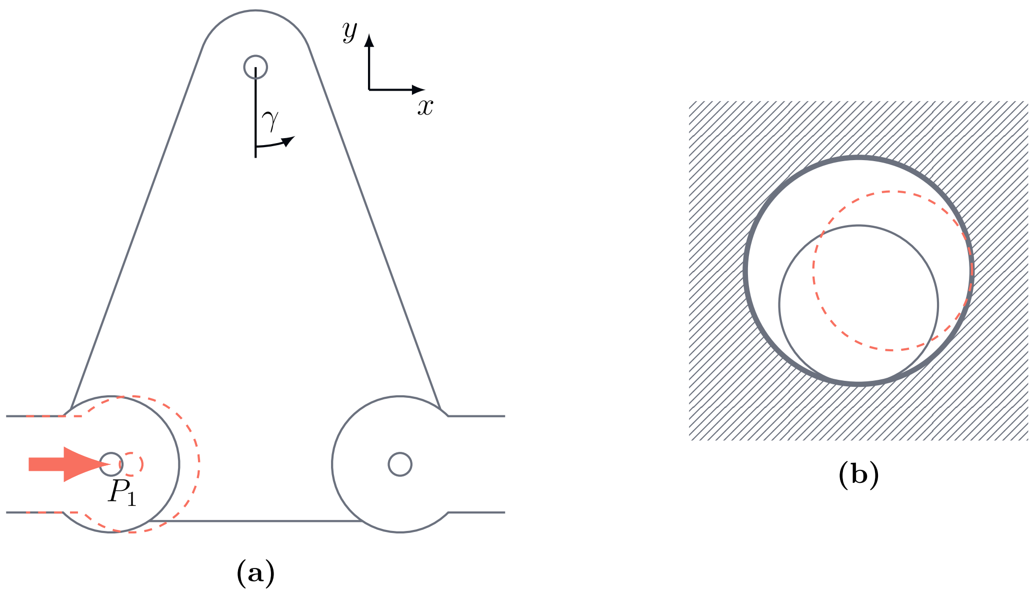


Figure 2.7: *(a) Backlash in a bellcrank. (b) Detailed view of the range of the pin movement in the bushing.*

Figure 2.8 shows, qualitatively, the resulting γ over a translational movement Δx of P_1 (normalized to Δx_{max}) for the case of a certain backlash in the revolute joint in P_1 compared to the case without any backlash.

Backlash Model for Revolute Joints

To model a revolute joint with a certain backlash, a planar joint consisting of one rotational DoF and, perpendicular to it, two translational DoFs is used. The translational DoFs allow the rotary axis to freely move within a certain area, representing the free play.

To constraint the movement to a specified area, a spring damper combination is implemented for each axis, which simulates a collision with the bushing by applying a force to the pin as soon as the rotary axis exceeds this area. If both translational DoFs are coupled, any shape can be defined for this free play area. In the following, a circular shape is intended, as this fits best with a round pin. The schematics for this case are illustrated in Figure 2.10.

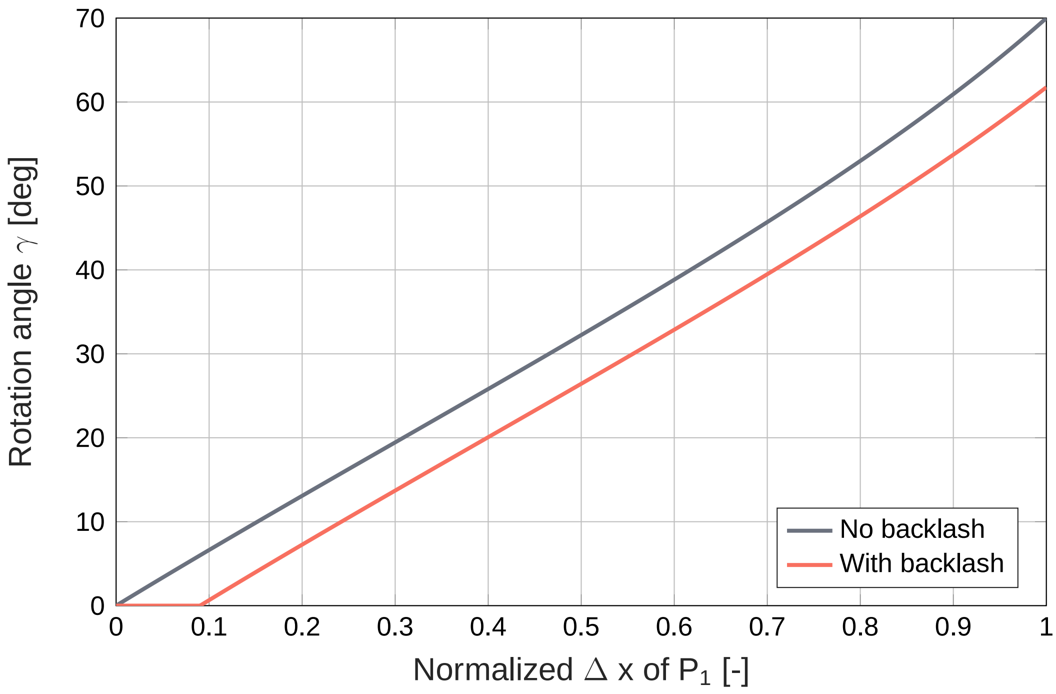


Figure 2.8: *Normalized outputs over the bellcrank rotation angle.*

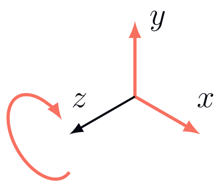


Figure 2.9: *Planar joint with one rotary and two translational DoFs.*

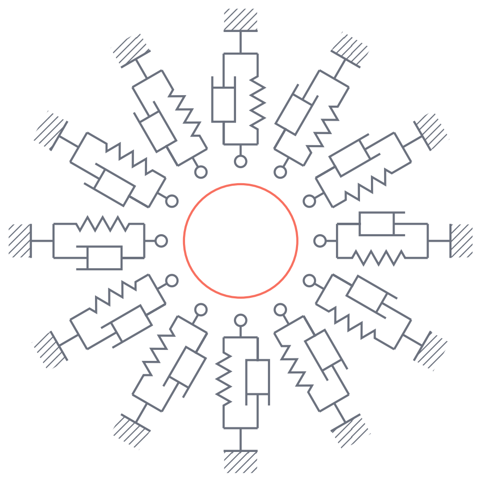
For the mathematical model in Equation 2.18, the pin is reduced to an axis with the position $\vec{p}$. Representing the backlash clearance, this axis is allowed to move freely within a certain radius r from the ideal axis, when there is no backlash. Only outside this area, the spring damper combination ensures that the maximum clearance of the joint connection is maintained. With

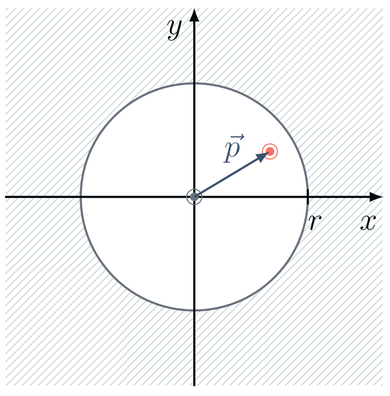
$$\vec{p} = \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.17}$$

where x and y are the coordinates of the rotary DoF notated in the joint coordinate frame, the force to maintain the joint connection is calculated as

$$\vec{F}_{backlash} = \begin{pmatrix} F_{backlash,x} \\ F_{backlash,y} \end{pmatrix} = \begin{cases} k\vec{p}\left(\frac{r}{\|\vec{p}\|} - 1\right) - c\dot{\vec{p}} & \text{for } |\vec{p}| > r \\ 0 & \text{for } |\vec{p}| \leq r. \end{cases} \tag{2.18}$$

To simulate a collision between solid bodies, the spring stiffness k must be set to appropriate high values. The damping term takes into account the inelastic component of the


(a)



(b)

Figure 2.10: *(a) Schematics of the revolute joint backlash model. (b) Visualization of the mathematical model.*

collision with the damping coefficient c.

The described backlash model can be combinated with the friction model presented in 2.2.1.

2.3 Cables

A push-pull rod system, as presented in Chapter 2.1, can partially or entirely be substituted by a control cable system. As cables can transmit only tensile forces, two cables are always needed for force transmission in two directions (push and pull). Another option is the usage of Flexball cables that allow compressive as well as tensile forces. While a cable system is usually lighter than a rod system, there is still more friction and more backlash within the system. Furthermore, conventional control cables stretch over time and also due to temperature changes, which requires a frequent adjustment of the cable tension. Nevertheless, cable systems are often installed, especially for controlling rudder and trim tabs.

2.3.1 Tension Regulators

In some FCSs, such as the Boeing 767, there is a significant difference in temperature expansion of the aircraft's aluminium fuselage and the steel control cables [Way18]. In these cases, tension regulators are used to maintain a constant preload in the cables automatically. Typically, FCS are designed to operate with a certain cable tension. Without tension regulators, temperature changes could cause a variation in the cable tension.

Tension regulators have separate sector halves for each cable end, which can respond to changes in the cable tension by a spring mechanism. However, both sector halves are connected via a self-locking mechanism, which allows a tension regulation only if both cable tensions are equal. Otherwise, the control system is active and does not allow any tension regulation.

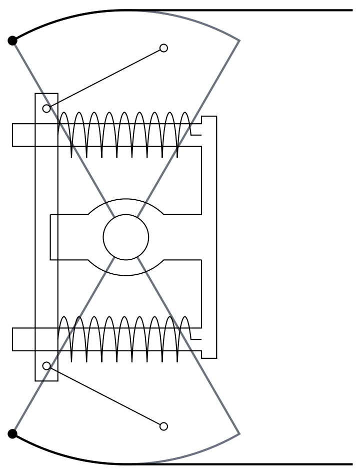


Figure 2.11: *Schematics of a cable tension regulator.*

2.4 Actuators

With airplanes becoming larger, the required control forces naturally increase. Pilots are not able to fly airplanes with a comfortable amount of effort when the aerodynamic forces become too large. Therefore, actuators are used to position the control surfaces. For this purpose, they produce a translational or rotational movement from a certain energy source, mostly of electrical or hydraulic nature. The first aircraft that used actuators was the *Boeing Model 307 Stratoliner* with it's first flight in 1938, already [RC93].

Besides the control surfaces, actuators are used for gear retraction, steering, breaking, and moving other components of the PFCS and *secondary flight control system* (SFCS) (e.g. flaps, slats) depending, of course, on the type of aircraft. However, to ensure a safe flight, a persistent authority over the control surfaces by the pilot is essential [RC93].

The control surfaces of commercial airplanes have been powered hydraulically for a while. However, today there is a trend toward more electric airplanes, and respectively actuators. Instead of hydraulic FBW actuators, which are connected to a centralized hydraulic system, *electrohydrostatic actuators* (EHAs) are used, which are basically self-contained hydraulic actuators incorporating a hydraulic pump that is driven by an electric motor. These new types of actuators are fundamentally electrically powered, but gearing and transmission are hydraulically achieved. A derivative is an *electrical back-up hydraulic actuator* (EBHA), which is a mixture between a conventional FBW hydraulic actuator and a EHA. EBHAs can be driven both in nominal mode hydraulically or electrically in case of a hydraulic system failure. Modern commercial airplanes use systems of this kind. In 2005, an *Airbus A380* was the first commercial airplane that flew with no hydraulics [Bos06, Tro07].

The *Dreamliner*, *Boeing*'s *787-8*, is the first large-capacity aircraft that uses *electromechanical actuators* (EMAs) within the PFCS, in particular to control two spoiler pairs on the wing surfaces [Boe14]. Today, spoilers can be counted among the PFCS, as they are also used for roll maneuvers in FBW airplanes. In EMAs all hydraulic technology is eliminated since they consist merely of a DC motor and a gear. Actuators containing hydraulics are, indeed,x by far stronger than EMAs. However, with a lower complexity, a lower weight as well as easier maintenance, they are very attractive for especially low power applications [Bos06]. Furthermore, they qualify perfectly for smaller aircraft with very limited payloads, such as in general aviation. This thesis only considers EMAs amongst actuators, as they are state-of-the-art.

2.4.1 EMA Model with Thermal Dependencies

The rotary actuator model and the thermal model presented in the following originate from [LHH]. The EMA model is based on the assumption that a brushless DC motor can be represented by a permanent magnet brushed DC motor. It disregards electrical commutation effects for simplification. However, thermal dependencies are considered, as they may have major impact on the system's performance.

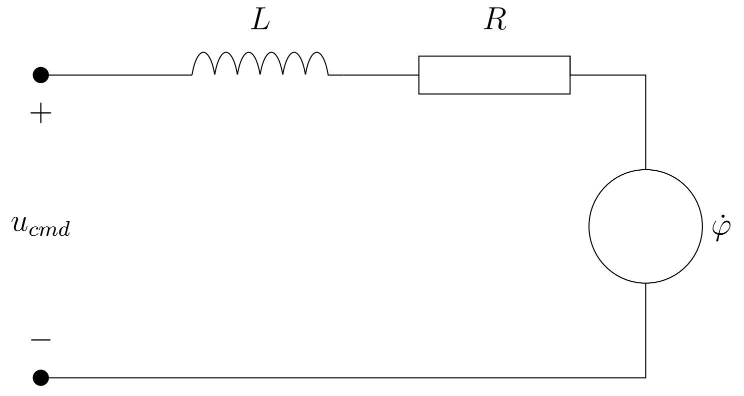


Figure 2.12: *Equivalent circuit diagram for the EMA model.*

The equivalent circuit diagram for this model is shown in Figure 2.12. The differential equation describing the motor dynamics is given by

$$L\frac{\mathrm{d}i(t)}{\mathrm{d}t} = u_{cmd}(t) - i(t)R(T_{coil}) - k_e(T_{coil})\dot{\varphi}(t) \tag{2.19}$$

with phase-to-phase inductance L, motor current $i(t)$, a controller commanded voltage $u_{cmd}(t)$, coil temperature T_{coil}, motor speed $\dot{\varphi}(t)$, and voltage constant $k_e(T_{coil})$. The phase-to-phase resistance $R(T_{coil})$ is defined as

$$R(T_{coil}) = R(T_0)\left(1 + \alpha_{T_0}(T_{coil} - T_0)\right) \tag{2.20}$$

where $\alpha_{T_0} = 0.00393K^{-1}$ is the annealed copper linear resistance temperature coefficient and T_0 is the specified temperature of the material coefficients. The torque output T_{motor} is modeled by

$$\mathrm{T}_{motor}\left(T_{coil}, t\right) = k_t(T_{coil}) i(t) - k_f \dot{\varphi}(t) \tag{2.21}$$

where $k_t(T_{coil})$ is the motor torque constant, and k_f is the friction coefficient. Regarding DC motors, the motor voltage constant equals the motor torque constant and is given by

$$k_e(T_{coil}) = k_t(T_{coil}) = k_t(T_0)\left(1 - B\left(T_{coil} - T_0\right)\right) \tag{2.22}$$

with the temperature coefficient of remanence $B(NdFeB) = 0.001K^{-1}$ for Neodymium-Iron-Boron (NdFeB) magnets, which are typically used in brushless DC motors. The acceleration of the motor's armature $\ddot{\varphi}$ is modeled by

$$\ddot{\varphi} = \frac{\mathrm{T}_{motor} + \mathrm{T}_{ext}}{J} \tag{2.23}$$

where T_{ext} is an external load and J is the armature's inertia tensor. This brushless DC motor model possesses a typical characteristic curve such as the one shown in Figure 2.13.

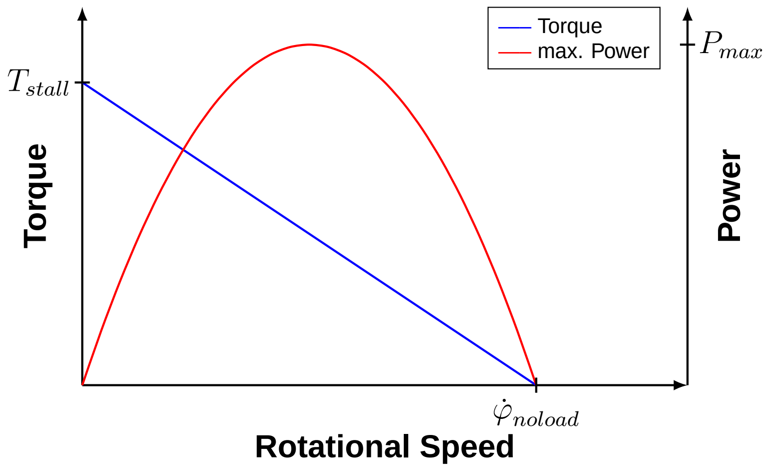


Figure 2.13: *Characteristic curve of a DC motor.*

Thermal Model

To derive the equations for the thermal model, a thermal equivalent electric circuit, shown in Figure 2.14, can be used. In compliance with Kirchhoff's current law, heat flow distributes to both paths within this parallel circuit:

$$P_{loss}(t) = \dot{Q}_C + \dot{Q}_R \tag{2.24}$$

$$\Delta T(t) = \Delta T_C = \Delta T_R + \underbrace{\Delta T_{C,env}}_{=0} \tag{2.25}$$

Thus, heat flow caused by a power loss $P_{loss}(t)$ can be described as

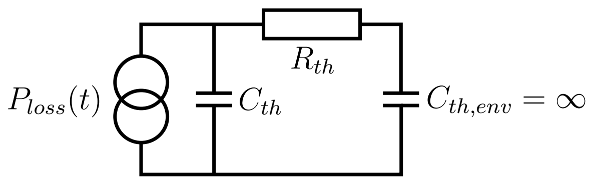


Figure 2.14: *Equivalent thermal circuit of a body heating up due to a heat flow source and heat lost to the environment.*

$$P_{loss}(t) = C_{th}\frac{\mathrm{d}\Delta T(t)}{\mathrm{d}t} + \frac{1}{R_{th}}\Delta T(t) \tag{2.26}$$

with the temperature difference $\Delta T(t)$ between a body with a given thermal resistance R_{th}, a heat capacity C_{th}, and a thermal time constant $\tau_{th} = R_{th}C_{th}$ and its environment is $T_{body} - T_{env}$. A Laplace transformation to the frequency domain reveals the equivalent transfer function for an easy implementation in *Simulink*:

$$P_{loss}(s) = C_{th}s\Delta T(s) + \frac{1}{R_{th}}\Delta T(s) \tag{2.27}$$

$$G(s) = \frac{\Delta T(s)}{P_{loss}(s)} = \frac{1}{\frac{\tau_{th}}{R_{th}}s + \frac{1}{R_{th}}} \tag{2.28}$$

Within the scope of this thesis, two bodies for thermal analysis are considered, namely the coil itself and the remaining parts of the motor. Thus, the coil temperature is given by

$$T_{coil}(t) = T_{env} + \Delta T_{coil,motor}(t) + \Delta T_{motor,env}(t) \tag{2.29}$$

2.4.2 Gearbox

As the rotational speed and the torque of a brushless DC motor are dependent on its diameter, gearboxes are required to appropriately alter these values for general or commercial aviation airplanes. Typically, a mechanical advantage greater than 1 is used, which lowers the output's rotational speed and increases the torque to suitable values. In the following, index 1 indicates the input side of the gearbox (DC motor side), whereas index 2 indicates the output side. With a given gear ratio G and no backlash, the transmission is described as

$$\begin{bmatrix}\varphi_1\\ \dot{\varphi}_1\end{bmatrix} = G\begin{bmatrix}\varphi_2\\ \dot{\varphi}_2\end{bmatrix} \tag{2.30}$$

where φ_2 is the position and $\dot{\varphi}_2$ is the velocity of the gearbox output, and φ_1 is the position and $\dot{\varphi}_1$ is the velocity of the gearbox input, which is also the DC motor's output. The transmission of the torques is inversed and can be written as

$$\mathrm{T}_1 = \frac{1}{G}\mathrm{T}_2 \tag{2.31}$$

with T_1 being the torque of the motor T_{motor} and T_2 representing the torque on the output side of the gearbox. The external torque in Equation 2.23 is the sum of external loads and losses due to friction and inefficiencies, so that

$$\mathrm{T}_{ext} = \frac{\mathrm{T}_2}{G} - \mathrm{T}_{Coulomb} - \mathrm{T}_{Viscous} \tag{2.32}$$

With the gearbox efficiency η_{Mech}, the coulomb losses are modeled by

$$\mathrm{T}_{Coulomb} = \begin{cases} (1-\eta_{Mech})\mathrm{T}_1, & \text{if } |\dot{\varphi}_1\mathrm{T}_1| > |\dot{\varphi}_2\mathrm{T}_2| \\ (1-\eta_{Mech})\frac{\mathrm{T}_2}{G}, & \text{if } |\dot{\varphi}_1\mathrm{T}_1| < |\dot{\varphi}_2\mathrm{T}_2| \end{cases} \tag{2.33}$$

depending on the power flow within the gearbox. The losses due to viscous friction are given by

$$\mathrm{T}_{Viscous} = \eta_V\dot{\varphi}_1 \tag{2.34}$$

where η_V is the viscous friction coefficient. Thus, with $\mathrm{T}_1 = \mathrm{T}_{motor}$ and T_2 being the torque at the gearbox output side $\mathrm{T}_{ext,gb}$, Equation 2.23 can be written as

$$\ddot{\varphi} = \begin{cases} \dfrac{\eta_{Mech}\mathrm{T}_{motor} + \frac{\mathrm{T}_{ext,gb}}{G} - \eta_V\dot{\varphi}_1}{J}, & \text{if } |\dot{\varphi}_1\mathrm{T}_1| > |\dot{\varphi}_2\mathrm{T}_2| \\ \dfrac{\mathrm{T}_{motor} + \eta_{Mech}\frac{\mathrm{T}_{ext,gb}}{G} - \eta_V\dot{\varphi}_1}{J}, & \text{if } |\dot{\varphi}_1\mathrm{T}_1| < |\dot{\varphi}_2\mathrm{T}_2| \end{cases} \tag{2.35}$$

2.4.3 Controller

Ultimately, an EMA is controlled by regulating the voltage that is applied on the coil. The authors of [LHH] use a cascade position controller, such as the one shown in Figure 2.15. The first cascade (outer loop) controls a command velocity vel_{cmd} using a proportional-integral controller based on the difference between commanded and actual position. The difference between this velocity command and the actual velocity is the control deviation that is fed to a proportional controller, which manipulates a command current i_{cmd} through the coil. The inner loop compares this command current with the actual coil current and controls a command voltage u_{cmd} by a proportional controller. Obviously, this command voltage is limited to the supply voltage $\pm u_{supply}$ of the system. If the EMA model represents an existing actuator, the data sheet helps to calculate the `Kp_cur` gain, as P_{max} and I_{max} should be listed

$$\texttt{Kp\_cur} = \frac{P_{max}}{I_{max}^2}. \tag{2.36}$$

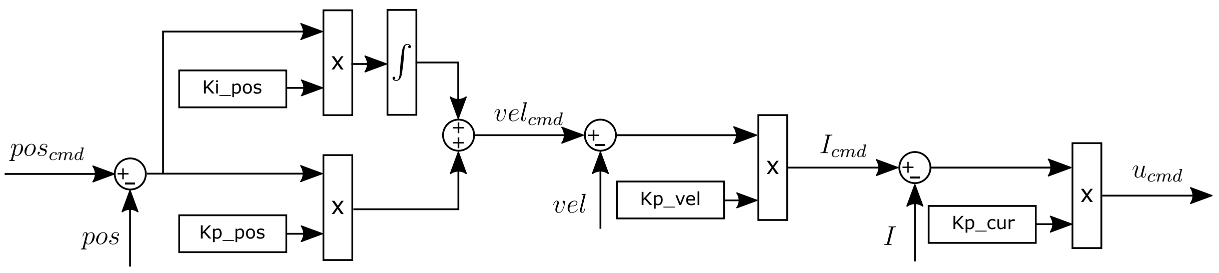


Figure 2.15: *Actuator cascade controller.*

2.5 Clutches

As already mentioned in the introductory chapter, clutches are used to intentionally break the power transmission at a certain point of a FCS. Within FCSs clutches are mostly safety-critical components that decouple actuators from the rest of the FCS in the event of an error.

Depending on the type of aircraft or even on the operating mode, there may be safety requirements that are completely contrary to those of other aircraft types or operating modes. For instance, clutches that link actuators to a mechanical FCS or clutches in a manned *optionally piloted vehicle* (OPV) must safely open in the case of an error (fail safe), whereas clutches which are installed in a UAV must never be opened inadvertently (fail secure). Furthermore, clutches should not allow a significant amount of free play, as this may affect the performance of flight control algorithms or even pilots. Additionally, they must not slip unintentionally, as this is similar to opening the clutch, which means a certain loss of control. Despite the great variety of clutches, electromagnetic tooth clutches are often used in FCSs, as they meet these requirements very well.

2.6 Sensors

Sensors represent the senses of an aircraft. They are technical devices that measure a certain physical property (e.g. temperature, pressure, acceleration) and output the quantity in the form of an electrical signal. This analog signal is typically converted into a digital signal using a *analog-to-digital converter* (ADC). A data interface then allows the communication with other components using standardized interfaces such as *Recommended Standard 232* (RS-232) or bus systems like *Controller Area Network* (CAN) (e.g. ARINC 825).

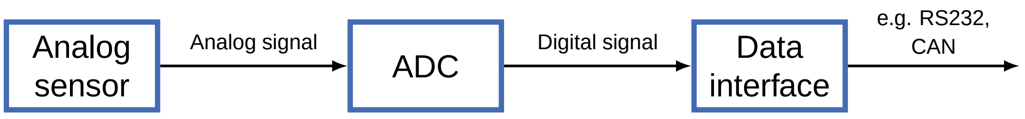


Figure 2.16: *Sensor signal processing.*

Sometimes the physical properties themselves are not returned but are used to derive further parameters, such as when measuring positions or angles. Today's aircraft use countless sensors to measure

- accelerations or angular rates (inertial sensors, accelerometers and gyros),
- static and dynamic pressure, temperatures, flow angles (air data sensors),
- pilot forces or control inputs,
- control surface deflections,
- engine parameters,
- position or velocity (global navigation satellite systems).

In the context of this thesis, sensors for measuring pilot forces and position sensors to determine control surface deflections or control inputs are of particular interest. Position measurements in commercial and large aircraft are mostly made using inductive measuring methods with *Linear Variable Differential Transformer* (LVDT) or *Rotary Variable Differential Transformer* (RVDT) sensors. These are electromechanical transducers that allow for measuring linear or rotary displacements. They enable the determination of the pilot control inputs or the control surfaces deflection. Using several sensors along one axis, even actuator runaways or broken links can be detected, for instance.

Forces within the control rods of a mechanical FCS can be measured using a strain-gauge or spring/LVDT-based instrumentation [Hon15]. Force sensors within the load-path of an FCS can be used to determine aerodynamic forces or to measure pilot forces. The latter is in particular used for a force-based disengagement of AP systems as is, for example, recommended by Airbus [Air93].

2.6.1 Error Characteristics

No sensors are ideal, instead they exhibit different types of error characteristics that can be grouped in three categories [Hol17c]:

Systematic Errors

Typical systematic errors are biases, scale factor errors, nonlinearities, or misalignments, and can be determined and removed by calibration.

Random Errors

The most popular random error is sensor noise, which originates for example from mechanical instabilities, vibrations, or electrical noise [Gro08]. Noise errors cannot be removed completely, but certain filters such as the Kalman filter can be used to minimize them.

Errors Due to Sensor Dynamics

Dedicated sensors, such as the vertical speed indicator or even accelerometers, exhibit a time-delayed output due to the way the values are measured.

3

A Bidirectional Method for Connecting CAD Tools with Simscape Multibody

This chapter describes the development of a method for bidirectional data transfer between CAD software and *Simscape Multibody*. The main purpose is to close the presented tool-chain gap and to translate CAD models automatically to MBS models. After a market overview over existing solutions, the requirements on the method are specified. Based on this information, a software architecture is designed and the implementation of essential features is described. The chapter concludes with a presentation of the developed optimization framework.

3.1 Market Overview

The world of interfaces between CAD systems and *MATLAB* or *Simscape Multibody* is fairly negligible. This section lists the most important and popular ones that come closest to the method developed within the scope of this thesis. Each interface application is examined in particular with regards to supported CAD systems, communication and interfaces, *Simscape Multibody* model building, and required manual action. A definition of the kinematic constraints within the CAD tools is premise of all listed software products.

3.1.1 Simscape Multibody Link

MathWorks has its own interface to the CAD world called *Simscape Multibody Link*. It is included in the installation of *MATLAB*; only the corresponding plug-ins for the CAD tools have to be downloaded and installed separately. So far, these plug-ins are available for SolidWorks®, PTC® Creo™, and Autodesk Inventor® [The16b]. Popular CAD tools like *CATIA* or *NX* are not yet supported.

Simscape Multibody Link and its plug-ins allow CAD assemblies to be exported from the corresponding tools to an XML file. This XML file contains all of the essential data for a *Simscape Multibody* model. At the same time, STL files are generated that allow a 3D visualization of the individual parts within the *Mechanics Explorer*.

Using the `smimport` function of the *MATLAB* engine starts the import of the CAD model into *Simscape Multibody* by building a model on the basis of the information stored in the XML file. Thereby the generated *Solid* blocks are automatically linked to their corresponding STL file.

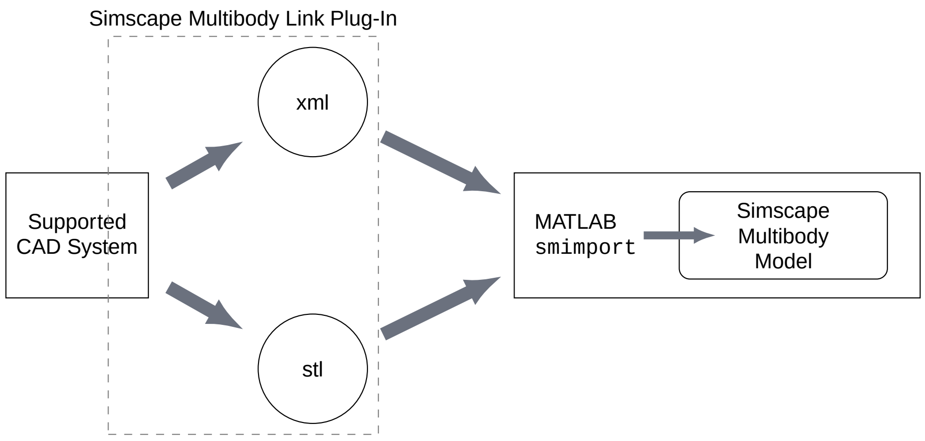


Figure 3.1: *Concept of Simscape Multibody Link.*

The final outcome is a translated and parametrized *Simscape Multibody* model with automated visualization capabilities. This free interface is appropriate if a CAD model needs only to be converted without any other communication between the CAD tool and *MATLAB*. However, there is clearly room for improvement, as the data transfer is file-based and many widespread CAD tools are not supported.

Table 3.1: *Features and functionality of Simscape Multibody Link.*

CATIA Connectivity	Bidirectional Connection	Controllable from MATLAB	MBS Model Building	Parametrized Model
-	-	✓	✓	✓

3.1.2 CAMAT

CAMAT (CATIA-MATLAB-Translator) was developed by employees of the *PROSTEP AG* as a proof of concept of a translator for generating *SimMechanics 1st Generation* models from *CATIA V5* models [BFS10]. The authors state that they selected these particular applications because they are very popular tools in their respective domains. The translator itself uses *CATIA*'s programming interface, *component application architecture* (CAA), which presents an API to the internals of the software [Dzo08]. The usage of this particular API requires an additional expensive license. However, it also provides profound access to *CATIA* and is used in this case to extract all relevant information, saving it in the form of an XML file. This intermediary XML file is created in the same manner that the *Simscape Multibody Link* interface XML files are created. Afterwards, the MBS model is built by importing and processing the XML file by the *MATLAB* engine. For visualization, *CAMAT* does not use the *Mechanics Explorer* but *CATIA* itself by actuating the corresponding joints.

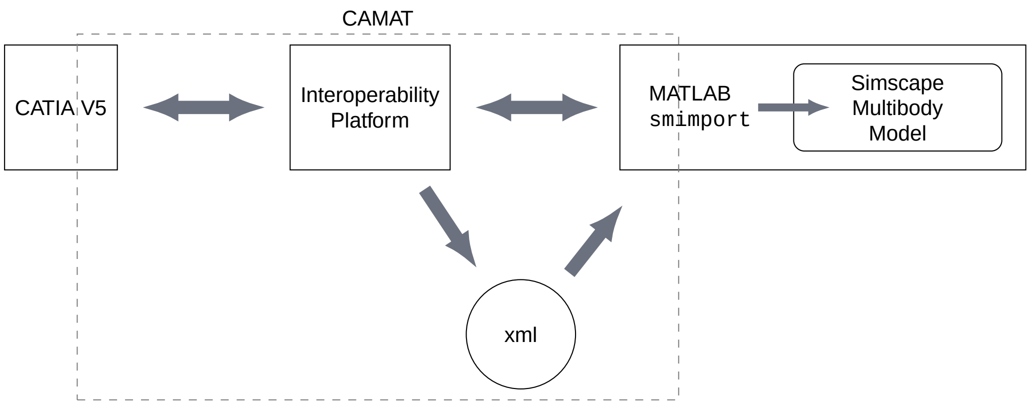


Figure 3.2: *Concept of CAMAT.*

When using *CAMAT*, both software tools are controlled remotely via the interoperability platform, a GUI. Unfortunately, this prevents the user from applying the broad functionality of the *MathWorks* suite to the CAD model.

With *CAMAT* being a proof of concept, it definitely has potential, but there are some downsides to the concept due to missing functionality and license costs. The interface is not able to change CAD parameters from *MATLAB*. Further, extra *CATIA* licenses have to be purchased because functions of the CAA library are used. Finally, no optimization functionality is available even though there is a bidirectional connection between the CAD tool and *MATLAB*.

Table 3.2: *Features and functionality of CAMAT.*

CATIA Connectivity	Bidirectional Connection	Controllable from MATLAB	MBS Model Building	Parametrized Model
✓	✓	-	✓	-

3.1.3 CAPRI

CAPRI (Computational Analysis Programming Interface), by *CADNEXUS*, is a CAD vendor-neutral API that can be used for a bidirectional connection between the geometric modeling world and analysis suites [DWH03]. In addition to an integration of CAD systems with *computer-aided engineering* (CAE) applications, *CAPRI* allows for automatic design updates triggered by human interaction and other software such as optimization tools [CADa]. Moreover, the interface is able to communicate with *MATLAB* using the *CAPRI MATLAB Connector* [CADb]. However, there is no opportunity provided to translate CAD models to *Simscape Multibody* models. CAD data can only be transferred to *MATLAB*, and *CAPRI* functions can be used in *MATLAB* scripts.

CAPRI does cover another important feature: bidirectional communication. As mentioned in the introductory chapter, communication in both directions is essential to an optimization of CAD parameters from *MATLAB*.

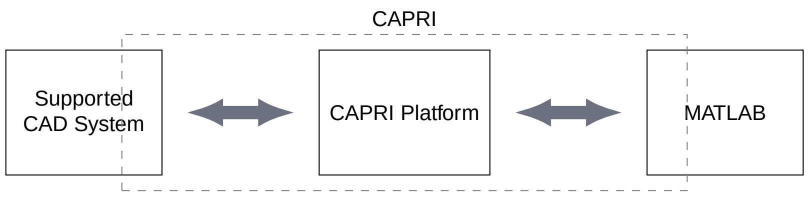


Figure 3.3: *Concept of CAPRI.*

Table 3.3: *Features and functionality of CAPRI.*

CATIA Connectivity	Bidirectional Connection	Controllable from MATLAB	MBS Model Building	Parametrized Model
✓	✓	✓	-	-

3.2 Basic Strategy and Overview

Before the method used in this thesis is specified, the basic strategy for its development is presented in this section. In terms of development processes, both worlds, the construction and the simulation world, complement one another very well, as the results of one influence the model of the other. Especially MBS requires mechanical and kinematic information that somehow already exists in a digital form in the CAD software. Therefore, the primary task of the method presented in this thesis is to abstract the CAD model and save it to a data model that can be used for the creation of an equivalent MBS model in *Simscape Multibody*. This requires the processing and translation of the mechanical and kinematic data to a form that is supported by the MBS software. Furthermore, algorithms must be developed that analyze relationships between components and calculate an adequate diagram structure in order to generate a reasonable and human manageable MBS model.

To ensure that the broad spectrum of *MATLAB* functionality can access CAD parameters, an implementation of the method in *MATLAB* itself is practical, as it ensures a seamless integration in the MathWorks world. The utilization of *object-orientated programming* (OOP) allows a combination of dividing the method into sensible units and defining a convenient data model. Separating the CAD-specific parts from the CAD-independent part enables a reasonable degree of modularity. Therefore, CAD-specific components are outsourced to DLLs, which establish a connection to the corresponding construction software using Microsoft's COM technology.

The *V5 Automation API* of Dassault Système's *CATIA V5* is accessible via the COM technology and allows for the capture of all the necessary information. In this way, the *CAAV5* API, which is associated with expensive licensing costs, can be omitted.

The method developed in this thesis helps to solve today's typical engineering problems when developing mechatronic products but focuses particularly on the design of FCSs. An optimization framework is developed that incorporates the method of this thesis and allows for a straightforward optimization of CAD and *MATLAB / Simulink* parameters based on results of a dynamic simulation. Naturally, this requires the communication between the software tools to be bidirectional in order to enable the modification of CAD parameters. In addition, models describing certain physical effects that occur in mechanical systems and can significantly affect system behavior are integrated. The parametrization of these models is thereby undertaken in the CAD software tools.

3.3 Requirements

The development process introduced in Chapter 1.4 requires functionality that the previously mentioned existing software solutions lack. This concrete process allows for deriving important requirements on the method developed in this thesis, which are formulated in this section. However, this collection of requirements is not intended to represent a complete *software requirements specification* but lists the essential and desirable features:

- **Bidirectional Connection Between CAD Tools and *MATLAB***: At the core of the method presented in this thesis is an intercommunication between CAD applications and *MATLAB*. This can be achieved by establishing a connection that allows for a data stream from the CAD tool to *MATLAB* and vice versa. Such a bidirectional data exchange is fundamental for an optimization of CAD models with respect to MBS results. It allows CAD parameters to be changed from *MATLAB* and the communication loop over both tools to be closed.

- **Support of *CATIA* V5**: As previously mentioned, the CAD software of *Dassault Systèmes* is one of the most widespread tools. Thus, the support of *CATIA* V5 by the developed method is essential.

- **Modular Concept for an Easy Connection of Additional CAD Systems**: The technical solution that establishes a bidirectional connection may not be *CATIA*-specific. It must be modular and has to provide the opportunity to easily attach additional CAD systems.

- **Provision of Necessary Data**: All the information that is required to automatically build a *Simscape Multibody* model has to be captured by the developed method. Table 3.4 gives an overview over required and optional data.

Table 3.4: *Data required for building a Simscape Multibody model.*

Object	Required Information
Parts	Instance name, mass, position of CG, inertia tensor, position, orientation
Joints	joint name, type, parts, direction, origin

- **Automatic *Simscape Multibody* Model Building**: The transfered CAD data needs to be processed automatically so that a *Simscape Multibody* model is created without any manual action. As such, the created model has to be be parameterized, meaning the data is stored in the workspace of *MATLAB* while the *Simscape Multibody* model references the corresponding variables.
- ***MATLAB* as Master**: The method must be controllable by *MATLAB*. This is necessary for applying *MATLAB* functions, such as an optimization algorithm, to the CAD model. *MATLAB* must be able, for example, to change CAD parameters and trigger an update of altered CAD data like masses or inertia tensors.

3.4 Specification of the Software Architecture

The specification of the approach presented in this thesis is the subject of this section. First boundary conditions for the solution are defined. Afterwards a high-level overview of the partitioning of functionality and responsibilities is provided. Finally, a class-based software architecture is presented.

3.4.1 Design Considerations

This subsection briefly deals with general constraints, dependencies, and objectives before focusing on the concept behind.

Operating Environment

The developed method is intended to be operated on a standard personal computer with at least *Windows*® 7 running as the operating system. Furthermore, the method is considered to be an "offline" interface, meaning both software tools as well as the method itself are meant to run on the same computer.

Development Environment

For the sake of completeness, the software environment used for the implementation of the approach is specified below:

- Microsoft® *Windows* 10 Version 1607

- MathWorks' *MATLAB* Version 2015b (english language)
- Dassault Systèmes' *CATIA* Version 5 Service Pack 7 Release 20 Hot Fix 54 (german language)
- Microsoft's *.NET Framework* Version 4.6.2
- Microsoft's *Visual Studio 2010* Version 10.0.40219.1 SP1Rel

Programming Paradigm

As part of coding, the paradigm of OOP is used, as this allows one to create classes that represent objects with certain attributes and certain behaviours in the form of methods. Since classes can be nested, OOP provides the opportunity to define a custom data structure, which is tailored to the data.

User Interface

In terms of the *user interface* (UI), the control is simple but not graphical. Since the focus of this thesis lies on functionality and not design, a *command-line interface* (CLI) is used instead of a GUI.

3.4.2 Partitioning of Functionality and Responsibilities

Before implementing the method, a basic concept or software architecture with proper partitioning of the functionality and responsibilities must be specified. Figure 3.4 shows a concept that satisfies all the requirements defined in Section 3.3.

The basis of the approach developed in this thesis is a framework coded in *MATLAB*, the controlling platform. In this way, the requirement that the method has to be controllable by *MATLAB* is ensured. Furthermore, as stated in the contributions of this thesis, a bidirectional connection provides a data exchange in both directions and closes the loop over both software tools, which is necessary for an optimization of CAD parameters from *MATLAB*. This is indicated by the bent arrows. Additionally, instead of creating an XML file and using the `smimport` command, the approach presented in this thesis stores the particular data in a custom data structure and builds a *Simscape Multibody* model programmatically.

In order to enable an operation with multiple CAD software tools, a certain degree of architecture modularity is reasonable. This can be achieved, for instance, by the separation of the functions that are specific to the individual CAD system and the ones that are independent of it. With such a separation, only the CAD-system-specific pieces need to be implemented in order to attach a new 3D modeling environment. The part containing the independent functions does not have to be changed. Collecting all functions that are specific to one CAD system in a DLL, which can be embedded in the *MATLAB* framework, is an appropriate and very practical technical solution that has many advantages.

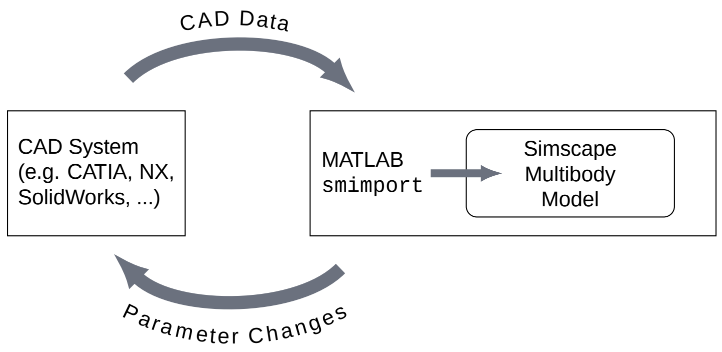


Figure 3.4: *Interface concept resulting from the requirements defined in Section 3.3.*

DLL files represent the shared library concept of Microsoft *Windows*, which provides a mechanism for shared code. As such, a DLL contains code or a collection of functions that might and can be used by multiple software programs at the same time. They are not executable directly, but they are intended to be loaded or integrated by external software.

Having all functions that are dependent on a 3D modeling tool packaged in such a DLL file allows new software to be easily attached by simply creating a new DLL without having to change the *MATLAB* framework itself. An interface concept of this kind is illustrated in Figure 3.5. However, this requires the specification of a generalized interface between the DLLs and the framework (see Appendix C). Every DLL has to match this generalized interface. In this way, each DLL establishes a connection to the particular CAD software and acts as an intermediate translator.

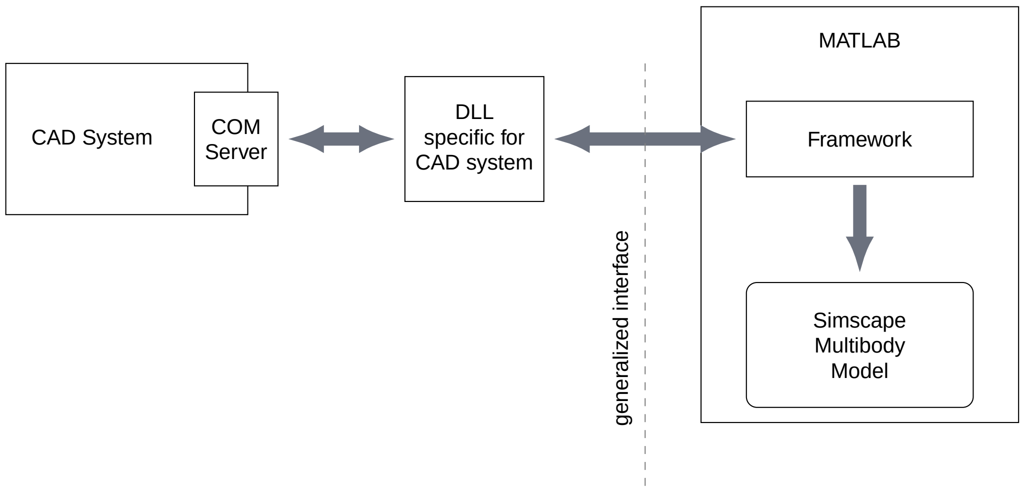


Figure 3.5: *Communication and partitioning of the interface.*

The separation of the DLL and the framework code in this solution has several advantages. For instance, the framework part must not be recompiled or even revealed in case of further DLL developments or updates. Especially in terms of updates, this is advantageous, since DLLs can be easily replaced if, for example, a CAD system's API changes. Moreover, they can be used by other software tools. In general, the programming language of the DLLs is independent of the *MATLAB* framework as long as the generalized interface can be satisfied.

The individual parts of the interface concept of Figure 3.5 and their responsibilities within the overall context in particular are discussed in the following.

COM Server

As Figure 3.5 shows, all of the CAD system's communication runs via its COM server. It makes the API of the CAD system accessible to all COM-capable programming languages using Microsoft's COM technology. This standard enables reusable software components and allows for easy interaction with these objects across process or software boundaries. The COM server therefore provides COM components or objects, including their properties and methods, to COM clients. After a client has connected to the server, it can create instances of the server's objects, which allow access to the data of the running process. Therefore, the client sets a pointer to the corresponding object instance, making it available within its own process (see Figure 3.6).

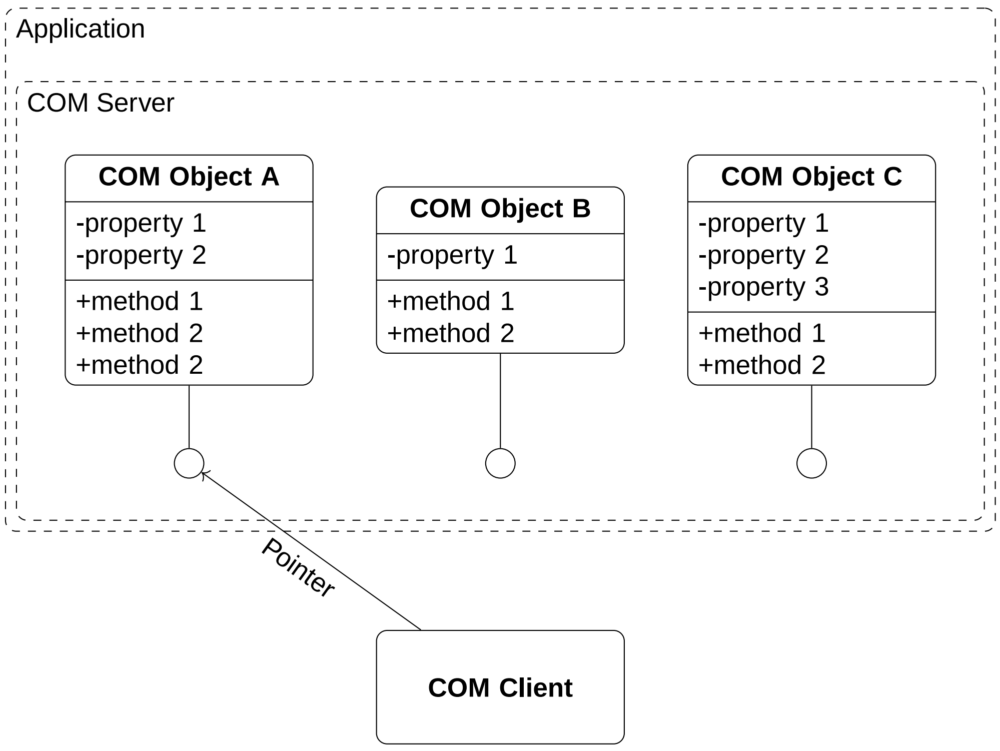


Figure 3.6: *Example of Microsoft's COM technology.*

In this thesis' approach, the DLL files are the COM clients that communicate with the CAD tools. The counterpart, the COM server of the CAD software tools, is not developed within the scope of this thesis but comes already with the programs and is addressed by the DLLs. It is used for communicating with the API and enables one to create objects of sessions, assemblies, or parts, for example. It is essential that the properties and methods of these objects provide all the information required to build a *Simscape Multibody* model automatically. The API is a sort of a bottleneck in terms of the data exchange. If it is not able to supply the needed data, the CAD tool is not suitable for the method developed in this thesis.

CAD Specific DLL Files

The DLL files that represent the corresponding COM client for the CAD systems are an important part of the approach. They have to be tailored to the particular CAD software and must match the generalized interface to the *MATLAB* framework. Consequently, they do not yet come with the software programs. Instead, they must be developed for each CAD tool. Regarding the communication between the DLLs and *MATLAB*, the .NET framework, a successor to the COM standard, is used. It enables the DLL files to use features of the .NET framework and allows for easy integration of these files into *MATLAB* by simply using the `NET.addAssembly()` command. This enables it to access all public functions of the DLL.

The main task of the DLLs is to handle the communication between two particular software tools. They act as a connector and translate generalized high-level commands into CAD system specific algorithms. The DLLs are in particular responsible for the bidirectional communication. They gather all the necessary data and send it back in a form that matches the generalized interface. This is realized by functions that collect specific data and store it in the return value. These functions are then called from the *MATLAB* framework. Moreover, the data exchange from *MATLAB* to the CAD software is also realized within the DLL. For that reason, the DLLs contain a method that addresses user-defined parameters in the CAD model and modifies their values according to the input arguments.

MATLAB **Framework**

The *MATLAB* framework is the essential part of the approach presented in this thesis. It represents the *human-machine interface* (HMI) and has control of the entire software package. The HMI, which is basically constituted by *MATLAB*'s command line, allows the user to control the method. A command from the user triggers the execution of the corresponding algorithms, such as connecting to a CAD tool, opening a model, or transferring CAD data. In addition, the HMI is used to display information that might be relevant for the user.

To be able to control the CAD system or the whole software package, the CAD-specific DLL must be embedded.

Additionally, the framework has two other important responsibilities. On the one hand, it manages the entire data storage. The framework contains a system of several classes, which is specially tailored for storing CAD related data. These class definitions are used to create corresponding objects, such as parts, joints, or constraints. On the other hand, the framework processes this data and automatically builds an equivalent MBS or *Simscape Multibody* model.

3.4.3 Class-Based Framework Architecture

This subsection focuses on the class structure of the *MATLAB* framework, part of the developed method. Figure 3.7 provides a simple overview of the framework's classes. It allows one to grasp the relation between the classes by providing two characteristics, namely the navigability and the type of the relation. More details on the classes can be found in Appendix B. If a connection between two classes includes an arrow tip, navigation is only possible in the arrow's direction. However, if it is a line without an arrowhead, it can be navigated in both directions. Furthermore, a dashed connecting line indicates a reference to an already existing instance of a class. On the other hand, if a line is solid, the class instance object is truly created and a property or part of the creating object.

In the following, the relationships and responsibilities of the classes are briefly discussed. Classes with a lined background in Figure 3.7 represent the data storage classes, which are primarily used for structuring and saving the CAD data. They perform very little calculations and data processing and offer barely any functionality. Contrary to these data storage classes, the unshaded functional classes are basically responsible for controlling the interface and process data.

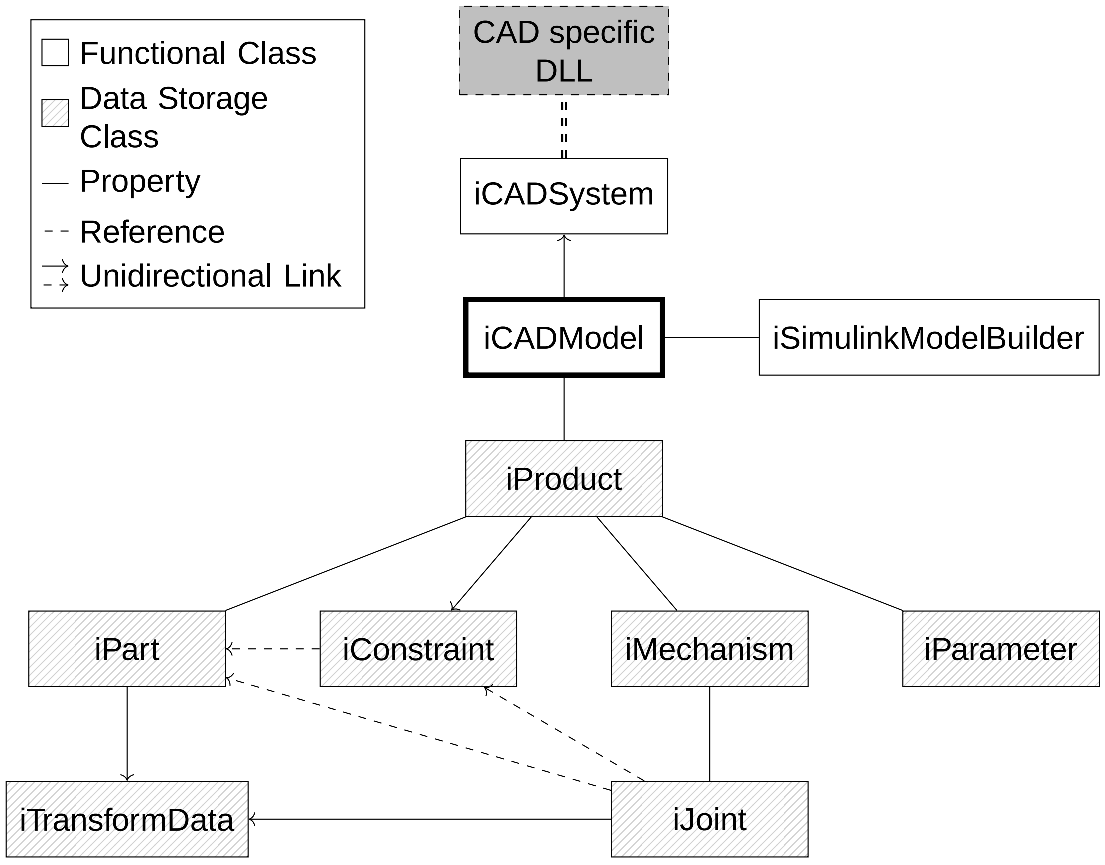


Figure 3.7: *Overview of the framework's classes.*

The central class of the framework is `iCADModel`, of which the user creates an object instance to control the framework. As a sort of root object, it encapsulates the entire framework, including the CAD data, and is stored in the *MATLAB* workspace. It creates an instance of the `iCADSystem` class, which is stored as a property of the `iCADModel` object. The `iCADSystem` object defines the generalized interface and includes one of the CAD specific DLLs to establish a connection to a particular CAD software.

Any data that `iCADModel` receives from `iCADSystem` is processed and saved in a hierarchical structure in the form of objects of the data storage classes. In addition to this hierarchy, objects are partially cross-linked by references, which allows for further navigational opportunities.

The `iCADModel` class also creates an instance of the `iSimulinkModelBuilder` class and saves it as a property as soon as it is required. As part of the `iCADModel` object and due to the bidirectional linking of the classes it is able to access the objects of the data storage classes when building a *Simscape Multibody* model. These objects also serve as the models' databases and are referenced when parameterizing the model.

3.5 Implementation of the Key Functionality

As previously mentioned, a description of the implementation of the individual classes is intentionally omitted here because of the amount of irrelevant information but can be found in Appendix B. Instead, the key functions of the approach presented in this thesis are explained in the following. For a manual with more detailed information about the application of the method, see Appendix A.

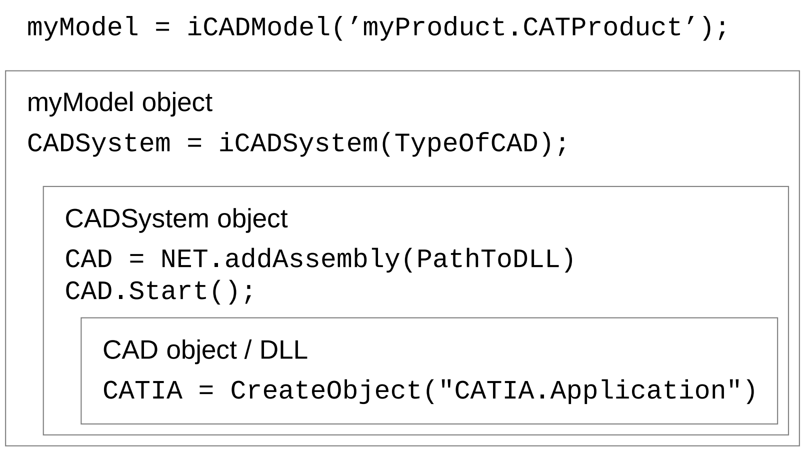
3.5.1 Connecting MATLAB and the CAD Software

The connection between *MATLAB* and the CAD software is established as soon as the user creates an instance of the `iCADModel` class. Consequently, an argument is passed that contains the path to the assembly or product file of the CAD model. Depending on the file extension, the CAD system is determined and a new instance of the `iCADSystem` class is created within the `iCADModel` object. Depending on the software, the corresponding CAD-specific DLL is then made accessible to the framework. Each DLL provides a `Start()` method, which creates a new object or instance of the CAD software. The nested function calls are shown in Figure 3.8.

3.5.2 Transferring CAD Data to MATLAB

The method developed in this thesis contributes to a more integral development process for mechatronic products by closing the gap between design and simulation software. The method must therefore transfer specific data from the CAD tool to *MATLAB*. This process is described in the following.

The transfer of CAD data is initially triggered by the user with an object of the type `iCADModel`. The object provides a `GetCADData` method, which sequentially calls functions from the embedded DLL. This returns the specific data. The communication



```
myModel = iCADModel('myProduct.CATProduct');
```

myModel object

```
CADSystem = iCADSystem(TypeOfCAD);
```

CADSystem object

```
CAD = NET.addAssembly(PathToDLL)
CAD.Start();
```

CAD object / DLL

```
CATIA = CreateObject("CATIA.Application")
```

Figure 3.8: *Establishing a connection to the CAD Software.*

concept is shown in Figure 3.9 while the mentioned sequence is illustrated in Figure 3.10.

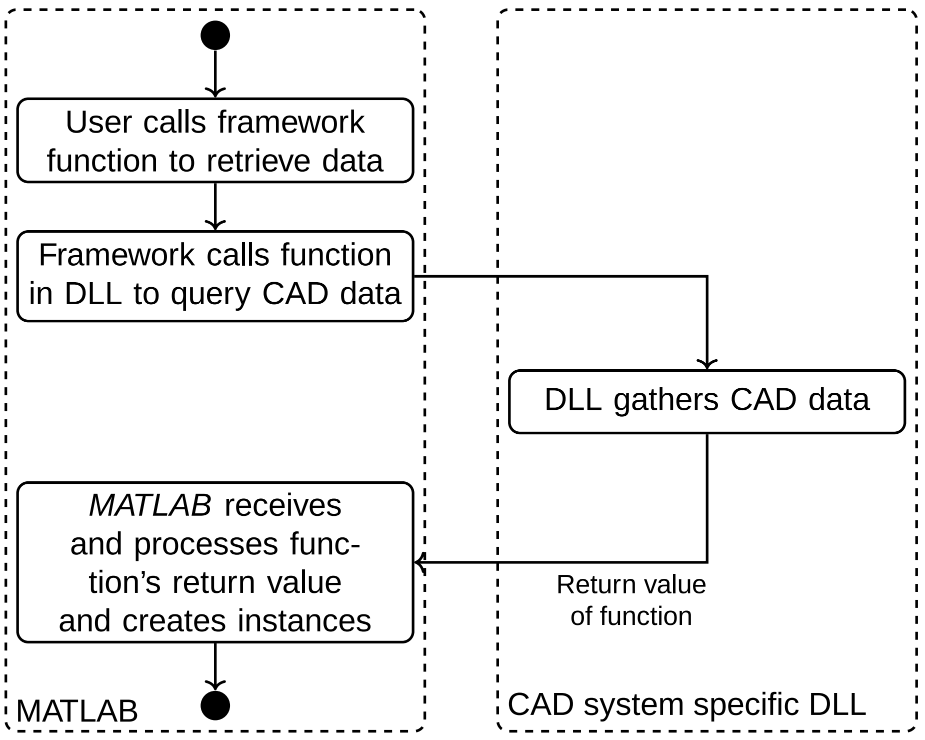

User calls framework function to retrieve data
Framework calls function in DLL to query CAD data
DLL gathers CAD data
MATLAB receives and processes function's return value and creates instances
Return value of function
MATLAB
CAD system specific DLL

Figure 3.9: *Activity diagram of the data transfer concept.*

First, all data regarding parts and products is transferred, followed by constraints, joints, and finally, the user-defined parameters. The split of the functions might be slightly less efficient for an initial transfer, but it allows particular functions to be reused and redundant information to be skipped when, for instance, only a subset of the CAD data is updated during optimization. As such, a significant amount of time can be saved in these cases. For the same reason, the data capture is clustered instead of having each parameter queried individually by *MATLAB*, which is less time efficient due to the many function calls.

The data of a complete transfer is listed in Table 3.5. The position and orientation

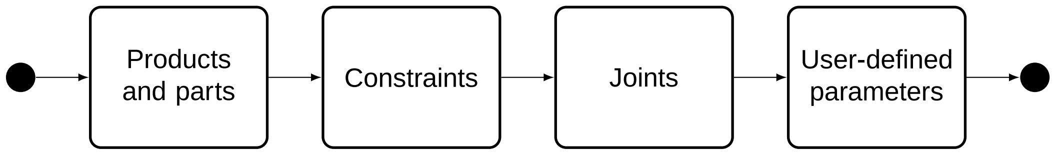


Figure 3.10: *Sequential steps of the CAD data transfer.*

of products and parts is noted in the world frame, while the CG position and the inertia tensor are noted in the coordinate frame of the particular body. Constraints themselves

Table 3.5: *Required CAD Data.*

Object	Required Information
Products	Instance name, part name, position, orientation, parts (optional: filename)
Parts	Instance name, product name, mass, position of CG, inertia tensor, position, orientation (optional: filename, material density, material Young's modulus, color)
Constraints	Parent product, constraint name, type, geometry, parts, direction, origin
Joints	Parent product, mechanism name, joint name, type, constraints

cannot be created directly in *Simscape Multibody*. In *CATIA*, joints more-or-less represent a collection of constraints so that the essential information regarding the associated parts, position, or orientation is not a property of the joint objects themselves. For this reason, data related to constraints is transferred to gather the essential joint data. The geometry attribute essentially returns the dimension of the constraint and can either be a point, a line, or a plane in this thesis. This information is particularly important for the interpretation of the constraint or joint direction when generating joint coordinate frames in post-processing (see Subsection 3.5.3). The constraint type is primarily used to determine parts that are fixed in 3D space. Considering future features, the geometry and type attributes can also be further processed for an automatic determination of joint types based on constraint data, for example. Regarding joints, the essential information is the joint type, which defines the DoFs at the end. In Figure 3.11, all joint types that are supported by the approach of this thesis as well as their DoFs (orange) are shown. As described, the joint direction, orientation, and the associated parts are gathered over the corresponding constraints.

Updating CAD Information

A modification of user-defined parameters might very likely result in a change of part dimensions, part positions, or orientations. However, assuming that the topology of the model has not changed, only a subset of parameters may have been affected by the modification. The number of parts, part names, or joint types, for instance, still remain the

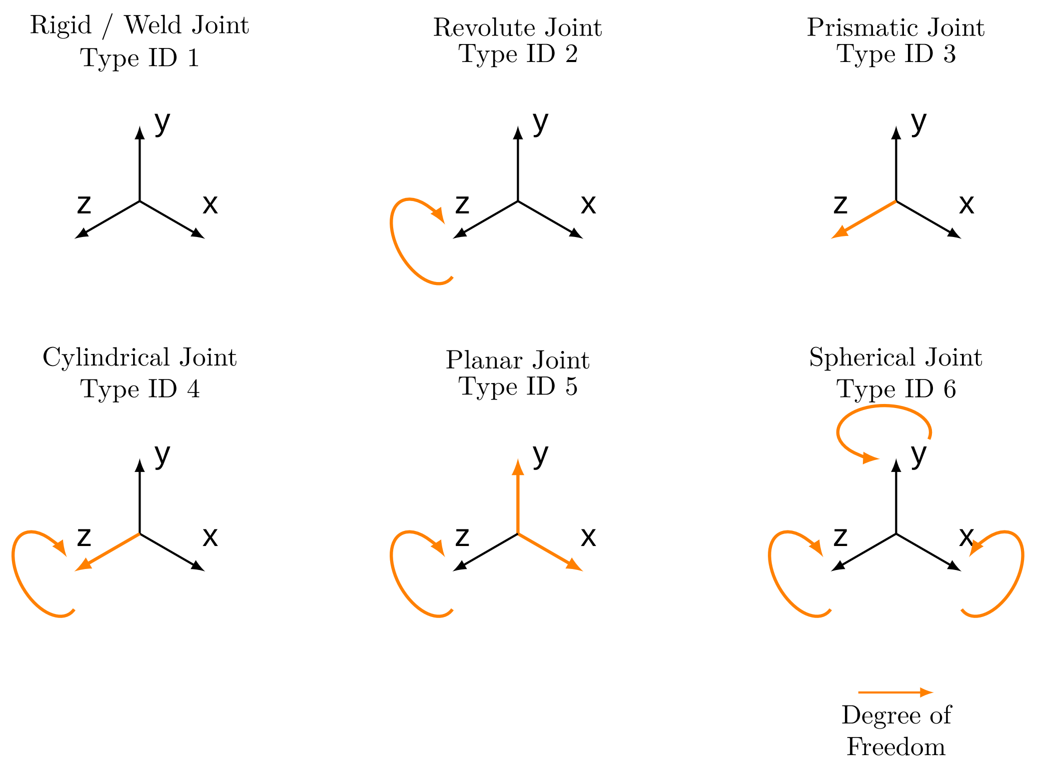


Figure 3.11: *Overview of joints supported by the interface with their degrees of freedom.*

same and do not have to be queried again. Omitting these unchanged values when querying an updated dataset is time-saving, which is particularly useful for iterative parameter changes.

To enable such a time-efficient data update, the `UpdateCADData` method is implemented. It is assumed that parameters are not used to change the CAD model topology but to parameterize, for example, dimensions, positions, or material constants. This enables one to query only a subset of parameters in a time-optimized manner and to skip time-consuming data sources that are unaffected by a parameter change, such as products, joints or user-defined parameters. Nevertheless, user-defined parameters are included in the update query by default, but the method allows them to be omitted with an optional boolean argument when required. The individual sequential steps that are performed to update the CAD data are illustrated in Figure 3.12.

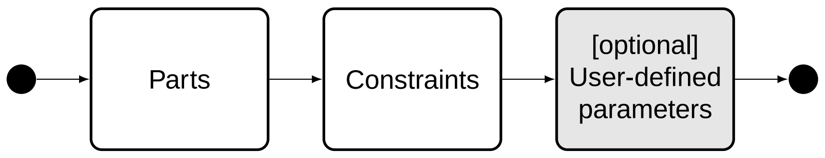


Figure 3.12: *Sequential steps for updating CAD data.*

3.5.3 Data Storage and Post-Processing

The data that is returned by the DLL methods to the `GetCADData` or `UpdateCADData` method is then post-processed. Depending on the data, objects or instances of the corresponding class are created, to which the transferred data is assigned. At the end of the process, the data structure is generated by nesting objects and their properties. Apart from generating the data structure, some of the data has to be post-processed. In *Simscape Multibody*, solids and joints are positioned and oriented relatively to each other. Since positions and orientations which come from the CAD software are notated in the global coordinate system, relative translations and rotations between them have to be computed for a sequential modeling. Therefore, joint coordinate frames are generated and quaternions are calculated to describe relative orientations of parts and joints.

Data Structure of the MATLAB Framework

Class properties and the nesting of objects that result from the specification of the classes (see Figure 3.7) determine the data structure of the *MATLAB* framework. In Figure 3.13, the CAD model data is shown. For a complete overview of the implementation, see Appendix B.

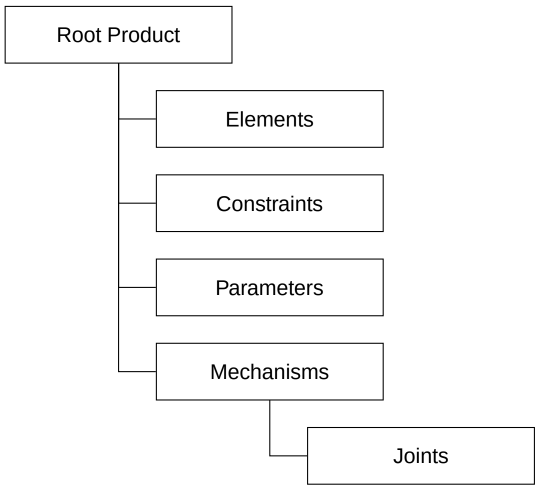


Figure 3.13: *Data structure of the MATLAB framework.*

The data structure is mainly based on the tree structure of *CATIA V5*, since this tool was the only one available at the time of the software specification. Also at this time, it was assumed that the data or tree structure of other common CAD tools was similar to that of *CATIA*, which is illustrated in Figure 3.14. In order to minimize the effort for the development of new DLL files, the scope of the DLL code must be as small as possible. For that reason, the data storage or class structure of the presented method is similar to the data structure of *CATIA V5*.

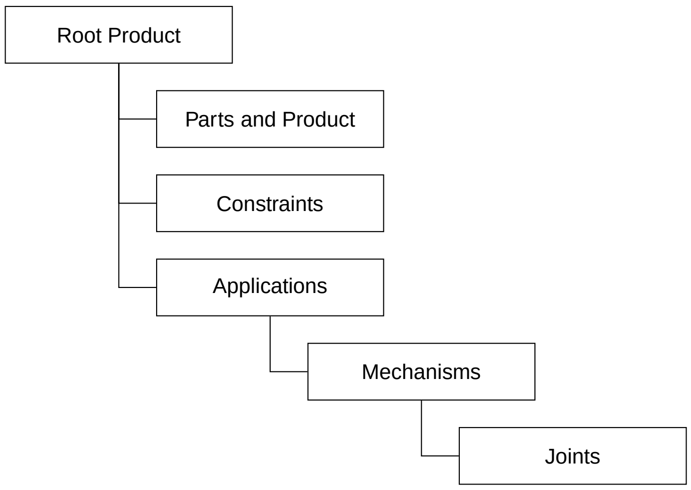


Figure 3.14: *Tree structure of CATIA.*

Calculation of Quaternions

Quaternions are used for the attitude representation of solids or joints with respect to reference coordinate frames and are commonly applied in kinematics since they offer advantages over the *Euler* angles. According to [GY12], there is no singularity such as the gimbal lock and the computational costs are fewer. The basis for the widespread quaternion representation is *Euler*'s rotation theorem. It states that the relative orientation of any two coordinate frames can always be expressed by exactly one rotation about an axis.

A quaternion is a vector with four components, where one component is the quaternion's scalar part q_s and the other three represent the vector $\vec{q_v}$. While the position of the scalar part varies, the quaternion definition of the *MATLAB Aerospace* toolbox, with the scalar part being the first component, is applied in this thesis:

$$q = \begin{bmatrix} q_s \\ \vec{q_v} \end{bmatrix} = \begin{bmatrix} q_s \\ q_x \\ q_y \\ q_z \end{bmatrix} \tag{3.1}$$

In the following, the methods described in [GY12] are used to calculate a quaternion from two given coordinate frames or an angle and a corresponding axis.

For the determination of a quaternion, which represents the rotation of a coordinate frame A to another coordinate frame B, a *direction cosine matrix* (DCM) is first described. The given coordinate frames A and B are noted in the global coordinate frame / world frame W:

$$(A)_W = \begin{bmatrix} \vec{i}_A & \vec{j}_A & \vec{k}_A \end{bmatrix}_W \tag{3.2}$$

$$(B)_W = \begin{bmatrix} \vec{i}_B & \vec{j}_B & \vec{k}_B \end{bmatrix}_W \tag{3.3}$$

with the unit vectors $\vec{i}, \vec{j}, \vec{k}$ in the direction of their x, y, and z axes. The DCM M_{BA} is created according to

$$M_{BA} = \begin{bmatrix} \vec{i}_A \cdot \vec{i}_B & \vec{j}_A \cdot \vec{i}_B & \vec{k}_A \cdot \vec{i}_B \\ \vec{i}_A \cdot \vec{j}_B & \vec{j}_A \cdot \vec{j}_B & \vec{k}_A \cdot \vec{j}_B \\ \vec{i}_A \cdot \vec{k}_B & \vec{j}_A \cdot \vec{k}_B & \vec{k}_A \cdot \vec{k}_B \end{bmatrix} \tag{3.4}$$

Afterwards, the trace of the DCM is determined by summing the elements of the main diagonal as

$$tr(M_{BA}) = \sum_{i=1}^{3} m_{ii} = m_{11} + m_{22} + m_{33} \tag{3.5}$$

If the trace is > 0, the scalar component q_s and the vector component $\vec{q_v}$ of the quaternion can be calculated by

$$q_s = \frac{1}{2}\sqrt{tr(M_{BA}) + 1} \tag{3.6}$$

$$\vec{q_v} = \begin{pmatrix} \frac{m_{32} - m_{23}}{2\sqrt{tr(M_{BA}) + 1}} \\ \frac{m_{13} - m_{31}}{2\sqrt{tr(M_{BA}) + 1}} \\ \frac{m_{21} - m_{12}}{2\sqrt{tr(M_{BA}) + 1}} \end{pmatrix} \tag{3.7}$$

If $tr(M_{BA}) \leq 0$, the greatest element of the diagonal values of M_{BA} determines the calculation. In the case that m_{11} is the greatest diagonal value, the components are calculated as follows:

$$q_s = \frac{m_{32} - m_{23}}{2\sqrt{1 + m_{11} - m_{22} - m_{33}}} \tag{3.8}$$

$$\vec{q_v} = \begin{pmatrix} \frac{1}{2}\sqrt{1 + m_{11} - m_{22} - m_{33}} \\ \frac{m_{21} - m_{12}}{2\sqrt{1 + m_{11} - m_{22} - m_{33}}} \\ \frac{m_{13} - m_{31}}{2\sqrt{1 + m_{11} - m_{22} - m_{33}}} \end{pmatrix} \tag{3.9}$$

Whenever m_{22} is the biggest element, the quaternion is defined as

$$q_s = \frac{m_{13} - m_{31}}{2\sqrt{1 - m_{11} + m_{22} - m_{33}}} \tag{3.10}$$

$$\vec{q_v} = \begin{pmatrix} \frac{m_{21} - m_{12}}{2\sqrt{1 - m_{11} + m_{22} - m_{33}}} \\ \frac{1}{2}\sqrt{1 - m_{11} + m_{22} - m_{33}} \\ \frac{m_{32} - m_{23}}{2\sqrt{1 - m_{11} + m_{22} - m_{33}}} \end{pmatrix} \tag{3.11}$$

Finally, for the case where m_{33} is the maximum of the diagonal values, the quaternion components are calculated by

$$q_s = \frac{m_{21} - m_{12}}{2\sqrt{1 - m_{11} - m_{22} + m_{33}}} \tag{3.12}$$

$$\vec{q_v} = \begin{pmatrix} \frac{m_{13} - m_{31}}{2\sqrt{1 - m_{11} - m_{22} + m_{33}}} \\ \frac{m_{32} - m_{23}}{2\sqrt{1 - m_{11} - m_{22} + m_{33}}} \\ \frac{1}{2}\sqrt{1 - m_{11} - m_{22} + m_{33}} \end{pmatrix} \tag{3.13}$$

For a given rotation with an axis $\vec{r}$ and an angle ϕ, the corresponding quaternion can be calculated according to

$$q = \begin{bmatrix} \cos\left(\frac{\phi}{2}\right) \\ \vec{r} \cdot \sin\left(\frac{\phi}{2}\right) \end{bmatrix} \tag{3.14}$$

Calculation of the Joint Coordinate System

As mentioned previously, joints are basically bundles of constraints, so CAD systems most likely do not provide corresponding coordinate frames. However, in *Simscape Multibody*, each joint has two coordinate frames, between which a relative motion is allowed, depending on the joint type. In order to ensure a proper model building, these two coordinate frames must have the same orientation after an initial transfer. Otherwise, the initial position or rotation states of joints are not zero, which might cause problems, as *Simscape Multibody* reverts them to zero as long as corresponding state targets are not set. For this reason, one generic, temporary coordinate frame must be generated for each joint, which is used to calculate relative orientations.

An efficient way to generate the generic joint coordinate frame is to rotate a reference frame from a linked part in a way that its z-axis (which is the relevant one for the DoFs of joints in *Simscape Multibody*) points in the same direction as the constraint or joint direction. This can be done by determining the quaternion which describes such a rotation. Subsequently, the rotation can be applied to the unit vectors of the reference frame. A further benefit is that the relative orientation of this particular joint-part pair is now already computed in the form of a quaternion and can be directly assigned to the joint object. According to Equation 3.14, only the rotation axis and the rotation angle are required. If $\vec{z}$ represents the direction of the reference frame z-axis and $\vec{d}$ is the direction of the joint, then the required angle ϕ for the rotation can be calculated as follows:

$$\cos\phi = \frac{\vec{z} \circ \vec{d}}{|\vec{z}| \cdot |\vec{d}|} \tag{3.15}$$

whereas the axis for the rotation $\vec{r}$ is simply derived from the cross product:

$$\vec{r} = \vec{z} \times \vec{d} \tag{3.16}$$

A vector $\vec{v}$ can be rotated using a quaternion q and its conjugated version q^*:

$$\begin{bmatrix} 0 \\ \vec{v}\,' \end{bmatrix} = \left(q \begin{bmatrix} 0 \\ \vec{v} \end{bmatrix} \right) q^* \tag{3.17}$$

where the conjugated has an inverted vector part. As the multiplication of quaternions is non-commutative, the execution order is important, which shall be symbolized by the brackets. After [GY12], two quaternions can be multiplied by

$$q = q_1 q_2 = \begin{bmatrix} q_{s,1} \cdot q_{s,2} - \vec{q}_{v,1} \circ \vec{q}_{v,2} \\ q_{s,1} \cdot \vec{q}_{v,1} + q_{s,2} \cdot \vec{q}_{v,1} + \vec{q}_{v,1} \times \vec{q}_{v,2} \end{bmatrix} \tag{3.18}$$

However, for the quaternion multiplication and the calculation of the conjugated quaternion, the *MATLAB Aerospace* toolbox provides the `quatmultiply` and the `quatconj` functions, which are used in this approach.

After all three unit vectors of the reference frame are rotated, the joint coordinate frame is generated. Naturally, there are joints that do not have a certain direction, like spherical joints. In these cases, the joint coordinate frame is set to an identity matrix I_3. As soon as the joint coordinate frame is available, all other relative orientations or quaternions involving this joint can be calculated according to the method using two coordinate frames, which is described in the previous section.

Calculation of the Relative Joint Transformation Data

The methods for calculating the relative orientation are already described in the previous sections. In the following, the general approach to calculate the transformation data of joints, that is the relative orientation and translation to the connected parts (or the world frame), is described. The implemented method calculates this information and saves it in the form of `iTransformData` objects. Every joint has two connected frames, and the calculation is performed for each connection.

The implemented method first checks the joint type, since weld joints, which provide no DoFs at all, are treated differently. The orientation and position of a weld joint are irrelevant, because the *Weld* joint block simply keeps its coordinate frames congruent. With these aligned to the coordinate frame of one of both parts, only the relative orientation to the coordinate frame of the second part must be determined. The same applies to the data for the relative translation which is calculated by a trivial vector subtraction. This is subsequently multiplied with the corresponding transformation matrix for a notation in the correct coordinate frame. Let $(O_1)_W$ be the matrix of the unit vectors of the first part's coordinate frame, $(\vec{P_1})_W$ be the position of the first part, and $(\vec{P_2})_W$ be the position of the second part. Then the translation noted in the coordinate frame of the first part O_1 is:

$$\left(\vec{t}\right)_{O_1} = f((O_1)_W, (\vec{P_1})_W, (\vec{P_2})_W) = ((O_1)_W)^\intercal \left((\vec{P_2})_W - (\vec{P_1})_W\right) \tag{3.19}$$

For weld joints, only one transformation dataset is determined. For other joint types, there are two datasets describing the transformation to the other two connected parts or frames. In these cases, the method checks whether there is already a coordinate frame for the joint. If there is not, it is generated based on the coordinate frame of the part, and the quaternion is saved as a by-product. Otherwise, the quaternion for the relative orientation to the part's coordinate frame is determined. The translational data is derived analogous to Equation 3.19 but in this case $(\vec{P_2})_W$ represent the joint position.

3.5.4 Automatic Simscape Multibody Model Building

In order to automatically convert a CAD model to an MBS model, a method is developed to build the corresponding *Simscape Multibody* model on the basis of the transferred CAD data. Basically, the algorithm uses three different block types to create a model, which are *Solid*, *Joint*, and *Rigid Transform* blocks. While parts are always represented by a *Solid* block, there is a variety of joint blocks available, where the transferred joint ID is used to choose the equivalent one. However, in the case of a revolute joint, which is supposed to have backlash, two translatory DoFs perpendicular to the rotary axis are required. For this reason, instead of a *Revolute Joint* block a *Planar Joint* block is used, which provides exactly this combination of DoFs. Finally, the relative position and orientation of parts and joints is achieved via *Rigid Transform* blocks. Depending on the user-defined CAD parameters, additional friction or backlash models are added to particular joints. In order to create decent diagrams, an auto layout algorithm is implemented, which determines sensible coordinates for each block in advance.

The method adds these blocks programmatically and parameterizes them with the transferred data. Instead of copying the data, the paths to the corresponding locations in the data structure in the *MATLAB* workspace are set. This is particularly beneficial when the CAD model changes and the MBS model needs to be adjusted. Due to the parametrization, there is no need to change the *Simscape Multibody* model itself. Only the data in the *MATLAB* workspace requires the new, updated information.

Determination of Sensible Block Coordinates

Building a model automatically, easily leads to confusing diagrams. For this reason, the method presented in this thesis aims for a clear model structure and determines reasonable coordinates for each block. Therefore, a technique from graph theory is used. The *Breadth First Search* (BFS) is an algorithm for traversing graph structures and is typically used

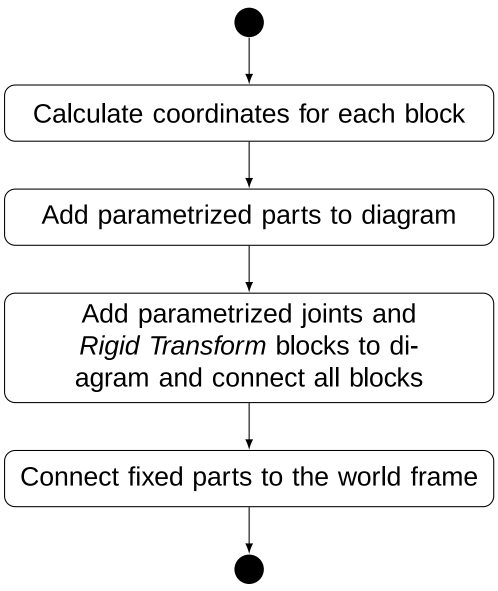

Calculate coordinates for each block
Add parametrized parts to diagram
Add parametrized joints and *Rigid Transform* blocks to diagram and connect all blocks
Connect fixed parts to the world frame

Figure 3.15: *Activity diagram of the model building algorithm.*

to find the shortest path between two vertices [CH05]. The technique discovers vertices level-by-level. Therefore, first all of the direct adjacent vertices are explored (which were not visited yet) before the search continues at the next level.

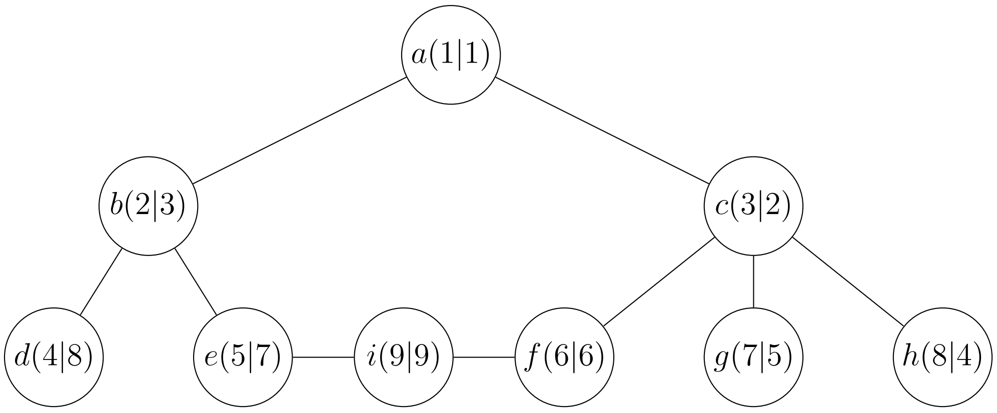


Figure 3.16: *Network traversed by "left-to-right" and "right-to-left" BFS.*

In order to determine the x and y coordinates of each block, the BFS algorithm is applied twice to an adjacent matrix. This matrix is created based on the parts, joints, and their connections and represents the model structure or the block network. First the BFS is performed in a "left-to-right" manner for the x-axis and then in a "right-to-left" manner for the y-axis. The coordinates of each block result from the order in which they are found. In the vertices of the example network shown in Figure 3.16, the vertex identifier and the resulting x and y coordinates are given. Diagrams generated by this method are usually elongated and expand at an angle of 45 degrees to the upper right. To obtain a diagram that extends from left to right, a rotation is applied. For the exemplary network, the resulting diagram with BFS coordinates is illustrated on the left in Figure 3.17. As can be seen, the distance between the individual layers varies significantly. Thus, the algorithm adjusts the horizontal spacing to an equidistant distribution after rotating the coordinates, in order to keep diagrams practical (see on the right in Figure 3.17).

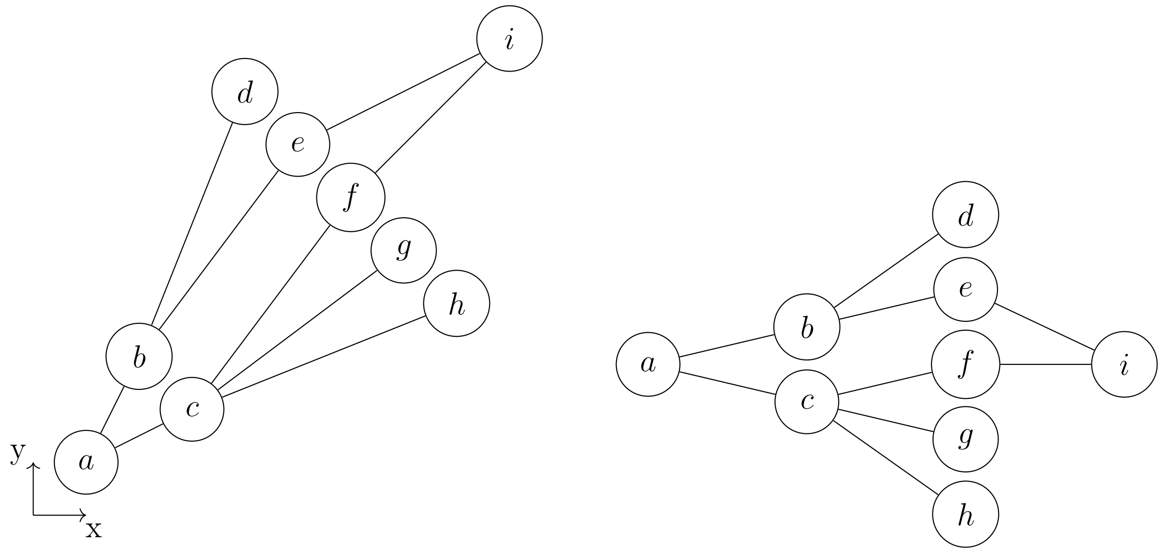


Figure 3.17: *BFS and rotated, condensed diagram layout.*

Consideration of Friction and Backlash in Revolute Joints

As stated in the contributions of this thesis, the developed method automatically adds models for friction and backlash to user-selected revolute joints. In the analysis of mechanical systems, especially with regard to mechanical FCSs, friction and backlash in joints can have a significant influence on system behavior. Particularly in mechatronic systems, these non-linear effects often cause control problems [Hac15]. Therefore, it is reasonable to include these effects in a simulation. In terms of friction, *Simscape Multibody* does not offer a direct setting. Utilization of *Simscape Foundation Library* blocks within the mechanical domain allows one to take friction into account with the *Rotational Friction* block, which considers Stribeck, Coulomb, and viscous friction. The method developed in this thesis, however, requires no manual modification of the *Simscape Multibody* model to add the sophisticated LUGRE friction model (see Chapter 2.2.1) to selected joints, which covers effects like hysteresis, frictional lag, pre-sliding displacement, or stick-slip [Hac15]. This allows for a much more efficient and easier creation of high fidelity models.

The selection and the necessary parameterization regarding friction and backlash is not undertaken in *Simulink* but in the CAD software tool. In order to provide joints with the LUGRE friction model, user-defined parameters are specified in the CAD software. The basic idea is to create a parameter set for a particular case of friction with geometry, material, and lubricant information. This allows the parameter set to be assigned multiple times to different joints, thus proving to be an efficient data entry. A parameterization for friction must contain the parameters listed in Table 3.6, which are defined in the root product. The parameters are grouped by the numeric identifier #. The names of

Table 3.6: *Parameter set for defining friction.*

Parameter Name	Symbol	Unit
`frictionSetting#_Radius`	r	m
`frictionSetting#_Length`	l	m
`frictionSetting#_Gap`	y	m
`frictionSetting#_Eta`	η	$N \cdot s/m^2$
`frictionSetting#_Mu_c`	μ_S	-
`frictionSetting#_Mu_s`	μ_C	-
`frictionSetting#_Sigma_0`	σ_0	
`frictionSetting#_V_s`	v_s	m/s
`frictionSetting#_Mechanism`		
`frictionSetting#_Joints`		

the joints that should receive this friction model are specified, comma-separated, in the parameter `frictionSetting#_Joints`. To ensure uniqueness, the mechanism name must also be specified. If the parameters are available in the CAD system, the model builder method can be called with `CreateSimulinkModel('Friction', true)`. This triggers the processing of the added data and the attachment of friction models. The integration into a revolute joint is achieved by formulating a forward dynamics problem, where the normal forces and the rotational speed is added to the friction model and the calculated torque due to friction is reintroduced to the joint.

For introducing backlash, a similar procedure is required. Like friction, backlash in joints is not a feature of *Simscape Multibody.* Using hard stop blocks of the *Simscape Foundation Library* allows backlash to be simulated but a dynamic altering of the clearance is not possible. For a proper representation of a circular backlash, as it appears in revolute joints, the method developed in this thesis automatically adds the model presented in Subsection 2.2.2 to selected revolute joints. The required parameters are shown in Table 3.7. To trigger the model builder to process the information and to add the backlash model to corresponding joints, the method has to be called with `CreateSimulinkModel('Backlash', true)`. Naturally, both effects can be realized simultaneously when calling the method `CreateSimulinkModel('Friction', true, 'Backlash', true)`. A backlash model is also linked to a revolute joint by formulating a forward dynamics problem. Consequently, the position of the two translational DoFs are used to calculate backlash forces, which keep the DoFs within the specified clearance.

For performance purposes, both the friction model and the backlash model are added in the form of S-Functions.

Table 3.7: *Parameter set for defining friction.*

Parameter Name	Symbol	Unit
`backlashSetting#_Gap`	r	m
`backlashSetting#_K`	k	N/m
`backlashSetting#_D`	d	$N \cdot s/m$
`backlashSetting#_Mechanism`		
`backlashSetting#_Joints`		

Strength Analyses

Buckling is a serious threat to mechanical FCSs and must be avoided in any case. For airplanes certified according to CS-23, requirements on strength are given in CS-23.305, while AC 23-19A ([Fed12]) states that column structures, such as control system pushrods in single-load-path design applications, cannot be allowed to buckle under ultimate load conditions. An examination of the buckling risk is therefore reasonable.

With the functions provided by *Simscape Multibody*, a direct assessment of the buckling risk is not possible. Therefore, the method developed in this thesis determines the critical load according to Euler for user-selected parts. The required Young's modulus is specified in the CAD system either in the form of user-defined parameters (dominant) or via the assigned material. In terms of the second moment of area, it can also either be specified exactly as a parameter (dominant) or very roughly approximated. Using the internal forces of a part allows one to then estimate the risk of buckling for a given scenario during the simulation. Thus, even an optimization might take the risk into account. The internal forces are determined via *Weld* joints, which are added to both ends of the part. This guarantees that internal force caused by the part's dead weight is considered as well, leading to two slightly different signals.

The developed method prepares the buckling analysis for parts with the following parameters:

```
isBucklingRelevant = 1
I                  = 7.653e-9
E                  = 8.8e+10
```

where `I` is the second moment of area, and `E` is Young's modulus. The specification of `I` and `E` is not mandatory, as otherwise the information is gained from material data or is approximated. Calling the model builder via `CreateSimulinkModel('Buckling', true)` undertakes all measures for a buckling analysis.

Examining the risk of buckling using the presented method provides an initial estimate but is by no means accurate enough to substitute a necessary buckling strength study.

Approximation of the Second Moment of Area

In case the second moment of area is not given via a user-defined parameter, it is approximated for rotationally symmetric profiles in order to calculate a critical load. This is based on the available moment of inertia. The second moment of area can be calculated, according to [RS08], as

$$I_x = I_y = \frac{I_p}{2} = \frac{1}{2}\int_A r^2 dA \tag{3.20}$$

where I_p is the polar second moment of area. According to [PK01], the moment of inertia with respect to the z-axis is described as

$$J_{zz} = \int_m r^2 dm \tag{3.21}$$

Assuming a homogeneous density,

$$\rho(z) = const \tag{3.22}$$

the moment of inertia can be formulated as

$$\begin{align} J_{zz} &= \rho \int_V r^2 dV \tag{3.23}\\ &= \rho \cdot l \cdot \int_A r^2 dA \tag{3.24}\\ &= \rho \cdot l \cdot 2 \cdot I_x \tag{3.25} \end{align}$$

where l is the column or part length. Thus, the relation between I_x and J_{zz} is

$$I_x = \frac{J_{zz}}{2 \cdot \rho \cdot l} \tag{3.26}$$

As mentioned, the critical load after EULER as well as this approximation are only applicable to slim and rotational symmetric columns. As such, J_{zz} must always be the moment of inertia with respect to the longitudinal axis of the column. For this reason, the method uses the minimal value of the main diagonal of the inertia tensor for the calculation, to consider the cases where column direction is not the part's z-axis.

Of course, the derivation of the second moment of area from the moment of inertia is only exact for prismatic geometric bodies. Large parts of the rods in mechanical FCSs might be prismatic, but at their ends the shape changes to realize appropriate connection points. Most of the time, this results in more mass away from the longitudinal axis, leading to a larger moment of inertia and thus a larger second moment of area, which is akin to a mean value. However, EULER's formula requires the minimum second moment of area occurring in the parts. The presented approximation consequently represents only emergency values if the exact value is not available.

3.5.5 Modification of CAD Parameters

Typically, there is a large tool-chain gap between the design and the simulation world, preventing automated processes across tool boundaries and resulting in a large amount of manual work. Therefore, one of the main contributions of this thesis is the development of a method for a bidirectional data transfer between CAD software and *Simscape Multibody*. This closes the gap and allows for the application of methods on a model coupling both worlds. The developed method provides *MATLAB* access to the user-defined parameters in the CAD system. The bidirectionality is realized by the ability to modify them from *MATLAB*. Depending on the degree of parameterization, a CAD model can be modified considerably.

For the modification of CAD parameters, the same API and DLL are used. The generalized interface provides a function called `SetParameterValue`, which requires three arguments. Naturally, the parameter name and the value to which it is meant to be set are required. Since user-defined parameters can be an element of either products (root product) or parts and no pointer to the parameter parent object in the CAD software can be passed, a string containing the path of the instance names from the root product to the parameter parent is used. The DLL that has direct access to the objects in the CAD system splits this string and searches for the corresponding parameter.

3.6 Optimization Framework

In order to use the developed method for the easy integration of CAD parameters in *MATLAB* optimizations, a framework is created that integrates and controls the method. The framework is a collection of object-orientated classes and allows for the application of different *MATLAB* optimization algorithms as well as the external software package *Interior Point Optimizer* (IPOPT). Due to its very modular structure, further optimizers can be integrated without much effort. In the following, only the structure and strategies for increasing the efficiency of the framework are presented. The manual for the framework can be found in Appendix A.6.

3.6.1 Structure of an Optimization Problem

The structure of the optimization framework is illustrated in Figure 3.18. The individual classes for structuring an optimization problem are described in the following.

Problem

The main class of the optimization framework is of the type `Problem` and represents the only interface to the user. It has a property to choose an optimizer and contains the `iCADModel` object, which stores all information regarding the CAD model and makes the CAD system available to the optimizer. Furthermore, the `Problem` object holds

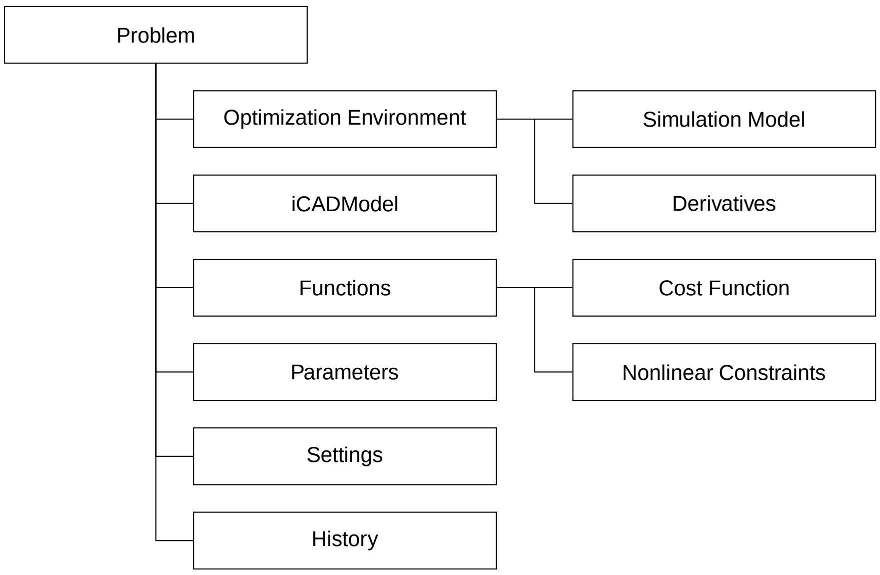


Figure 3.18: *Structure of the optimization framework.*

the optimization parameters, settings, history, and the optimization environment. The optimization parameters are created automatically within the `Problem` object using a function that allows for comfortable parameter searching by the names of the CAD parameter and its parent object. The function also handles the different parameter types (CAD or *MATLAB*).

Functions

The `Functions` property manages function handling. Here, the function handles of the user-defined cost function and the function for evaluating nonlinear constraints are stored.

Parameters

Each optimization parameter is an instance of the `OptimizationParameter` class. Objects of this type holds information such as initial values, upper and lower bounds, spacing for finite differences, scaling factor, parameter type, and the parameter itself. Depending on the type, the set routine either changes the corresponding CAD parameter or modifies the particular workspace variable.

Optimization Environment

The individual optimization algorithms require different function structures. Thus, the optimization environment is specific to the chosen algorithm. There is a particular class for every optimizer that handles the argument and returns the value structure. However, it is always derived from the same base class `OptimizationEnvironment`, which contains all parts not specific to the algorithm. Moreover, the simulation model and the user-defined cost function and nonlinear constraints are part of the optimization environment.

The `Derivatives` object is used for the derivation of their *Jacobian*.

Simulation Model

This class is a sort of a container for the *Simulink* simulation models. It controls the simulation and manages the data logging. Signals that are relevant for the optimization are logged and used for the evaluation of the cost function and nonlinear constraints. Furthermore, methods are implemented that allow for an efficient parameter update and a significant time-saving for specific, cost-intensive simulations.

Derivatives

For gradient-based optimization algorithms, the derivatives like $\frac{\delta J}{\delta \mathbf{z}}$, $\frac{\delta \mathbf{g}}{\delta \mathbf{z}}$, and $\frac{\delta \mathbf{h}}{\delta \mathbf{z}}$ are required. The `Derivatives` class provides the corresponding methods to calculate the gradients on the basis of a prior specified manner. In this thesis, only *forward* and *central* finite differences are applied, since the simulation model is considered a *black box*, where an analytical gradient determination is impossible. Nevertheless, other methods for the gradient derivation can easily be implemented into the framework.

Settings

The optimization algorithms provide a wide variety of settings which may influence the optimization results significantly (e.g. tolerances, maximum number of iterations). In addition, each optimizer has its own settings structure. To take these factors into account, the settings are specified outside the optimization framework but are subsequently passed. This also allows for a problem-specific configuration of the optimization algorithm. Furthermore, the `Settings` property allows the type of the gradient derivation to be specified.

3.6.2 Strategies for an Efficient Optimization

The presented framework is used in the remainder of this thesis for a simulation-based optimization across tool-boundaries, which includes CAD parameters. This is the typical application the framework was developed for.

However, the simulation of physical models can quickly become computationally intensive. Together with a finite-differences-based gradient determination, one single iteration can take a considerable amount of time. Especially for the more accurate central differences, the number of costly evaluations of the cost function relates to the number of optimization parameters according to $2n+1$. For this reason, the following two strategies are implemented to avoid unnecessary processes.

Sparse CAD Parameter Update

The sparse CAD parameter update strategy aims for a minimum costly interaction with the CAD software. After receiving a new set of optimization parameter values, typically all parameters are consequently updated in the *MATLAB* workspace and the CAD software. The update of CAD parameters and, in particular, the subsequent necessary update of the changed CAD model data is very time-intensive, depending on the complexity of the model. For this reason, a sparse update method is implemented to minimize the effort related to CAD parameter updates. In this method, CAD parameters are only updated if their value differs from the current one. Consequently, if no parameter change was sent, the CAD model remains unchanged, and thus there is no update of the model data required.

This eliminates costly but unnecessary interaction with the CAD software, especially in a simultaneous optimization of hard- and software parameters, which also includes non-CAD parameters. In combination with a finite differences gradient determination, the time savings become substantial.

Sparse Simulation

While the previous strategy focused on a minimum interaction with the CAD software, this strategy aims to avoid unnecessary simulations. There are optimization problem formulations like TSCHEBYSCHEFF that introduce additional parameters and reformulate the problem in order to achieve better convergences. Such a reformulation is done in the application example in Section 5.7. However, this reformulation leads to redundant, costly simulations, as the additional parameter has no effect on the simulation results themselves but only on the evaluation of the cost function and the nonlinear constraints. Therefore, the optimization parameters provide an attribute that indicates whether it has an influence on the simulation results. Consequently, a subset of simulation influencing parameters arises. As soon as there are already results available, which were obtained with the same parameter set, the simulation is skipped. Analogous to the aforementioned strategy, this significantly reduces the duration of finite-differences-based derivative approximation since no additional, costly simulation is required for such parameters.

4

Application to a Mechanical Flight Control System

This chapter describes the application of the developed interface in the course of a research project together with an aircraft manufacturer. The interface was used for the analysis and modification of a mechanical FCS of a general aviation aircraft in order to equip it with a new full envelope AP system (hardware and software), accordingly an *automatic flight control system* (AFCS). The analyses are necessary to determine requirements on the actuators focusing in particular on torque and dynamics. Furthermore, they allow one to gain information on the kinematic behavior of the FCS, such as the transfer function between actuator and surface, which is relevant for tuning the controller at early development stages. Although the procedures are analog for all three axes, the elevator axis is of major importance, therefore this chapter is limited to this axis.

Initially, a CAD model of the elevator axis of the given mechanical FCS is created and converted to a *Simscape Multibody* model. This axis is analyzed and optimized from a kinematic perspective before mechanical analyses are performed to determine minimum requirements on the actuator forces and dynamics.

4.1 Mechanism and 3D CAD Model

The CAD model of the elevator axis is built in *CATIA V5*. The parts are linked using predefined joints from the *DMU Kinematics* environment, which automatically creates equivalent constraints between the parts for each joint. These joint objects are used by the interface when generating the *Simscape Multibody* model.

The elevator axis consists of several push-pull rods, which are linked via levers and which connect the pilot's stick with the elevator surface. Due to lack of space, there is no degree of freedom in terms of the installation position of the new AP actuator for the elevator axis. An overview of the entire axis is illustrated in Figure 4.1.

The integration of the actuator in the already installed FCS is achieved with a modified

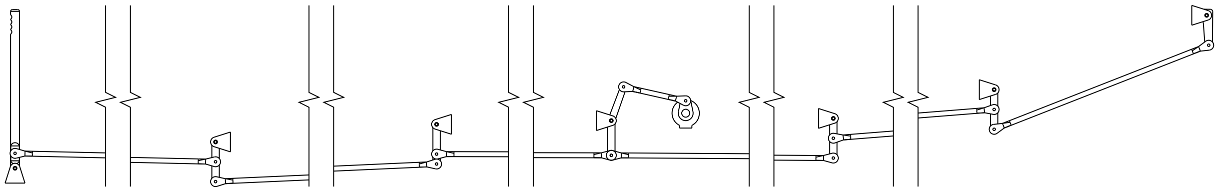


Figure 4.1: *Overview of the elevator axis.*

lever, which is connected to the lever of the actuator via a rod (see Figure 4.2). In this way, the existing certified mechanical FCS remains unchanged, making it much easier to obtain a permit to fly for flight tests. For optimization purposes, the orientation of the actuator, as well as the linkage to the existing elevator axis, was parameterized. This allows the parameters to be modified from *MATLAB*, which is necessary to apply *MATLAB* optimization algorithms. The parameters dX and dZ essentially describe the position of the upper joint, while L specifies the length of the actuator lever and β represents the actuator orientation.

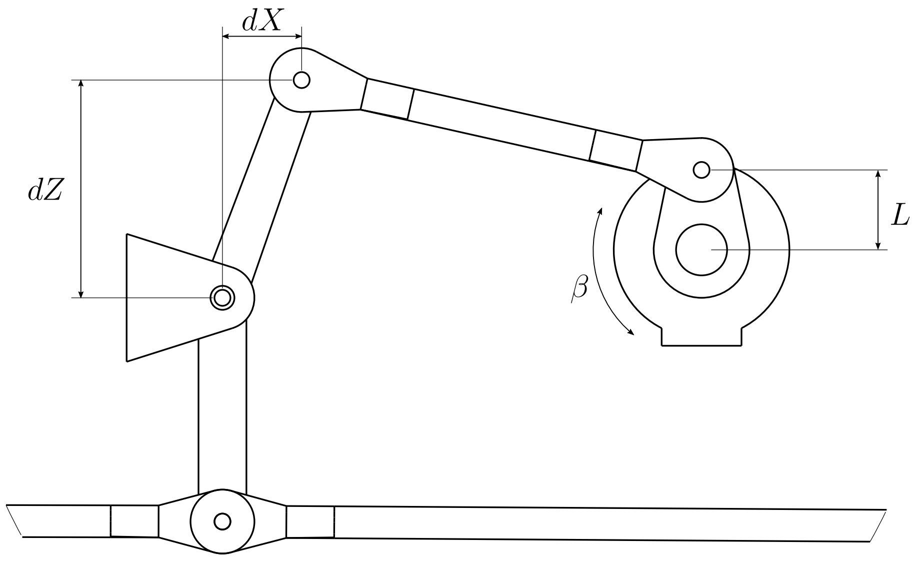


Figure 4.2: *Integration of the actuator in the existing mechanism.*

The length of the rod connecting the actuator lever with the modified lever of the existing mechanism is parameterized in such a way that the required length is measured automatically in *CATIA*. Table 4.1 lists the initial design parameterization of the integration.

4.2 Kinematic Analyses

First, the given mechanical FCS shall be analyzed in terms of kinematics. This means forces and the masses of the components are not considered in the analyses, but the mechanism's motion and its range of motion are reviewed. The kinematic analyses are

Table 4.1: *Initial parameterization of the integration.*

Parameter	Value
dX	$50mm$
dZ	$136mm$
L	$50mm$
β	$0°$

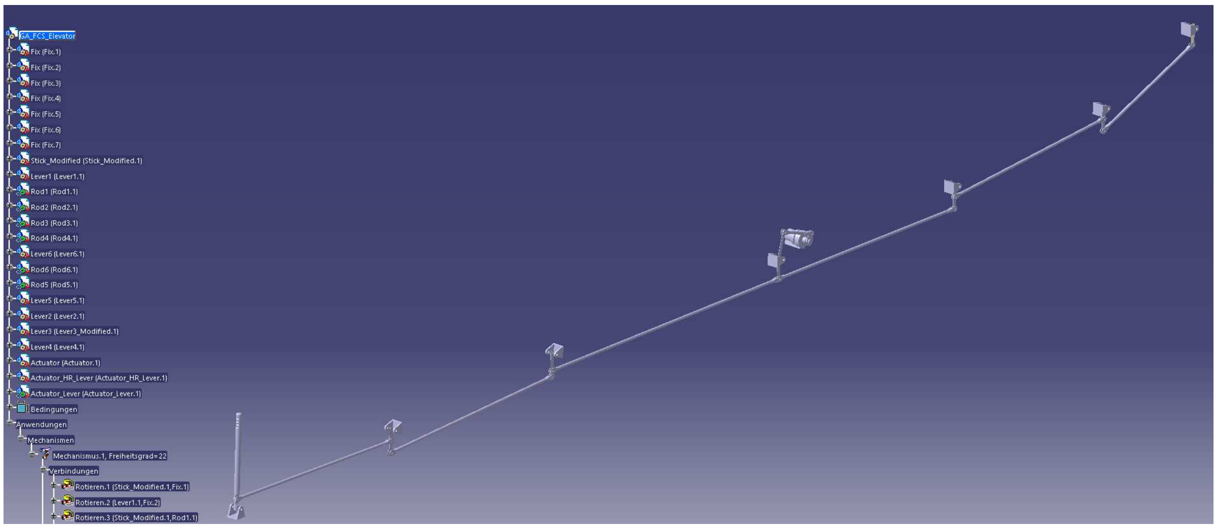


Figure 4.3: *CAD model of the elevator axis.*

performed with the help of an MBS model and are intended to specifically answer the following questions:

- What actuator deflections are necessary to achieve full elevator surface deflections?
- To what degree are nonlinearities observed, where the result is a variation in the gear ratio between actuator motion and control surface deflection over the entire stroke?

For this purpose, the CAD model of the mechanical FCS must be converted to a MBS model. This is accomplished via the method presented in Chapter 3. It allows for the automatic translation of the CAD model to a parameterized *Simscape Multibody* model within a fraction of the time that is needed for a manual transfer. Figure 4.4 shows an excerpt of the model that was created automatically by the interface. Parts and joints defined in the CAD model in the previous section can now be found in the MBS model.

In order to enable a visual assessment of the simulation and its results, the generation of *STL* files representing the parts is also triggered by the interface. This provides a 3D representation in the *Mechanics Explorer*, as the interface automatically assigns the files to the corresponding parts. The resulting visualization of the FCS's elevator axis in the *Mechanics Explorer* is shown in Figure 4.5.

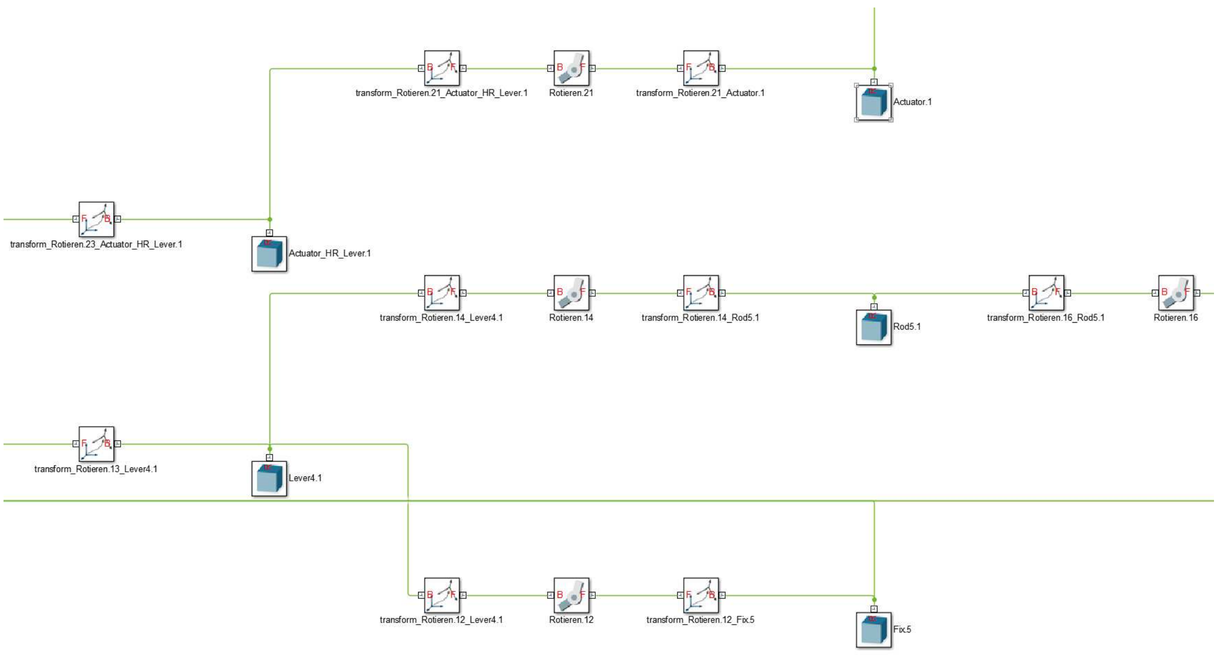

transform_Rotieren.21_Actuator_HR_Lever.1
Rotieren.21
transform_Rotieren.21_Actuator.1
Actuator.1
transform_Rotieren.23_Actuator_HR_Lever.1
Actuator_HR_Lever.1
transform_Rotieren.14_Lever4.1
Rotieren.14
transform_Rotieren.14_Rod5.1
Rod5.1
transform_Rotieren.16_Rod5.1
Rotieren.16
transform_Rotieren.13_Lever4.1
Lever4.1
transform_Rotieren.12_Lever4.1
Rotieren.12
transform_Rotieren.12_Fix.5
Fix.5

Figure 4.4: *Detailed view of the FCS elevator axis Simscape Multibody model.*

Before the actuator deflections and the variation in the gear ratio can be determined, the *Simscape Multibody* model has to be prepared. Therefore, methods of inverse dynamics are applied by actuating the joint of the elevator surface according to a predefined motion. Sensing the position of the actuator joint then provides the necessary data to determine the maximum deflections as well as to analyze nonlinearities.

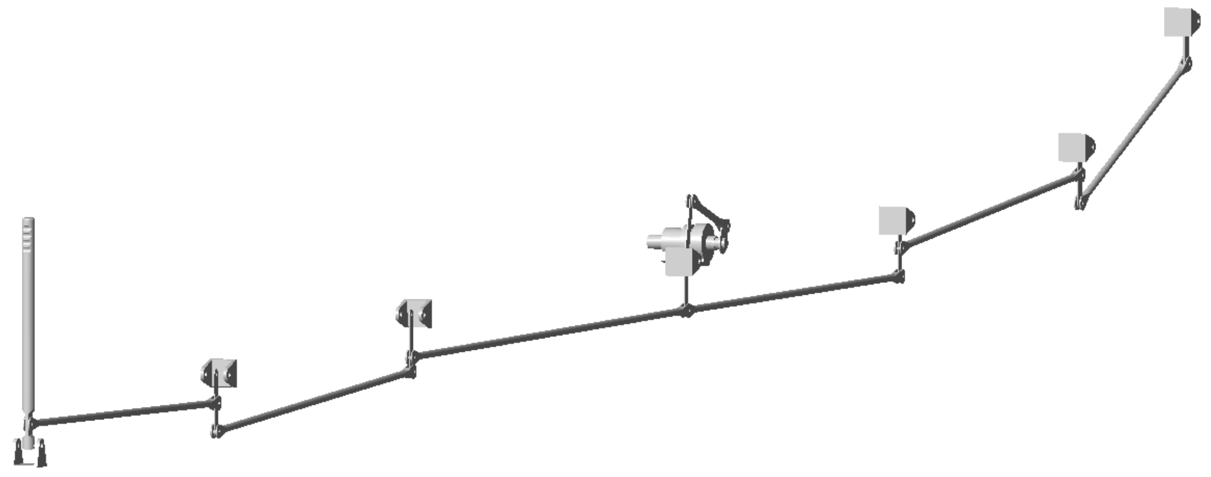


Figure 4.5: *3D representation of the MBS model visualized with the Mechanics Explorer.*

4.2.1 Preparation of the MBS Model

The motion of the elevator surface joint shall start at the upper hard stop ($\eta_{min} = -15°$) and move with a constant rate of $\dot{\eta} = \frac{3.5°}{1s}$ to the lower hard stop ($\eta_{max} = 20°$). In order to be able to specify the joint's motion, the *Motion Actuation* setting must be set to *Provided* *by Input* and the *Torque Actuation* setting has to be set to *Automatically Computed.* The additional input port of the joint block then allows the introduction of a *Physical Signal.* This signal must contain information on the position, velocity, and acceleration of the specified motion. It is generated using regular *Simulink* blocks and then converted to a *Physical Signal* via the *Simulink-PS Converter* block. The implementation of the motion signal for the kinematic analyses, which is introduced in the elevator surface joint, is shown in Figure 4.6. The initial condition of the integrator block is set to η_{min}. For evaluation purposes, the position of this joint is also measured and saved within a variable.

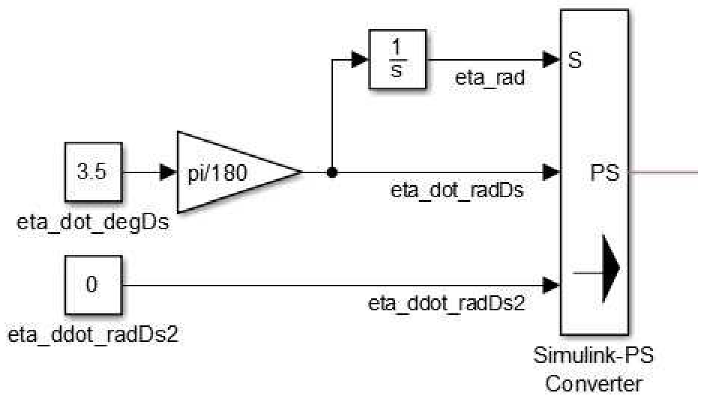


Figure 4.6: *Generation of the motion signal for the revolute joint of the elevator surface.*

The revolute joint between the actuator lever and the actuator itself provides the signal for the position measurement α_e of the actuator. Hence, position sensing is activated in the joint settings and the signal is recorded for further processing.

4.2.2 Evaluation of Simulation Results

After running the simulation, the recorded position signals of both joints can be evaluated regarding the questions introduced at the start of this section.

Maximum Actuator Deflections

The maximum actuator deflections are dependent on the maximum elevator deflections, the nonlinear gear ratio of actuator deflection to elevator deflection, and the neutral actuator position. The resulting actuator position over time, which is caused by the elevator deflection, is shown in Figure 4.7. The graph reveals a maximum deflection $\alpha_{e,max} = 55.0°$, which corresponds to the η_{min}. Likewise, the minimum actuator deflection $\alpha_{e,min} = -51.8°$ corresponds to the maximum elevator deflection η_{max}. The total actuator deflection sums up to $\Delta\alpha_e = 106.8°$.

Evaluation of Nonlinearities

Nonlinearities in the mechanism's motion can already be observed in Figure 4.7. The line representing the deflection of the actuator shows greater changes in the curvature,

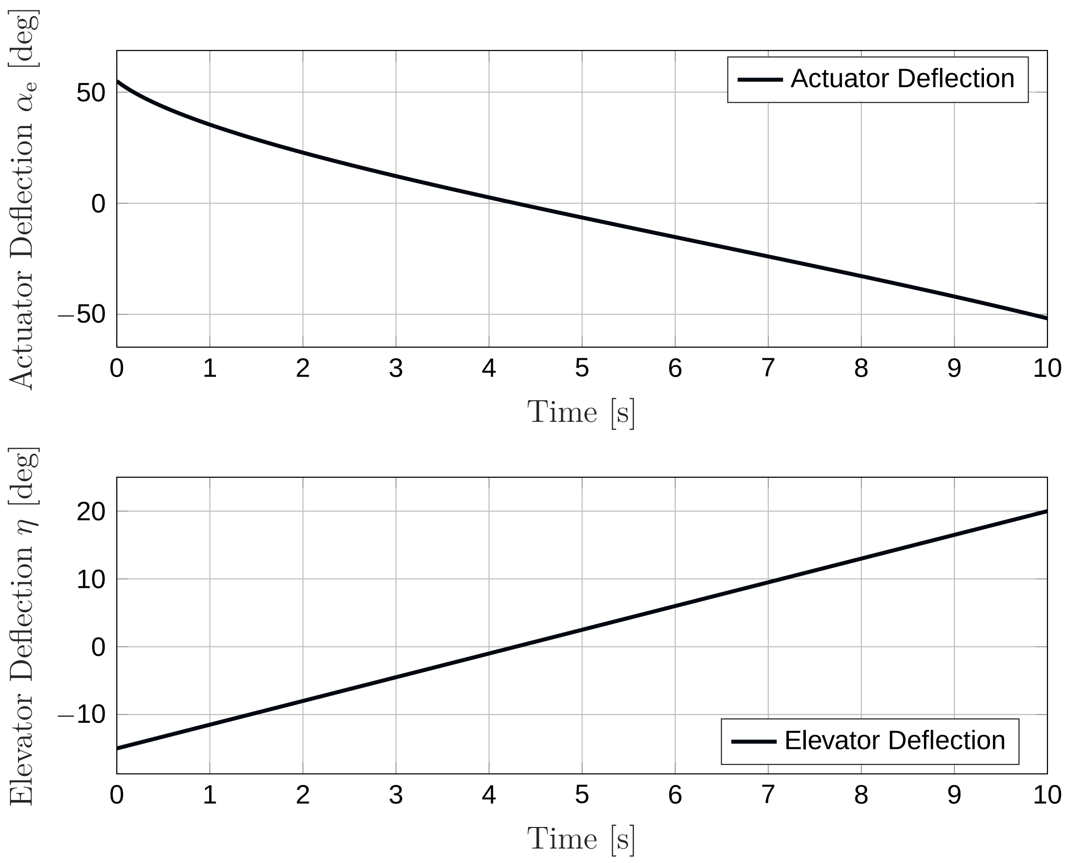


Figure 4.7: *Elevator and actuator deflections over time.*

especially at the beginning and at the end of the simulation. In these sections, the actuator and the elevator surfaces are near their full deflections. Figure 4.8 illustrates the elevator deflection plotted versus the actuator deflection. Furthermore, the ideal linear relationship is drawn to highlight the deviation due to nonlinearities. For the actuator deflection range $-10° \leq \alpha_e \leq 40°$ the effects might be negligible, but for larger deflections outside this range, they quickly become significant.

Nonlinearity becomes even more visible if the gear ratio is plotted versus the actuator deflection. For each time step i, the gear ratio can be calculated as

$$G_i = \frac{\Delta\alpha_{ei}}{\Delta\eta_i}. \tag{4.1}$$

Figure 4.9 shows the change of the gear ratio versus the actuator position. As already explained in Chapter 2.1.1, it can be seen that the gear ratio is growing at an accelerated rate with increasing actuator deflections. At the maximum actuator deflection, G grows to almost four times its value, from $G = 2.49$ to $G = 9.09$. It is obvious that such significant changes affect flight control algorithms and therefore must be carefully considered.

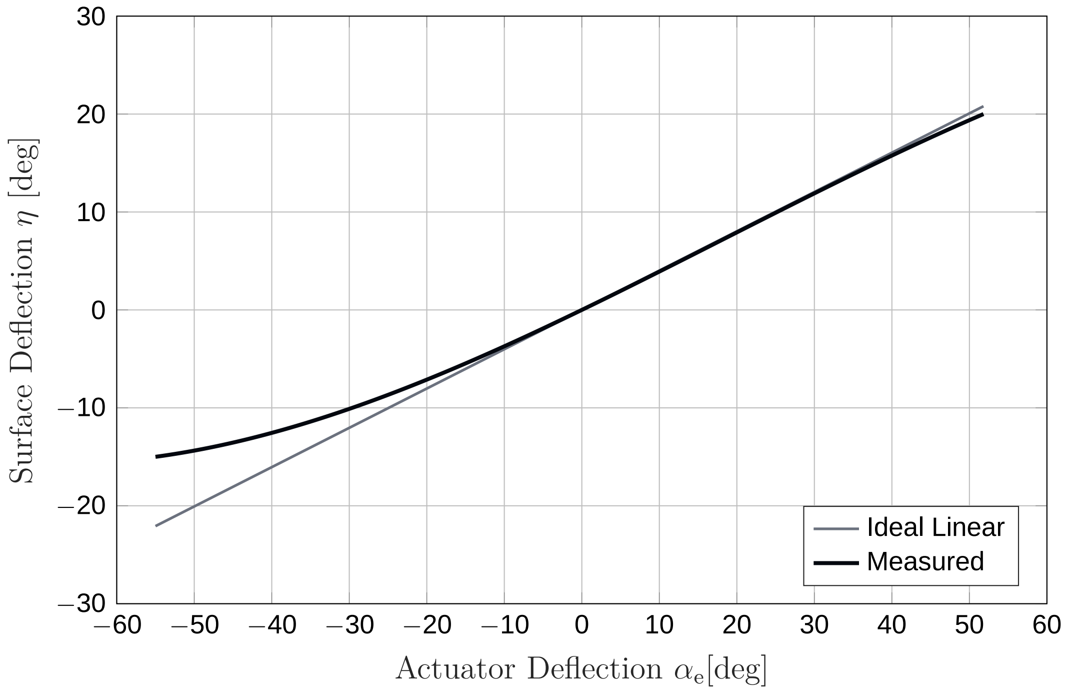


Figure 4.8: *Actuator position over time.*

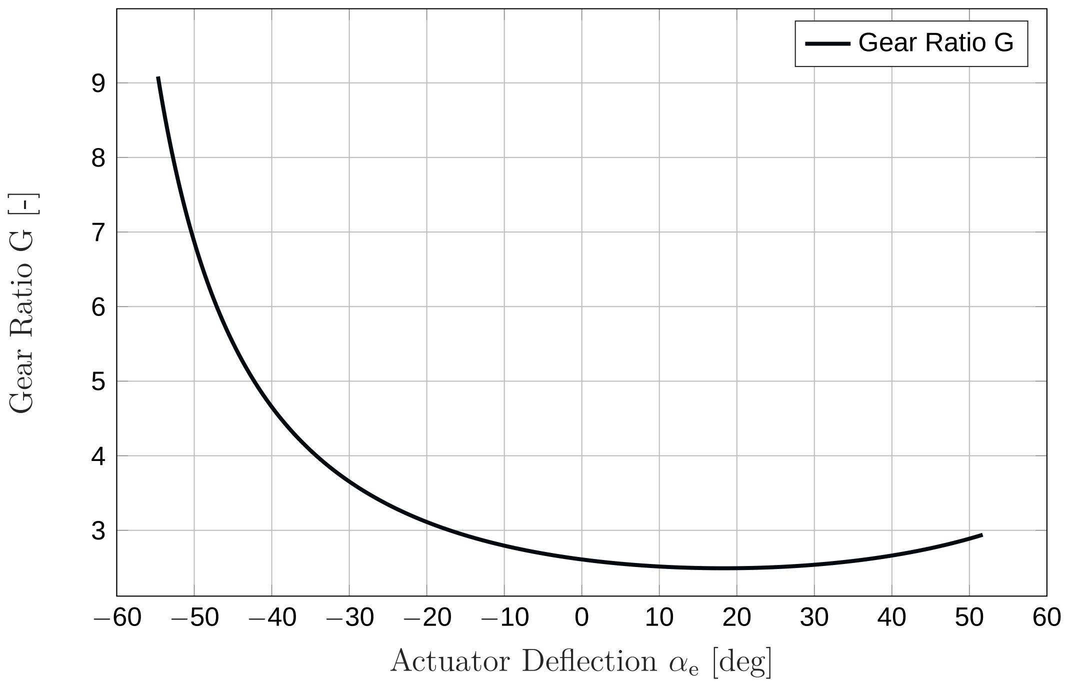


Figure 4.9: *Actuator position over time.*

Evaluation of the Kneeling Risk

As mentioned in Chapter 2.1.1, a kneeling state of the FCS can be a catastrophic failure and must be avoided at all times. For this reason, the risk of kneeling is assessed for the integration mechanism of the actuator by evaluating of the angles δ_1 and δ_2 (see Figure 4.10).

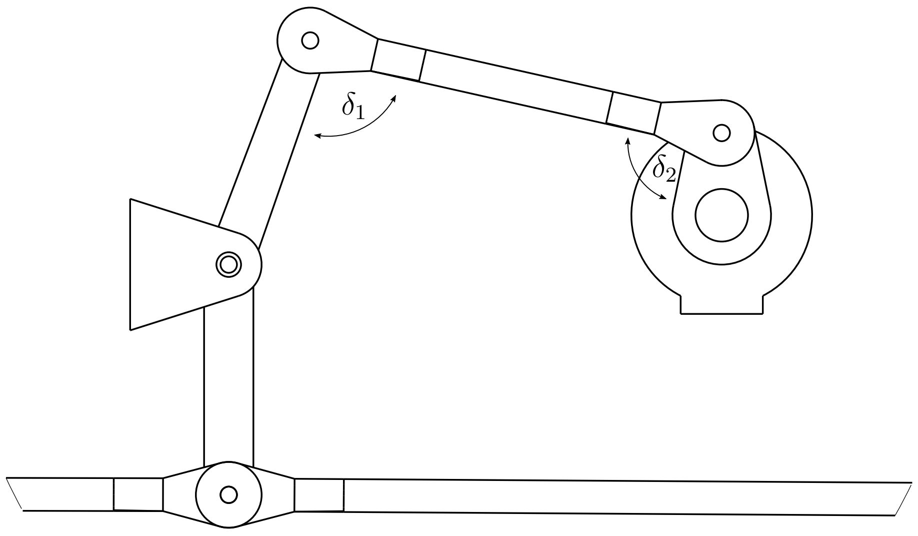


Figure 4.10: *Definition of the angles δ_1 and δ_2.*

The minimum distance of both angles to either 0° or 180° is measured and plotted in Figure 4.11. Naturally, the maximum distance from kneeling is 90°. Conversely, as soon as it is 0°, the mechanism is in a kneeling state. The evaluation of the simulation results reveals that δ_2 is only 14.3° away from a kneeling position when reaching η_{min}.

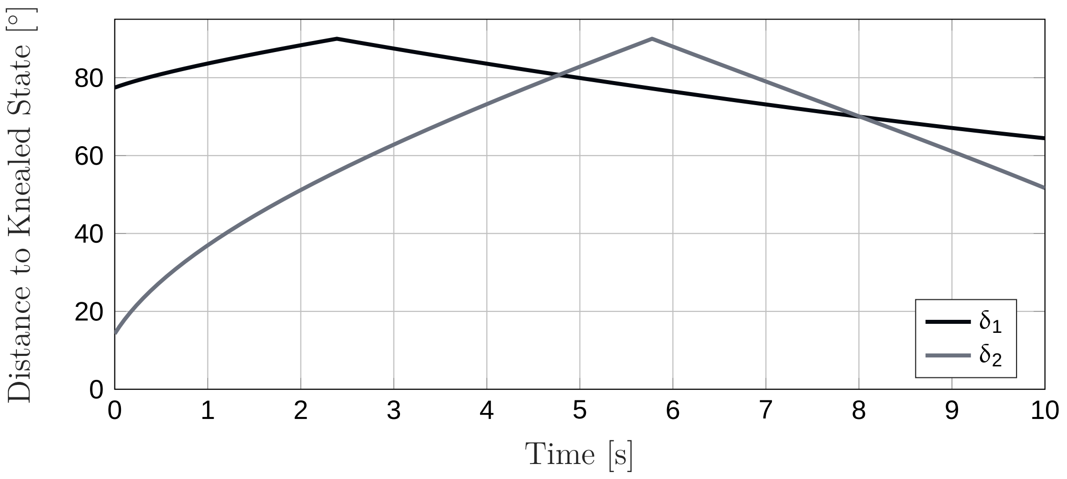


Figure 4.11: *Minimum distance of the angles δ_1 and δ_2 to a kneeling state.*

4.2.3 Optimization of CAD Parameters

The simulation results of the initial design with very asymmetric plots show a potential for optimization. The integrated optimization framework of the interface enables an optimization of the CAD parameters specified in Section 4.1 using built-in *MATLAB* algorithms, for example. This functionality is used to find a better design of the integration mechanism in terms of kinematic factors. The original design is used as an initial solution and is modified throughout the optimization process.

Constrained Optimization Problem Setup

For the optimization of the design parameters, *MATLAB*'s *fmincon* function is used, which finds a minimum of a constrained nonlinear multivariable function [The16b]. Therefore, a cost function $J_{Cost}(\mathbf{z})$, inequality constraints $\mathbf{g}(\mathbf{z})$, and equality constraints $\mathbf{h}(\mathbf{z})$ are defined. The constrained optimization problem can be formulated as

$$\min_{z\in\mathbb{R}^n} J_{Cost}(\mathbf{z}) \tag{4.2}$$

$$\text{s.t. } \mathbf{h}(\mathbf{z}) = 0 \tag{4.3}$$

$$\mathbf{g}(\mathbf{z}) \leq 0 \tag{4.4}$$

The *fmincon* function belongs to the gradient methods, where the gradient of the cost function at the current point determines the subsequent search direction. Naturally, this means that only local minima can be found with this method, which is typical for gradient methods. Since the gradients are not calculated analytically, they are determined via finite difference quotients. The gradient for each parameter is approximated via the central difference quotient as

$$D^0[J_{Cost}](z) = \frac{J_{Cost}(z+h) - J_{Cost}(z-h)}{2h} \tag{4.5}$$

where h is a constant spacing set to 10^{-2}. Significantly smaller values are neglected by *CATIA*.

Further, the chosen optimization algorithm allows setting upper and lower bounds of optimization parameters so that the solution is always in the range $lb \leq z \leq ub$ [The16b]. Table 4.2 lists them together with the initial values.

Table 4.2: *Optimization parameters, initial values, and limits.*

Parameter	Initial Value	Lower Bound	Upper Bound
dX	$50mm$	$30mm$	$200mm$
dZ	$136mm$	$30mm$	$200mm$
L	$50mm$	$45mm$	$100mm$
β	$0°$	$-15°$	$30°$

Cost Function

The primary goal of the optimization procedure is to find a CAD parameterization with fewer nonlinearities and a more uniform gear ratio over the actuator deflection range. Therefore, the cost function to be minimized must consider the gear ratio and is calculated by

$$\begin{aligned} J_{Cost} = \sum_{i=2}^{N-1} (G_i - G^*) &= \sum_{i=2}^{N-1} \left(\frac{\Delta\alpha_{ei}}{\Delta\eta_i} - G^* \right) \\ &= \sum_{i=2}^{N-1} \left(\frac{\alpha_{ei+1} - \alpha_{ei-1}}{\eta_{i+1} - \eta_{i-1}} - G^* \right) \end{aligned} \tag{4.6}$$

with

$$G^* = \min_i G_i \tag{4.7}$$

the minimum gear ratio of the current design, G_i the local gear ratio (based on central difference quotient), and N the total number of simulation time steps, which accordingly has an influence on the value of the cost function. For this reason, the number of time steps is made to be constant by using a fixed-step solver. With this formulation of costs, the optimization algorithm attempts to minimize the area between the curve and the curve's minimum, as shown in Figure 4.9. This can be achieved by flatten the curve towards its minimum while maintaining the minimum gear ratio, which results in the desired reduction of nonlinearities.

Nonlinear Constraints

The nonlinear constraints are formulated in form of equality and inequality constraints and are used to ensure that

- the minimal gear ratio of the optimized mechanism is not lower than that of the initial design and
- the solution is kinematically feasible.

As mentioned in the previous section, the cost function formulated in Equation 4.6 can be minimized by decreasing the variation of the gear ratio. In order to obtain comparable designs, it is desirable to keep the minimum gear ratio. As a means to maintain the minimum gear ratio during the optimization, the inequality constraint is formulated as

$$g(\mathbf{z}) = 2.49 - G^* \tag{4.8}$$

where G^* is the minimal gear ratio of the current design according to Equation 4.7, and 2.49 is the minimal gear ratio of the initial design.

To avoid the translation of the mechanism's geometry into mathematical constraints, other means are required to ensure the kinematic feasibility of the parameter set. The CAD model is parameterized in such a way that the initial state of the elevator axis is always feasible, but during the simulation, infeasible states may be reached. In these cases, *Simulink* stops the simulation, because a geometric constraint cannot be maintained any longer. This is exactly where the kinematic feasibility can be verified, namely by inspecting the final state of the elevator joint. The resulting formulation of the equality constraint is

$$h(\mathbf{z}) = \eta_{max} - \eta_N \tag{4.9}$$

As long as the elevator joint position in the last time step of the simulation η_N reaches the design maximum deflection η_{max}, the mechanism is kinematically feasible over the entire movement.

Results

The results of the optimization process are presented below. The optimized parameter set is listed in Table 4.3. It can be seen that the two parameters describing the length and angle of the modified lever dX and dZ ran into the corresponding lower and upper bound. This indicates that there is probably a better solution than the one found outside the search space. However, due to limited installation space and manufacturing, these limits should be maintained.

Table 4.3: *Comparison of initial and optimized parameters.*

Parameter	Initial Value	Lower Bound	Upper Bound	Optimized Value
dX	$50mm$	$30mm$	$200mm$	$30mm$
dZ	$136mm$	$30mm$	$200mm$	$200mm$
L	$50mm$	$45mm$	$100mm$	$70.8mm$
β	$0°$	$-15°$	$30°$	$13.8°$

The new actuator orientation is slightly rotated and the actuator lever is lengthened. The design with the optimized parameter values is illustrated in Figure 4.12 next to the initial design in gray. Both versions are shown with the elevator surface in neutral position.

The upper curve in Figure 4.13 represents the actuator deflection over time. With a maximum deflection $\alpha_{e,max} = 41.2°$ and a minimum deflection $\alpha_{e,min} = -54.8°$, the total actuator deflection range of $\Delta\alpha_e = 96°$ is reduced compared to the initial design.

With regards to nonlinearities, it can be seen in Figure 4.14 that the deviation from the ideal linear course is much smaller and also more symmetric than in Figure 4.8. The reduction of nonlinearities is also clearly visible in Figure 4.15, which shows the gear ratio

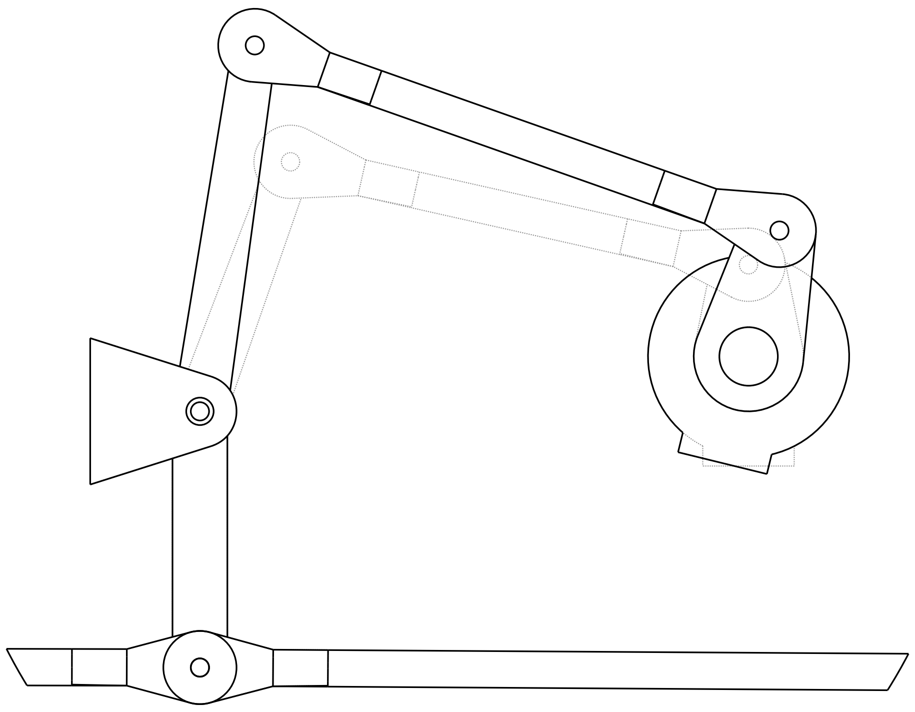


Figure 4.12: *Optimized integration of the actuator (gray: initial design).*

versus actuator deflection of the optimized and initial designs. Note that the minimum gear ratio has not been changed but only occurs in a different actuator position.

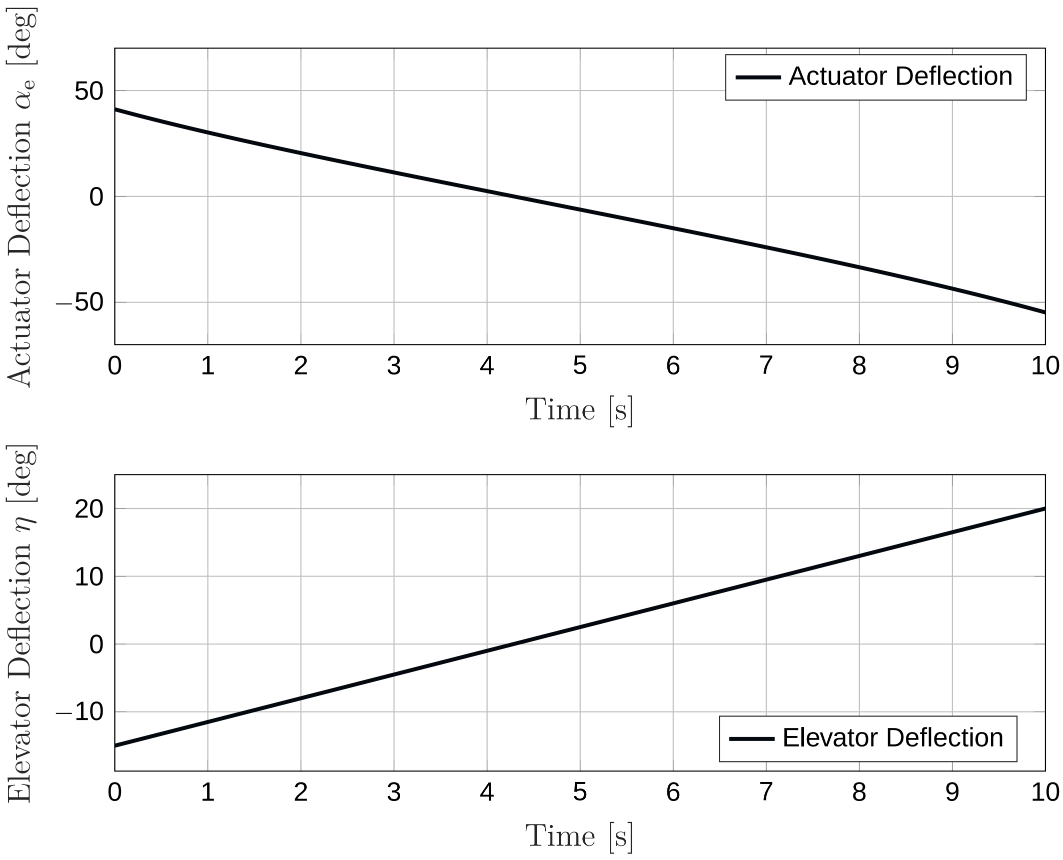


Figure 4.13: *Elevator and actuator deflections over time.*

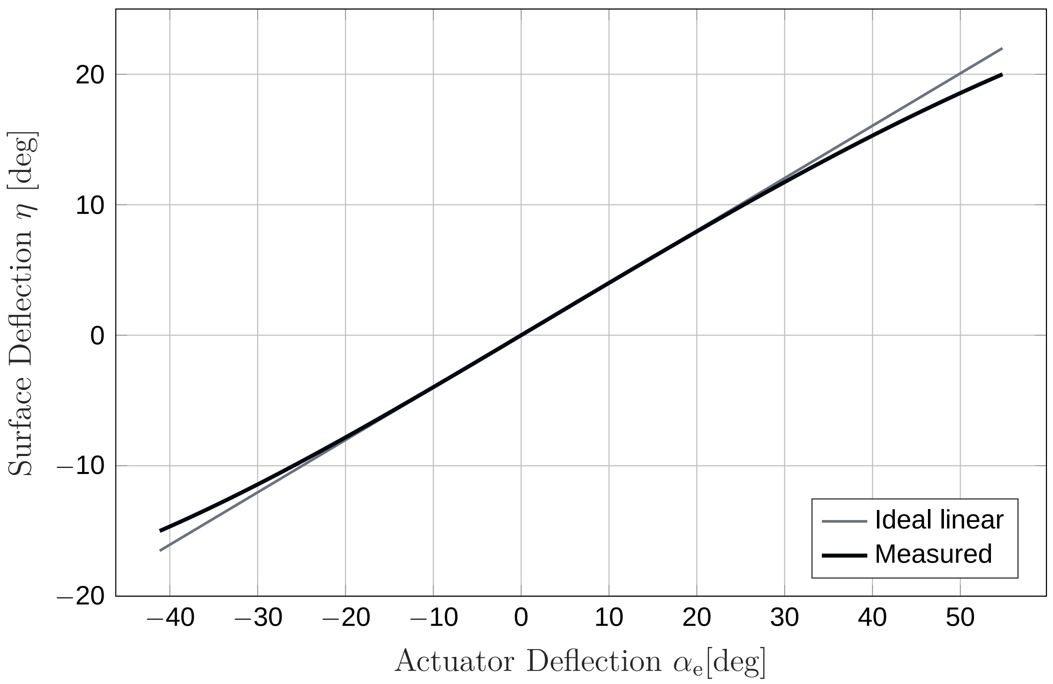


Figure 4.14: *Actuator position over time.*

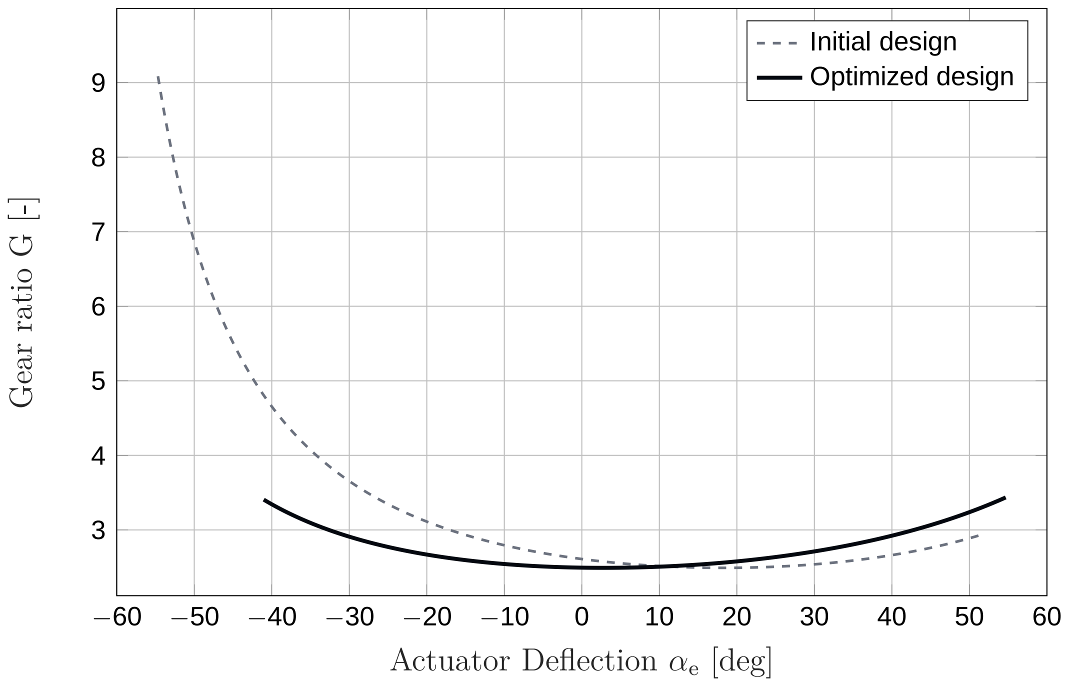


Figure 4.15: *Actuator position over time.*

The data from Table 4.4 allows for a quantification of the optimization results. It lists the costs associated with both designs. The costs of the initial design, the area between the curve and its minimum, could be reduced by 55.23%. The maximum gear ratio has been reduced by 62.16%.

Table 4.4: *Comparison of the simulation results of the initial and optimized designs.*

Parameter	Initial Design	Optimized Design
Costs	$5.569 \cdot 10^2$	$2.493 \cdot 10^2$
G_{min}	2.49	2.49
G_{max}	9.09	3.44

An analysis of the optimized design regarding the minimum distance to a kneeling state shows that the risk of kneeling decreased significantly compared to the initial design. With 33.4° δ_2 is now more than 19° further away from a kneeling position when reaching η_{min}.

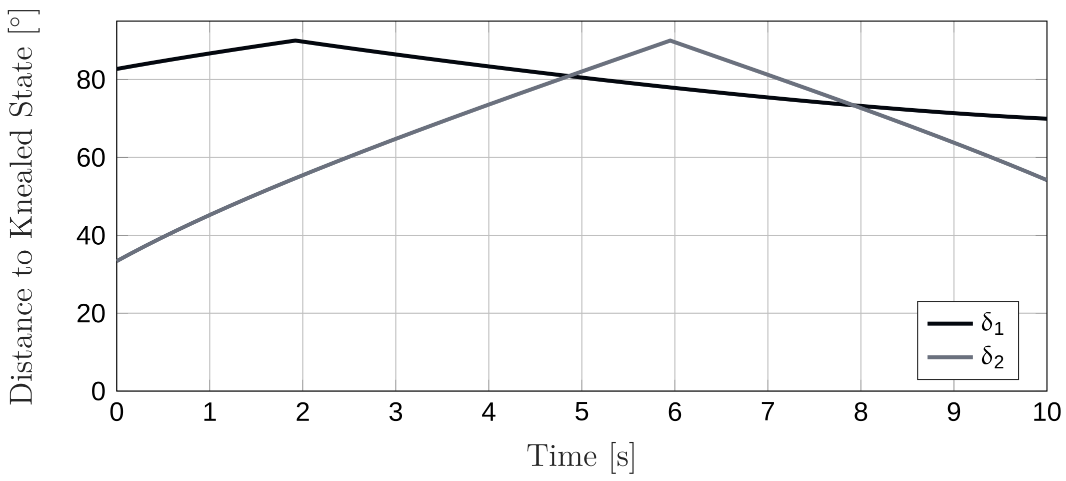


Figure 4.16: *Minimum distance of the angles δ_1 and δ_2 to a kneeling state.*

4.3 Mechanical Analyses

The following mechanical analyses simulate the elevator axis of the FCS with a higher degree of detail, aiming for the forecast of the actuator loads and the actuator performance. By evaluating the resulting flight performance using dynamic simulations, the risk of inaccurate dimensioning shall be reduced in the project. The actuator that is supposed to be integrated is almost identical to a model used in previous projects. Masses and inertias, friction in joints, aerodynamic forces or hinge moments, and actuator characteristics now play an important role. In the following, the optimized design of the actuator integration into the elevator axis is used. After the preparation of the CAD model and the MBS model, the simulation is performed and its results are evaluated and discussed.

4.3.1 Preparation of the Models

Before the simulations can be performed, the CAD model and the MBS model have to be adapted and enhanced. The information about the friction in joints is added to the model in *CATIA* while the actuator model and aerodynamic forces on the elevator surface are introduced to the *Simscape Multibody* model. Further, the actuator command signal is defined.

Inclusion of Friction

Since the *Simscape Multibody* toolbox does not provide ready-to-use friction models for its joint blocks, the LUGRE model (presented in Chapter 2.2.1) was implemented and integrated into the interface. It automatically adds the friction model to the corresponding joint blocks in the model. Thereby, the CAD model serves as a source for the characteristic parameters that significantly determine the joint friction and are therefore also required by the friction model. The values in the CAD software are stored in the form of user-defined parameters and are evaluated only in *MATLAB*. The following listing shows the parameters chosen for the study.

```
frictionSetting1_Mechanism = Mechanismus.1
frictionSetting1_Joints    = Rotieren.2, Rotieren.3, Rotieren.4,
   Rotieren.5, Rotieren.6, Rotieren.7, Rotieren.8, Rotieren.9,
   Rotieren.10, Rotieren.11, Rotieren.12, Rotieren.13, Rotieren.14,
   Rotieren.16, Rotieren.17, Rotieren.19, Rotieren.20, Rotieren.22,
   Rotieren.23
frictionSetting1_Length    = 0.01
frictionSetting1_Radius    = 7.5e-3
frictionSetting1_Gap       = 1e-3
frictionSetting1_Mu_s      = 0.7
frictionSetting1_Mu_c      = 0.6
frictionSetting1_Eta       = 100
frictionSetting1_V_s       = 10e-2
```

To avoid unnecessarily increasing the calculation effort of the simulations, the pin of the revolute joints is not represented in the models.

The coefficients of friction are of course strongly dependent on the materials and their processing, and the dynamic viscosity is very specific to the lubricant used. Since this information is not available and the parameters were not obtained in the course of measurements, a set of worst-case generic values in the order of magnitude of lubricated metals are chosen.

Integration of an Actuator Model

The actuator that shall be integrated is the same for all three axes and is also used in previous similar projects. The model is presented in Chapter 2.4.1. The operating voltage of the actuator as well as the gear ratio of the gearbox are chosen based on the resulting performance in the dynamic simulations. The actuator-specific parameters required by the actuator model can be found in Table 4.5.

Table 4.5: *Parameters of the integrated actuator.*

Name	Symbol	Value
Phase-to-phase inductance	L	$0.55mH$
Phase-to-phase resistance	R	1.25Ω
Voltage constant	k_e	$0.055Vs$
Torque constant	k_t	$0.055Nm/A$
Friction coefficient	k_f	$4 \cdot 10^{-7}Nms$
Moment of inertia	J	$7.5 \cdot 10^{-6}kgm^2$

The actuator model is formulated as a forward dynamics problem, which means the actuator load torque must originate from the corresponding joint block and is an input to the model, while the resulting motion is an output and has to be fed to the joint block. Conversely, the joint block itself represents an inverse dynamics problem.

With the controller implemented according to Subsection 2.4.3, the controlled variable in the following simulations is the actuator position after the gearbox.

Aerodynamic Forces / Hinge Moments

In order to simulate adequate actuator loads, the aerodynamic forces on the control surfaces must be considered. Originating from the dynamic pressure, these forces create moments about the hinge lines of the control surfaces. These hinge moments shall be introduced in the MBS model. Since the aircraft had no flight test instrumentation at the time of the project start, data from earlier flight tests for the design of the previous AP served as a data source. In that case, a *Static Longitudinal Stability* diagram contained relevant information on the stick force and corresponding elevator deflection for a certain flight state and aircraft configuration. The information on the basis of which the aerodynamic loads are calculated is shown in Figure 4.17. The first order polynomial fit, the stick force as a function of the elevator deflection, is described as

$$F_S(\eta) = 10.24 \cdot \eta - 91.8 \tag{4.10}$$

An implementation of this formula in the MBS model allows the stick force to be calculated and applied to the control inceptor related joint.

Actuator Deflection

The diagram in Figure 4.17 contains information about the stick force and corresponding elevator deflection. Together with an *Elevator Control Force in Maneuvers* diagram with stick force and corresponding acceleration data (for the same configuration and altitude),

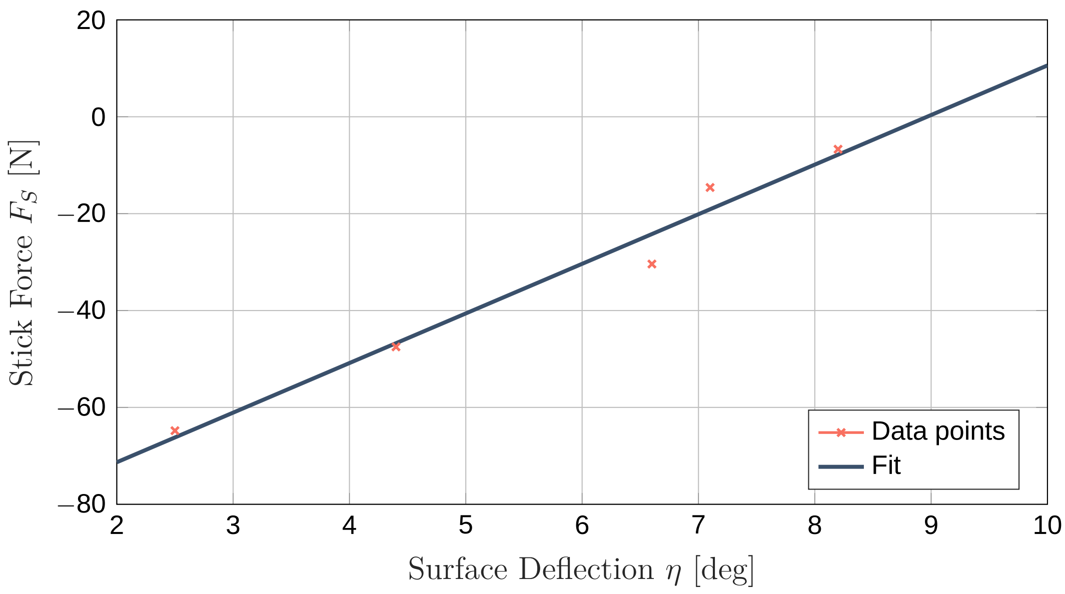


Figure 4.17: *Stick force vs. elevator surface deflection diagram.*

an estimation of the AFCS or aircraft performance can be made. Regarding the dynamic analysis, the performance shall be evaluated based on the maximum achievable load factor and the duration until 1.5g are reached. Therefore, the actuator is commanded a multi-step signal starting from the trim position to the position at which a load factor of $n_z = 1.5g$ is reached. Then the position command is set back to the trim position and finally to the maximum deflection. Each position shall be held for two seconds. The elevator trim deflection can be derived from Equation 4.10 with $\eta_{trim} = 8.96°$, which corresponds to an actuator trim deflection of $\alpha_{e,trim} = -22.63°$.

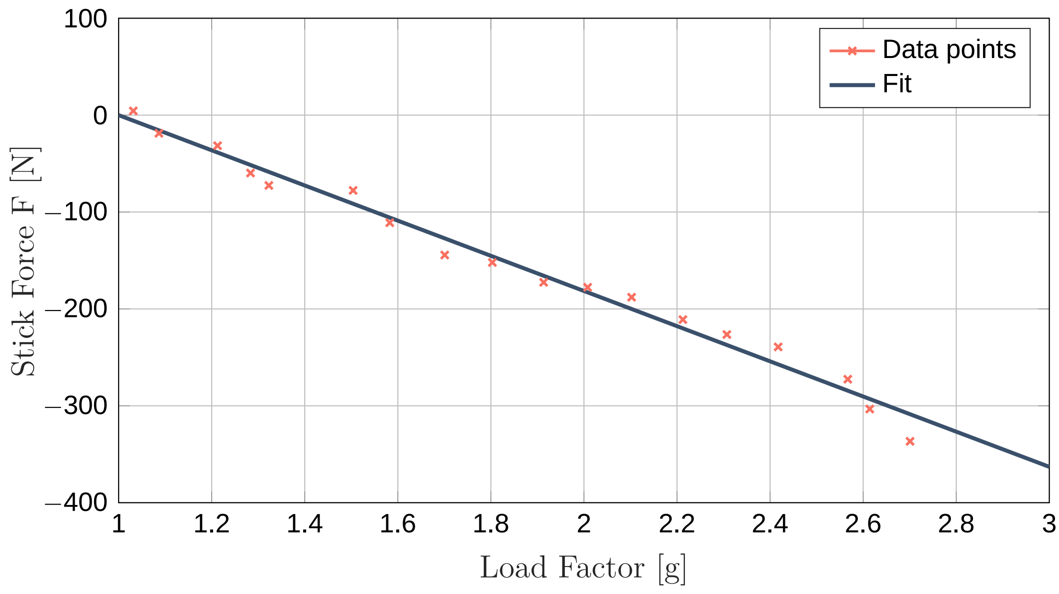


Figure 4.18: *Elevator control force in maneuvers diagram.*

As previously mentioned, the *Elevator Control Force in Maneuvers* diagram allows one to obtain the stick force and consequently $\eta_{1.5}$ the elevator deflection for $n_z = 1.5g$. The first order polynomial fit is represented by

$$F_S(n_z) = -181.5 \cdot n_z + 181.5 \tag{4.11}$$

According to Figure 4.18, a stick force of $F_S = -90.75N$ is required, which corresponds to an elevator deflection of $\eta_{1.5} = 0.1°$ and an actuator deflection of $\alpha_{e,1.5} = -0.23°$. Of course, the maximum achievable load factor occurs at the maximum elevator deflection of $\alpha_{e,max} = 41.2°$. Figure 4.19 illustrates the resulting actuator command signal.

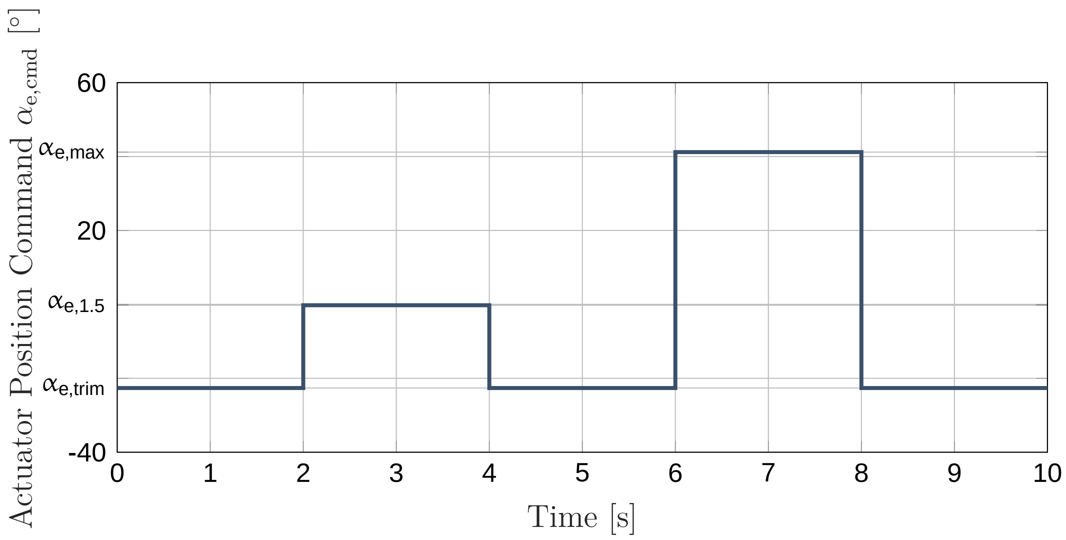


Figure 4.19: *Multi-step actuator command signal.*

4.3.2 Dynamic Simulation

The conducted dynamic simulations simulate a flight at very high altitudes for several combinations of actuator supply voltages (7V, 14V, 28V) and gear ratios (100, 200). Each configuration's simulation results are evaluated and discussed below.

Key Aspects and Boundary Conditions for the Evaluation

Since the AP is supposed to operate in major areas of the flight envelope, the minimum load factor to be achieved short term by the AFCS, was set to 1.75g, which corresponds to turns with a bank angle of slightly more than ±55°. Typical AP commands, however, are usually limited to $\Delta n_z = \pm 0.1g$.

The actuator rate plays an important role as well. It shall not be too fast for reaching higher load factors, as this means the actuator torque is very high and might harm the structure in the case of actuator runaways, for example. In addition, excessively high actuator or surface deflection rates become a problem for pilots with regards to their reaction times. Therefore, a maximum elevator deflection rate of $|\dot{\eta}_{max}| = 35\frac{°}{s}$ is defined, which results in 1 s for a full stop to full stop deflection. Naturally, the actuator rate must neither be too low, since in this case the AFCS may not react fast enough to the flight dynamics. Usually, the maximum deflection rate is limited by software.

Further, the coil temperature and power consumption should also be taken into account in the evaluation of simulation results. The environment temperature is set to $-10°C$ as the actuator is installed in an unpressurized area of the aircraft. According to the data sheets the actuator allows a maximum coil temperature of $T_{coil,max} = 180°C$.

Option 1: Supply Voltage 7V, Gear Ratio 100

The first configuration supplies the actuator with a maximum of 7 volts and the gearbox realizes a gear ratio of 100. The simulation results reveal that 95 percent of the initially commanded 1.5g are achieved after $\Delta t_{1.5,95\%} = 0.61s$ while the target load factor is achieved after $\Delta t_{1.5} = 0.75s$. When commanding the maximum elevator deflection, $n_{z,max} = 1.53g$ is reached after $\Delta t_{max} = 1.35s$ while 95 percent of this movement is already reached after $\Delta t_{max,95\%} = 0.85s$.

The longer duration for reaching 1.5g in the second step can be explained by the increased temperature of the coil caused by the first step. The same applies to the decreasing load factor after its peak value and to the decreasing voltage when holding the 1.5g level in the first commanded step. During both step commands, the temperature of the coil increased, which results in a higher resistance of the coil and consequently in less motor torque. The fact that the actuator does not reach the maximum commanded deflection and cannot hold the level after its maximum deflection suggests that the motor is operating at its limits.

Due to its low performance, this configuration is only limited suitable, depending on the performance requirements. However, the minimum performance requirement defined above are not met.

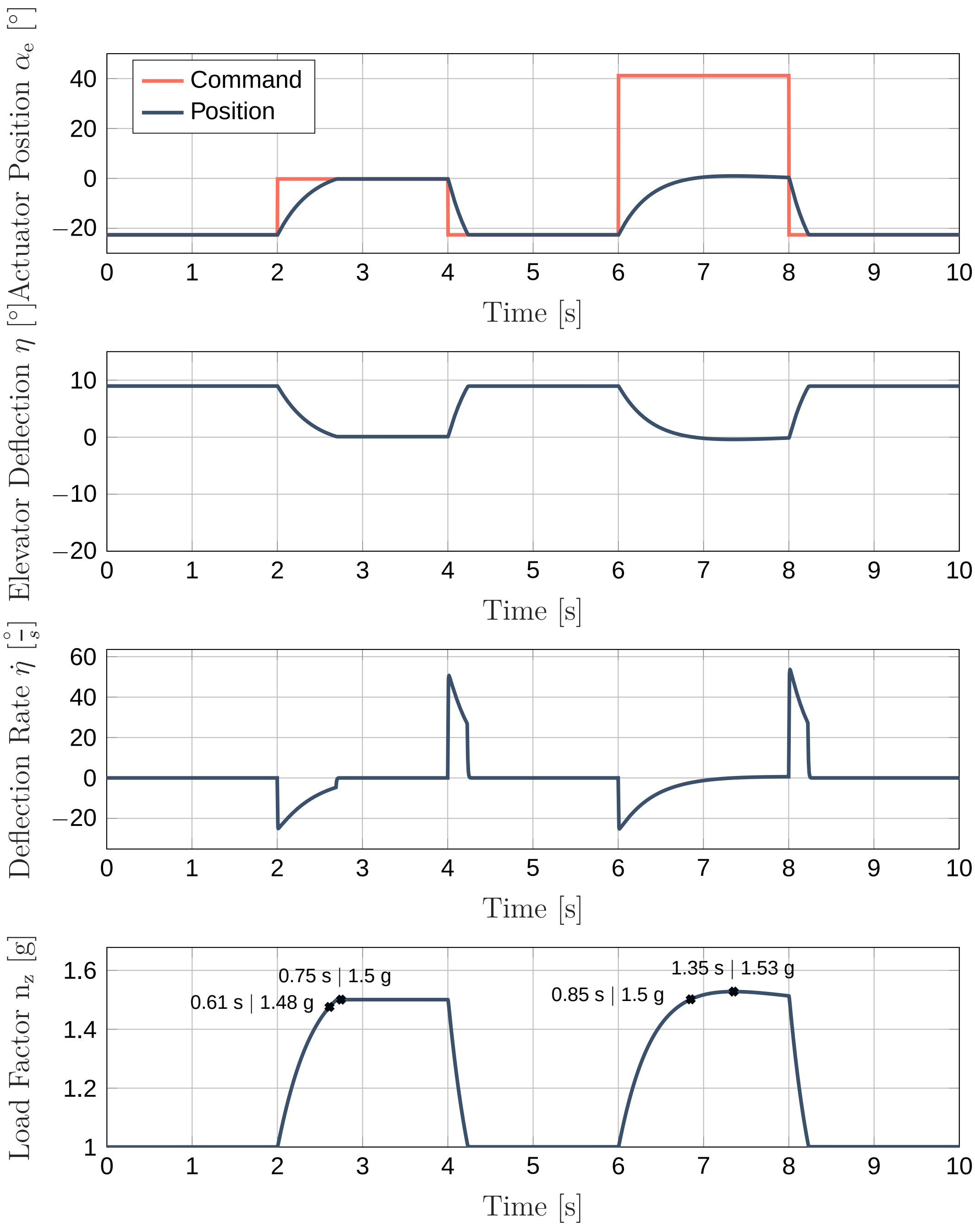


Figure 4.20: *Supply Voltage 7V, Gear Ratio 100.*

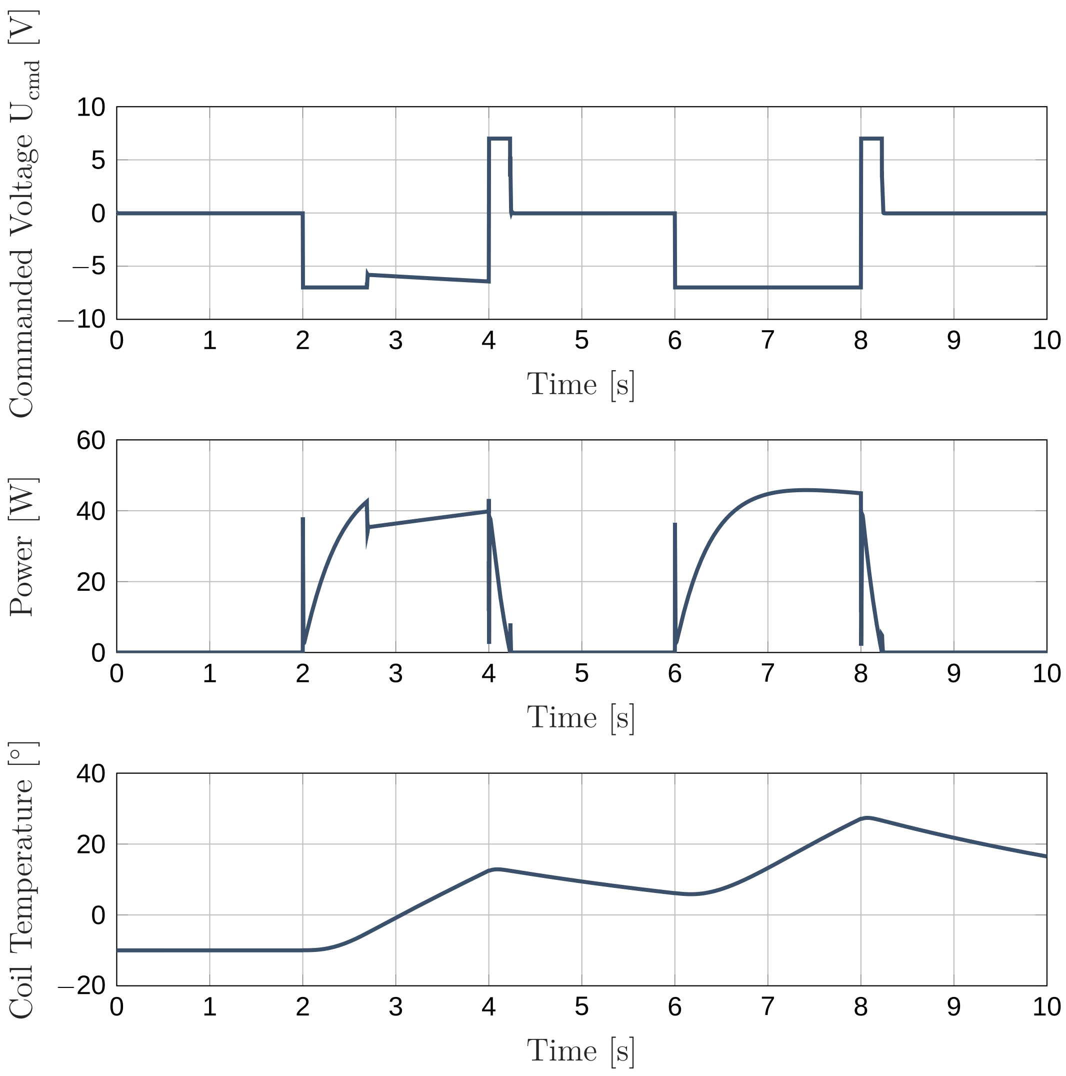


Figure 4.21: *Supply Voltage 7V, Gear Ratio 100.*

Option 2: Supply Voltage 14V, Gear Ratio 100

The supply of the same actuator and gearbox with a doubled voltage of 14 V results in a power four times higher. According to the simulation results, $n_z = 1.5g$ is reached more than twice as fast after $\Delta t_{1.5} = 0.28s$, whereas 95 percent of this level is achieved after only $\Delta t_{1.5,95\%} = 0.2s$. The maximum load factor of this configuration is $n_{z,max} = 2g$ and is achieved after $\Delta t_{max} = 1.03s$. After $\Delta t_{max,95\%} = 0.72s$ 95 percent of the maximum load factor are reached.

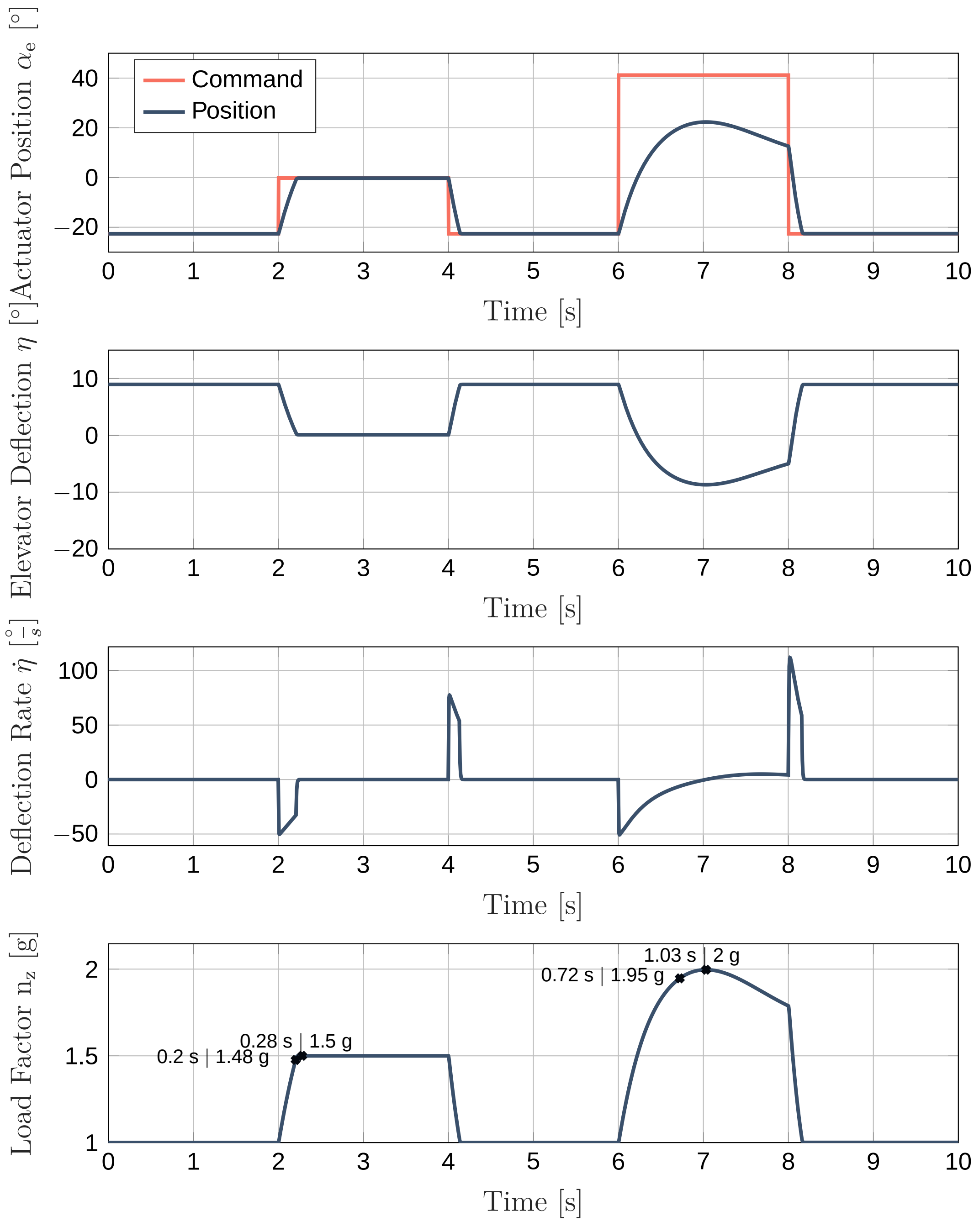


Figure 4.22: *Supply Voltage 14V, Gear Ratio 100.*

Figure 4.23 shows that the electrical power of the system has quadrupled, which is clearly not advantageous for a system with limited resources. The plots also show an increased coil temperature within the second step. This significant rise can be credited to the correspondingly higher power losses.

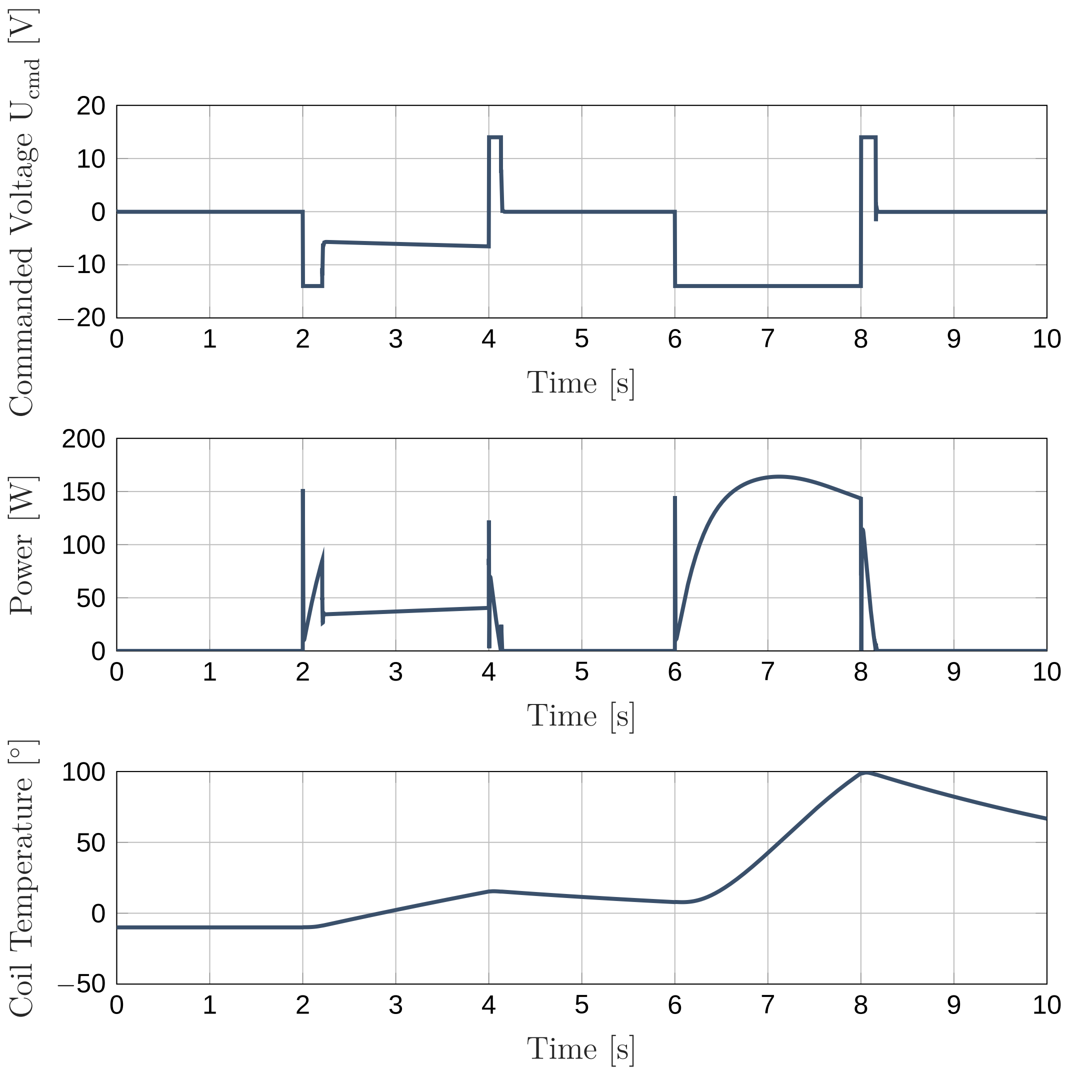


Figure 4.23: *Supply Voltage 14V, Gear Ratio 100.*

Option 3: Supply Voltage 28V, Gear Ratio 100

This configuration with a maximum voltage supply of 28 V and a gear ratio of 100, is the fastest among all those considered. After $\Delta t_{1.5} = 0.17s$ the commanded $n_z = 1.5g$, and after $\Delta t_{1.5,95\%} = 0.09s$, 95 percent of this level is achieved. The maximum elevator deflection, namely $n_{z,max} = 2.35g$ is reached after $\Delta t_{max} = 0.4s$ and can be held until the trim position is commanded again. Such fast actuator deflections are problematic, as mentioned above. The plot illustrating the elevator deflection rate in Figure 4.24 also confirms that the deflection rates are far too high, with more than $|\dot{\eta}_{max}| = 200\frac{\circ}{s}$. Such a configuration should definitely limit the maximum rates using software.

With the voltage being very close to the maximum voltage at $t = 8s$, it can be seen that the actuator operated at its limits when performing the maximum load factor step. The temperature rise within this phase can be problematic as it reaches the maximum value of $180°C$ for a short time. A reason for this is, of course, the again increased power consumption, which shows quadrupled peak values compared to a voltage supply of 14 V.

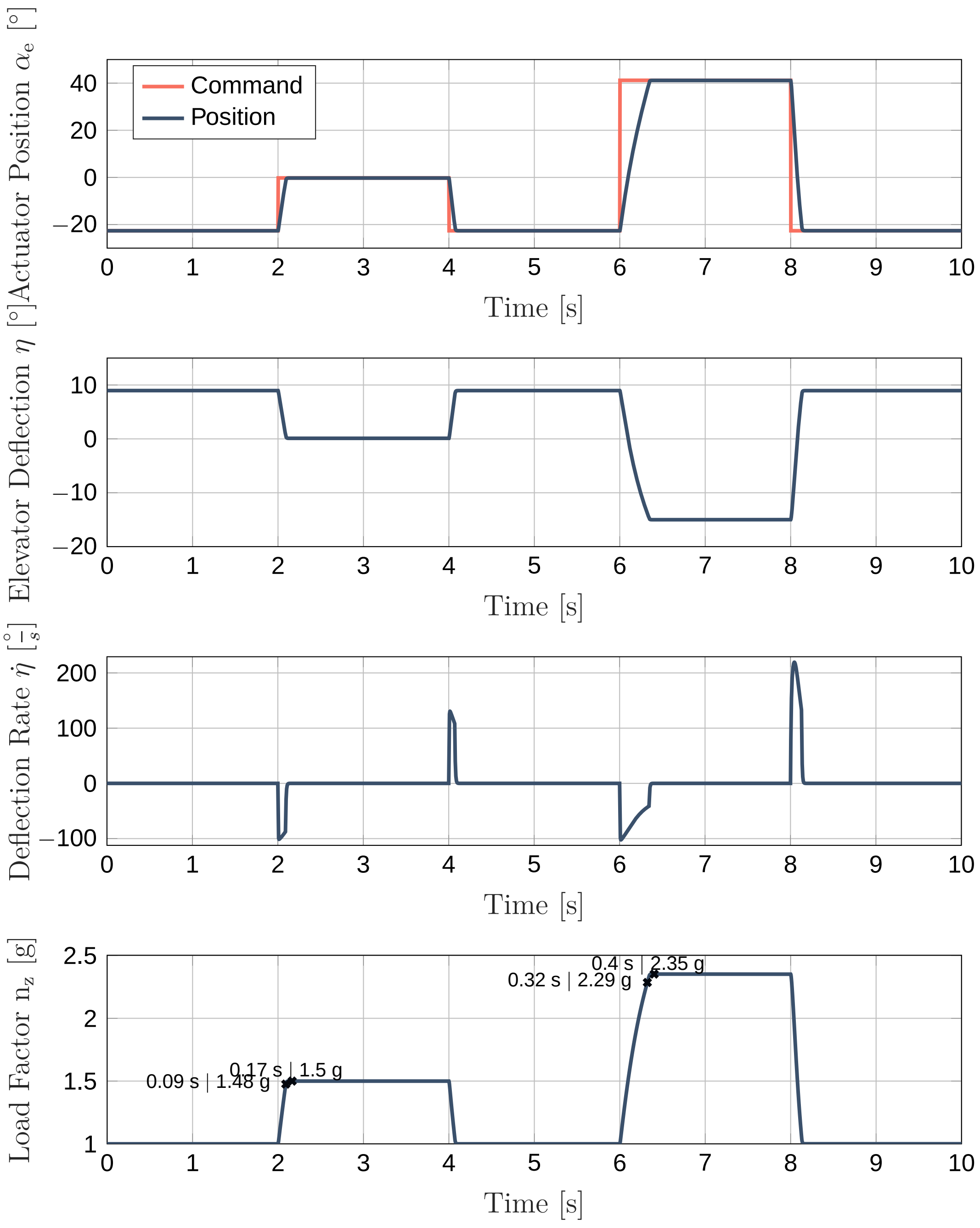


Figure 4.24: *Supply Voltage 28V, Gear Ratio 100.*

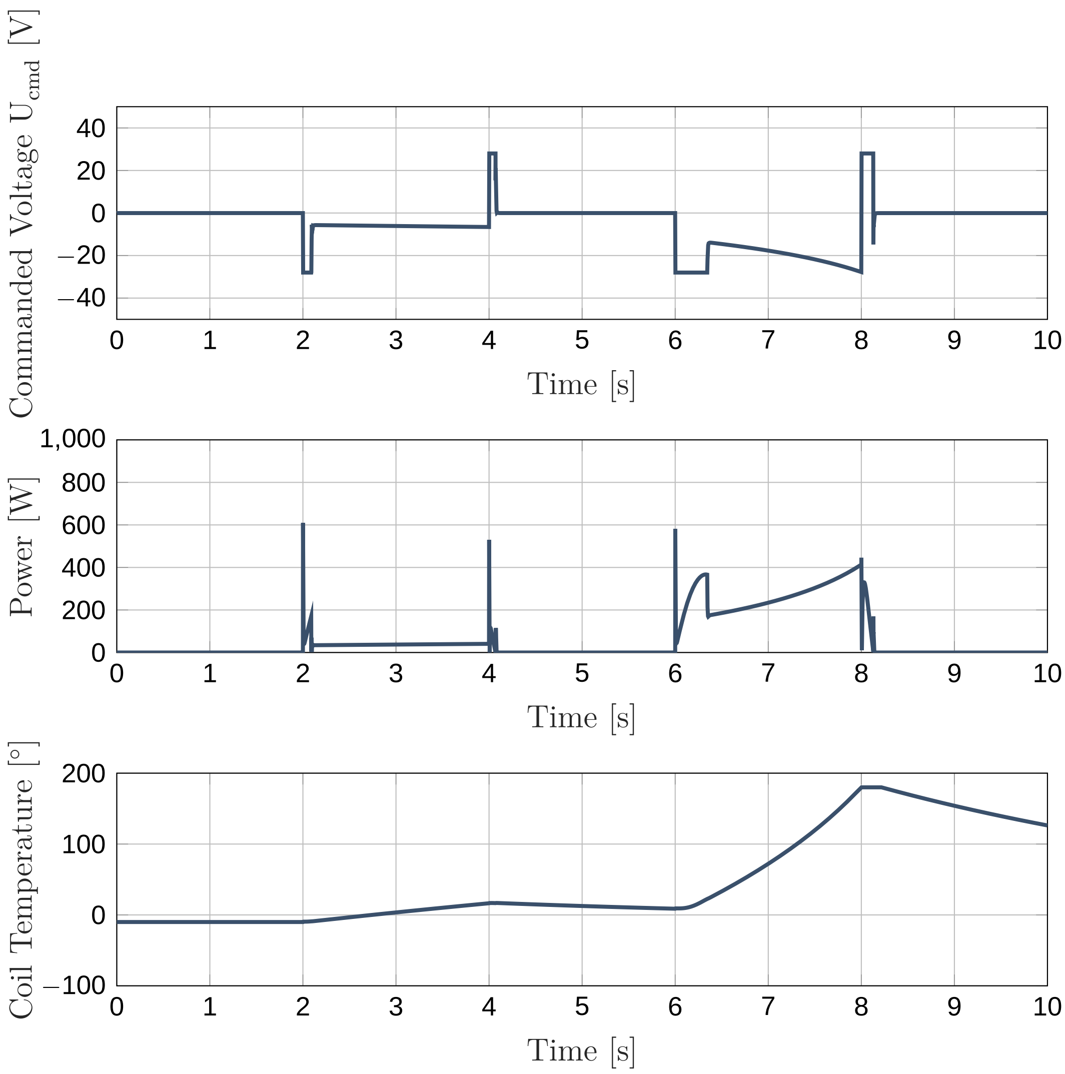


Figure 4.25: *Supply Voltage 28V, Gear Ratio 100.*

Option 4: Supply Voltage 14V, Gear Ratio 200

According to the simulation results of the previous configurations, a voltage supply of 14 V and a gear ratio of 200 appears to be promising. The results of the simulation show that this configuration reaches the $n_z = 1.5g$ after $\Delta t_{1.5} = 0.45s$ with a surface deflection rate of $|\dot{\eta}| = 25.5\frac{\circ}{s}$, which is acceptable. Due to the higher gear ratio, the actuator also reaches the minimum elevator deflection within $\Delta t_{max} = 1.41s$, which results in a load factor of $n_{z,max} = 2.35g$.

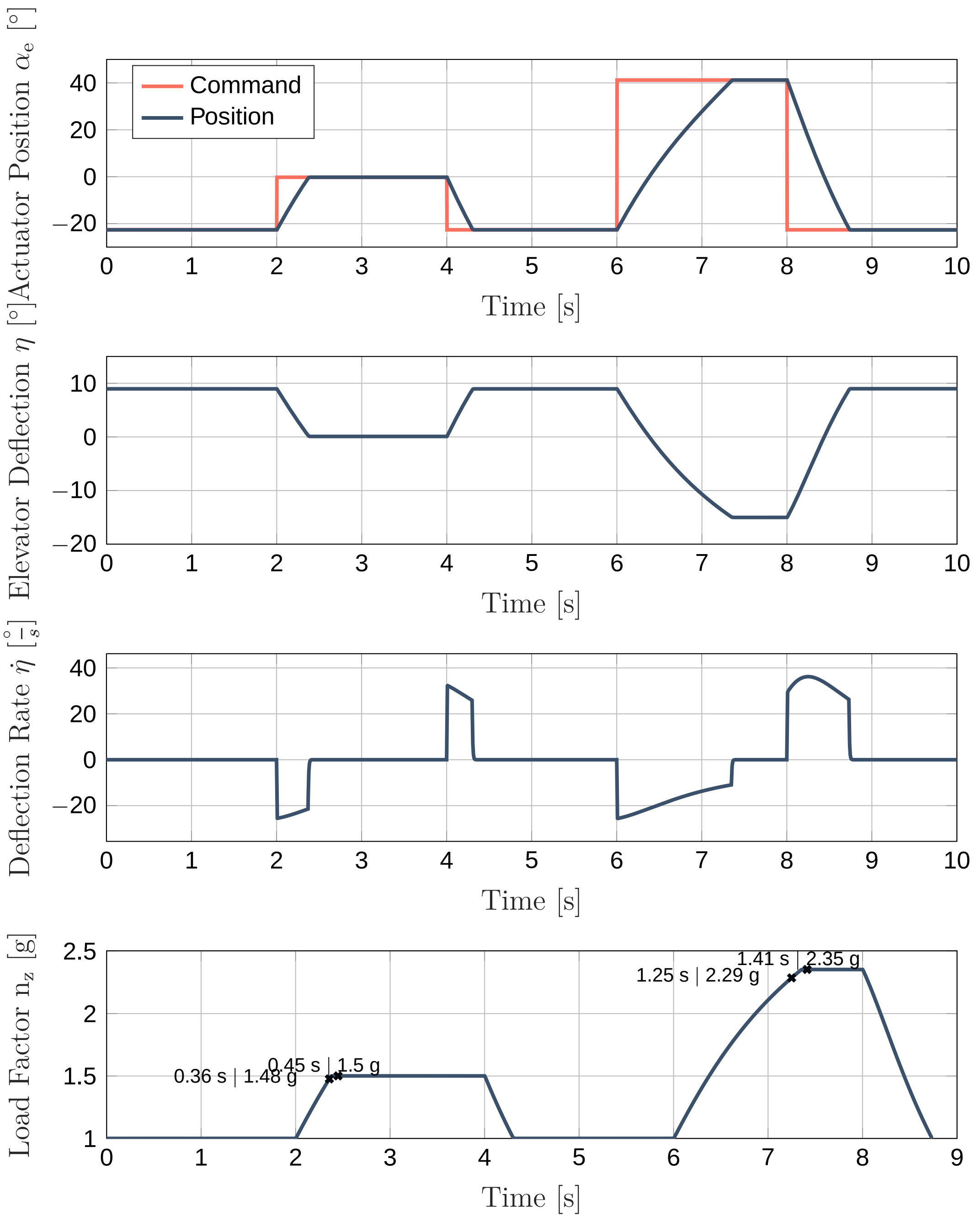


Figure 4.26: *Supply Voltage 14V, Gear Ratio 200.*

The surface deflection rate slightly exceeds the requirements for less than 300ms with a value of $|\dot{\eta}| = 36.2\frac{\circ}{s}$ when the maximum load factor level is left. Generally, the rates that are realized by this configuration should be fast enough in terms of flight dynamics but also slow enough to give pilots sufficient time to react.

This configuration is also characterized by its low temperature development. This is due to the low power consumption, especially during stationary system states. Peak values, however, still reach values of 150 W.

In this case, the gearbox reduced external loads for the actuator by half compared to the other configurations. Conversely, the actuator generates twice as much torque, which may be harmful for the aircraft structure in some cases of an error. In general, so called hard stops must be used to mechanically limit the actuator deflections and to protect the aircraft structure.

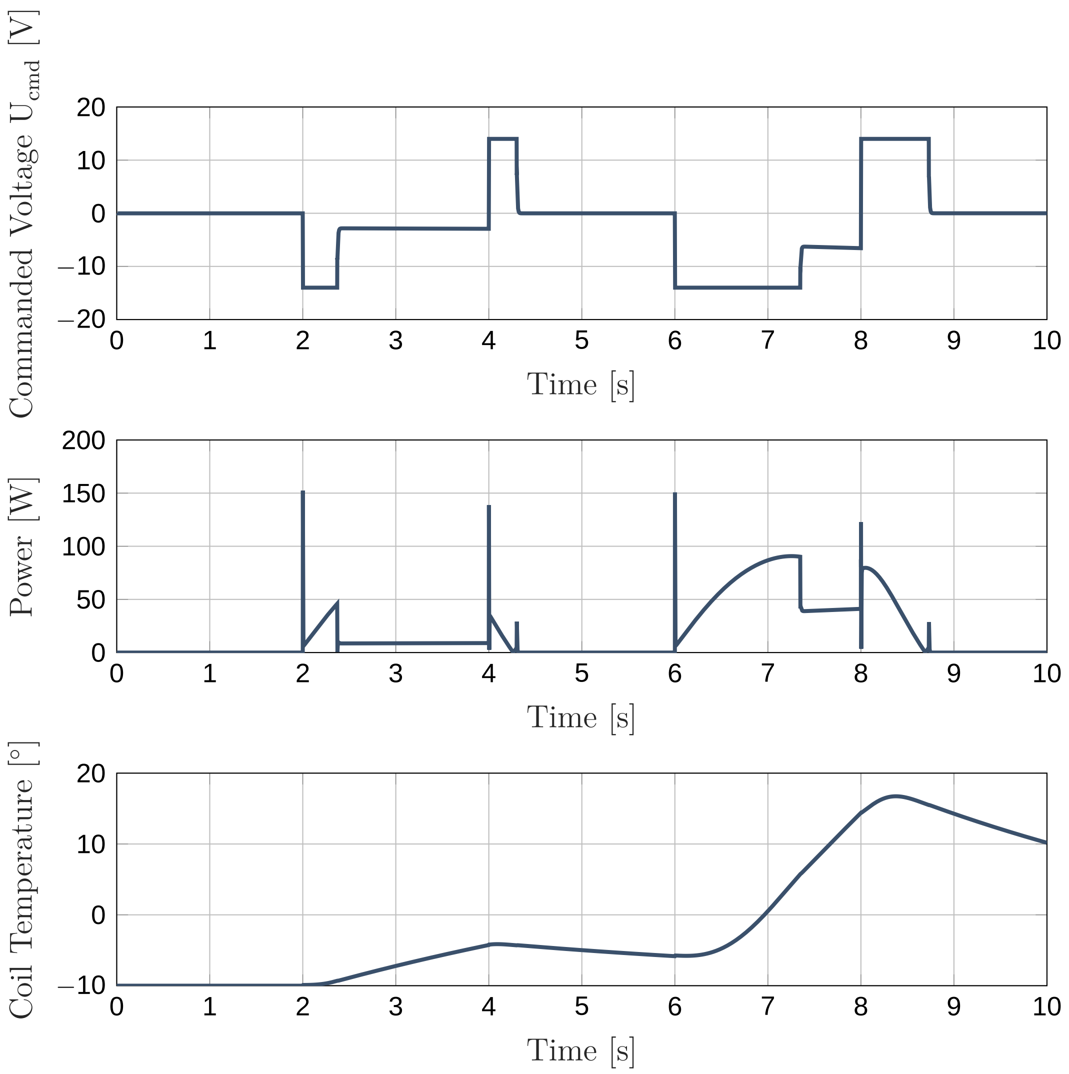


Figure 4.27: *Supply Voltage 14V, Gear Ratio 200.*

Conclusion and Next Steps

After analyzing the simulation results of the different voltage and transmission combinations, Option 4 appears to be the most appropriate. The performance in terms of maximum torque, surface deflection rate, and power consumption is the most satisfactory. Although the presented results look reasonable and considered friction in joints and actuator dynamics, the conducted simulations used static models for the hinge moments and did not consider the flight dynamics itself. This deficit is remedied in the following section, where a FDM is integrated into the simulation model.

4.4 High Fidelity Simulation with a Flight Dynamics Model

Using an FDM that represents the aircraft in question allows one not only assess the dynamic aircraft behavior but also to take into account the forces caused by the inertia during a maneuver. Consequently, this qualifies the simulation model for an analysis of the resulting handling qualities. For this reason, an FDM is included in the following simulation, which provides the necessary data to accelerate the MBS according to the aircraft's motion. In addition to this, backlash is simulated in certain joints to consider another physical effect occurring in real mechanical FCSs.

After the backlash is specified in the CAD model, it is converted again into a *Simscape Multibody* model. This is then modified to include the FDM, accelerate the mechanism, and represent the inertia of the elevator.

4.4.1 Inclusion of Backlash

The *Foundation Library* of the *Simscape* toolbox provides hard stop blocks that can be combined with *Simscape Multibody* blocks. However, it is not possible to simulate a circle-shaped clearance as it appears in revolute joints. Therefore, the backlash model presented in Chapter 2.2.2 is used for introducing backlash into the following simulation.

Since the backlash model is highly nonlinear, the simulation costs increase rapidly. This is why only certain but important joints are provided with this physical effect. Especially joints between the actuator and the elevator surface, which connect rods and levers, have a significant influence on the transfer function and are therefore enhanced with the backlash model (see red marked joints in Figure 4.28).

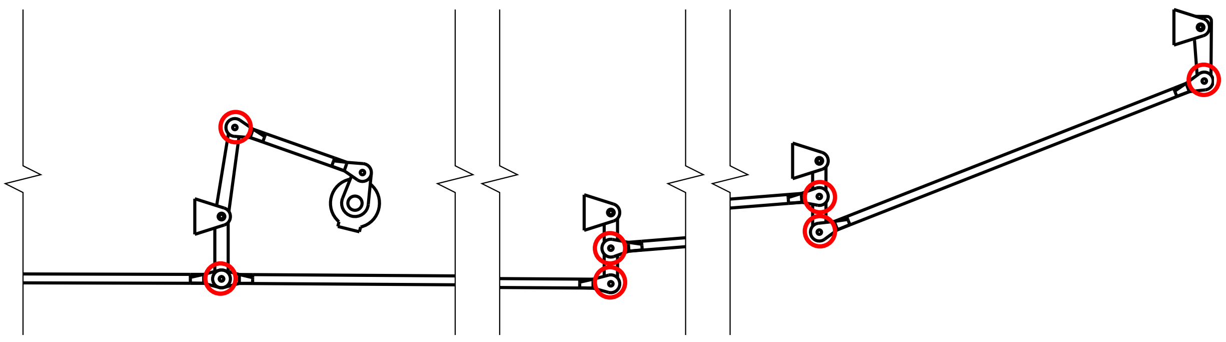


Figure 4.28: *Overview of the elevator axis.*

Analogous to the inclusion of friction, the definition of backlash in joints is undertaken in the CAD software. Therefore, the parameters below are added to the CAD model in the form of user-defined parameters:

```
backlashSetting1_Mechanism = Mechanismus.1
backlashSetting1_Joints    = Rotieren.11,Rotieren.13,Rotieren.14,
   Rotieren.16,Rotieren.17,Rotieren.19,Rotieren.22
backlashSetting1_Gap       = 0.0001/7
backlashSetting1_k         = 1e7
backlashSetting1_d         = 6e4
```

In this case, the clearance or gap for each joint is chosen such that the overall clearance sums up to $0.1mm$ over the entire actuator to elevator segment with the seven joints, including the backlash model.

4.4.2 Integration of the Flight Dynamics Model

The FDM of this particular aircraft was developed at the Institute of Flight System Dynamics as part of the research project. It includes nonlinear equations of motion, an environment simulation, motion kinematics, and an airframe block that contains weight and balance, propulsion, and aerodynamic models. The FDM is based on a comprehensive flight dynamics simulation framework, which was also developed at the institute and is further described in [ZSMH18].

The aerodynamic and propulsion data that had been identified for the design of the former AP served as a data source for the development of the model. In this particular case, the only input of the FDM is the elevator deflection, which is measured within the *Simscape Multibody* model. As outputs, the model returns the hinge moment for the elevator surface H_e and the aircraft's translational and angular accelerations. While H_e is directly routed to the corresponding joint, the acceleration data is used to accelerate the entire MBS according to the aircraft motion.

To obtain comparable results in the following analysis, the flight state of the FDM must be as similar as possible to that of the flight test data sheets used in the previous simulations. Unfortunately, the provided data is partly deficient, which leads to an altered momentum balance in the FDM. This results most importantly in a changed elevator trim position η_{trim} for the same flight state. Nevertheless, in order to get comparable simulation results, a stationary bias for the elevator deflection is used. Therefore, the η measured from the MBS model is subtracted from the bias of 8.96°, while the bounds η_{min} and η_{max} in the FDM are disabled.

Furthermore, the institute's trim routines use only the elevator surface for trimming, but the trim position for the trim tab is not calculated. This is necessary to reach a force-free actuator and with that a hinge moment $H_e \simeq 0$ in a trimmed flight state. With a relatively small trim tab area, but a large lever arm between the trim tab's aerodynamic center and the hinge line, it has a significant influence on the elevator hinge moment, which can be calculated according to [Hol17b] as

$$H_e = \overline{q}_t \cdot S_e \cdot \overline{c}_e \cdot C_{h,e} \tag{4.12}$$

where $\overline{q}_t$ is the dynamic pressure at the tail, S_e is the elevator wing area behind the hinge line, $\overline{c}_e$ is the mean aerodynamic chord of the elevator, and $C_{h,e}$ is the hinge moment coefficient of the elevator. This coefficient can be assumed as a linear function of the angle of attack, the surface deflection, and the trim tab deflection as

$$C_{h,e} = C_{h0,e} + C_{h\alpha,e} \cdot \alpha_t + C_{h\eta,e} \cdot \eta + C_{h\delta_{tab}} \cdot \delta_{tab} \tag{4.13}$$

with $C_{h0,e}$ as a static hinge moment coefficient, $C_{h\alpha,e}$ the hinge moment coefficient depending on the angle of attack, α_t the angle of attack at the tail, $C_{h\eta,e}$ the hinge moment coefficient depending on η, and $C_{h\delta_{tab}}$ the hinge moment coefficient depending on δ_{tab} the deflection of the trim tab. Setting Equation 4.13 equal to zero allows the trim tab deflection to be calculated eliminating the elevator hinge moment:

$$\delta_{tab} = -\frac{C_{h0,e}}{C_{h\delta_{tab}}} - \frac{C_{h\alpha,e}}{C_{h\delta_{tab}}} \cdot \alpha_t - \frac{C_{h\eta,e}}{C_{h\delta_{tab}}} \cdot \eta \tag{4.14}$$

Together with the aerodynamic coefficients, α_t from the simulation for the trimmed flight state, and $\eta_{trim} = -13.08°$, the trim tab has to be deflected by $\delta_{tab} = 15.14°$.

4.4.3 Acceleration of the Mechanism

As previously mentioned, the FDM returns the translational and angular accelerations acting on the aircraft. In the following simulation, this information is used to move the mechanism in the MBS, aiming for the mechanical FCS to experience the same accelerations as the aircraft. As a result, the inertia loads due to flight maneuvers can be taken into account (as stated in 1.7 Contribution). This is achieved by introducing a combination of a *Cartesian Joint* (3 translational DoFs) and a *Gimbal Joint* (3 rotational DoFs) between the fixtures of the FCS and the *World Frame*. This system, shown in Figure

4.29, moves a coordinate frame that corresponds to the reference point of the aircraft in the FDM. Accordingly, to obtain reasonable results, the mechanism is then correctly positioned and orientated relative to this point after the *Gimbal Joint* block.

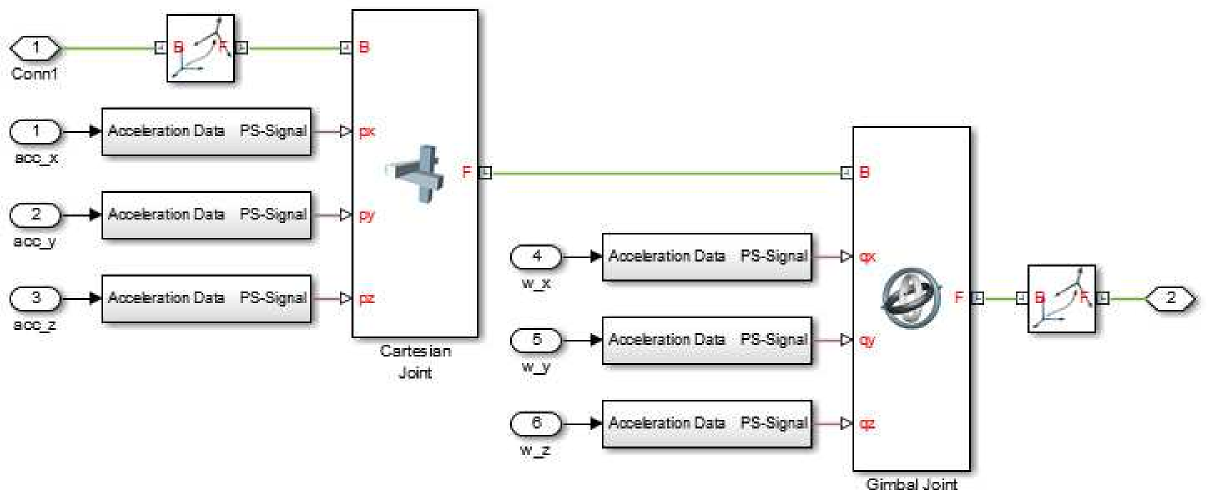


Figure 4.29: *Introduction of the acceleration data to the MBS model.*

Analogous to Section 4.2.1, the acceleration data, in the form of an ordinary *Simulink* signal, must be converted to a *Physical Signal* before it is led into the joint blocks. Therefore, the data is integrated twice in order to generate the velocity and the position data.

4.4.4 Representation of the Elevator Surface Inertia

The elevator surface itself is not part of the CAD model and thus there is no consideration of its inertia within the MBS. For a more detailed and more realistic simulation, the actual inertia data is approximated and then integrated into the model in the form of two point masses. One represents both elevator surfaces, including the trim tab, and another one represents both compensating masses, which are designed to lower the static hinge moment. The data provided by the aircraft manufacturer is listed in Table 4.6.

Table 4.6: *Elevator control surface moments, weights, and mass balances.*

Part	Mass	stat. Moment
Elevator (R) without compensating mass	$9.6kg$	-
Elevator (R) compensating mass included	$10.7kg$	$400.5Ncm$
Elevator (L, Trim tab included) without compensating mass	$11.1kg$	-
Elevator (L, Trim tab included) compensating mass included	$14.6kg$	$387.0Ncm$
Compensating mass is installed 42cm in front of hinge axis		

The mass of the elevator and the point mass data for the compensating mass can be directly derived from the table:

$$m_{el} = 9.6kg + 11.1kg = 20.7kg \tag{4.15}$$

$$m_{cm} = 1.1kg + 3.5kg = 4.6kg \tag{4.16}$$

$$x_{cm} = -0.42m \tag{4.17}$$

The equilibrium of moments for both point masses about the elevator hinge line can be formulated as

$$g \cdot (x_{cm} \cdot m_{cm} + x_{el} \cdot m_{el}) = 400.5Ncm + 387.0Ncm \tag{4.18}$$

where the lever arm for the elevator point mass can then easily be calculated as

$$x_{el} = \frac{7.875Nm - x_{cm} \cdot m_{cm} \cdot g}{m_{el} \cdot g} = 0.1321m. \tag{4.19}$$

Such a model, of course, does not exactly represent the moment of inertia of the elevator control surface but is a reasonable approximation when more precise data is not available.

4.4.5 Scenario and Simulation Results

The scenario for the set up high fidelity simulation differs from the scenario of the simulations conducted in Chapter 4.3.2. The reason for this is that the second maneuver for reaching the maximum load factor would not start from a trimmed flight state. Therefore, the following simulation only focuses on the maximum achievable load factor and neglects the first maneuver for reaching $1.5g$.

The plots showing the simulation results in Figure 4.30 indicate that actuator tracking and performance is very similar to that of Chapter 4.3.2. In particular, it reaches and holds the commanded position. Moreover, it can be seen that the elevator surface deflection follows the actuator movement. Nevertheless, the maximum upward deflection and the minimum value is only $min(\eta) = -14.35°$. The reason for the deviation from $\eta_{min} = -15°$ is the backlash introduced at certain joints, which compensates for parts of the movement. The consequences of backlash are also noticeable when the elevator surface position changes, most notably at $t = 3.9s$. The effects are more visible in the course of the surface deflection rate. In terms of the load factor, the maximum reached value of $n_z = 2.07g$ only deviates $0.28g$ from the simulation results, as presented in Chapter 4.3.2. Likewise, the time horizon of both simulations very closely match. The maximum load factor is reached within $\Delta t_{max} = 1.44s$, which is only $30ms$ longer than the Δt_{max} of the previous study. This is an indication that the results and the assumptions made in the previous simulation are quite plausible. Furthermore, the course of the load factor reveals the non-minimum phase behavior of the aircraft at $t = 1s$, which results from an initial height loss due to the negative lift generated by the elevator deflection right before the pitching moment sets in.

Figure 4.31 contains the electrical power P_{el} plot, which shows peak values at points where a surface movement begins, similar to the previous simulation results. The power output through the reverse driven surface, however, is significantly smaller. This is mainly related to a lower restoring hinge moment and to the lowering load on the actuator during the maneuver, which results from a decreasing hinge moment as the angle of attack at the tail α_t decreases. The lower power values lead to less power dissipation in the form of thermal energy. The total temperature rise over the entire maneuver is only $\Delta T_{coil} \approx +6°$ compared to $\approx +20°$ in the simulation of Option 4 in Chapter 4.3.2.

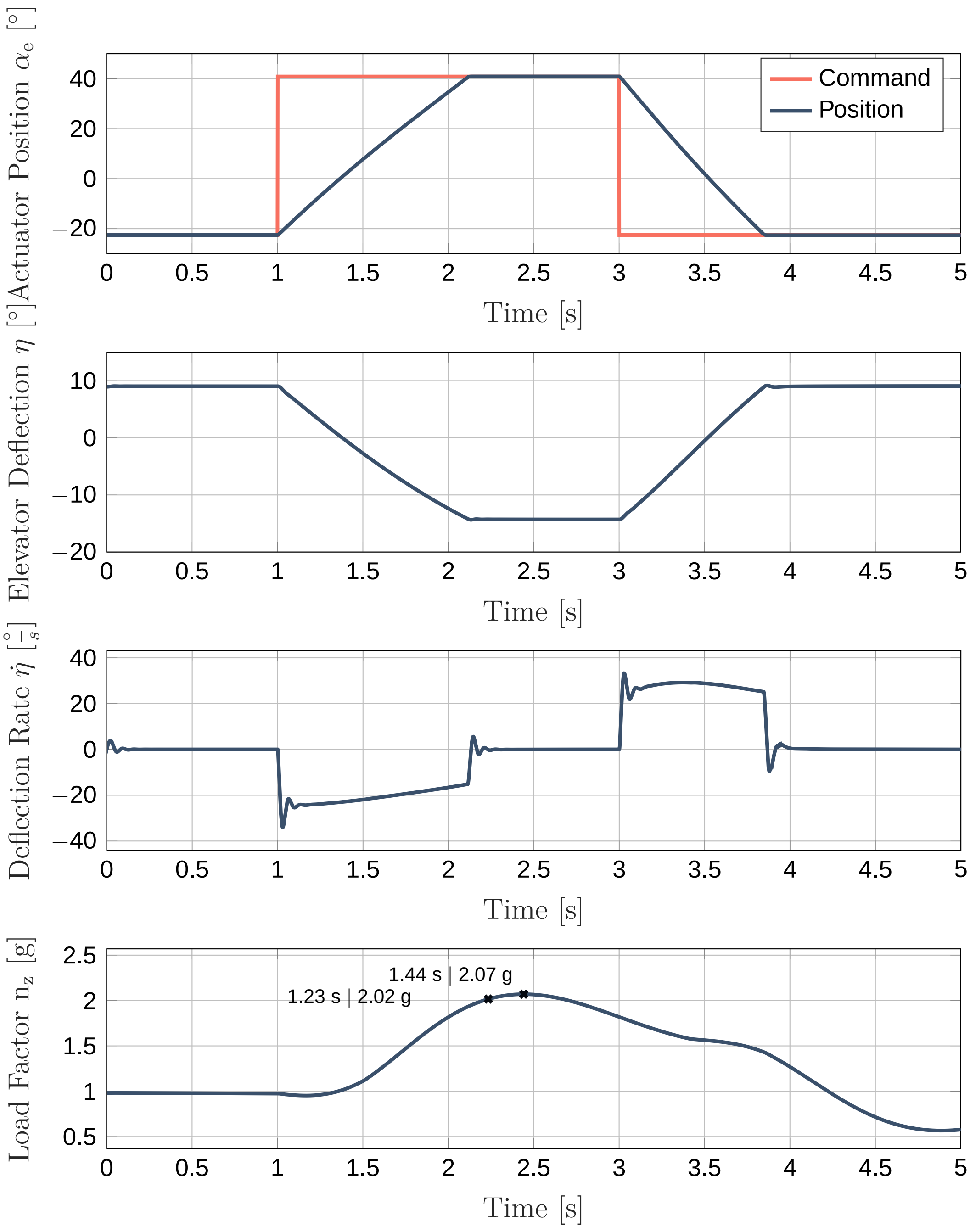


Figure 4.30: *High fidelity simulation results.*

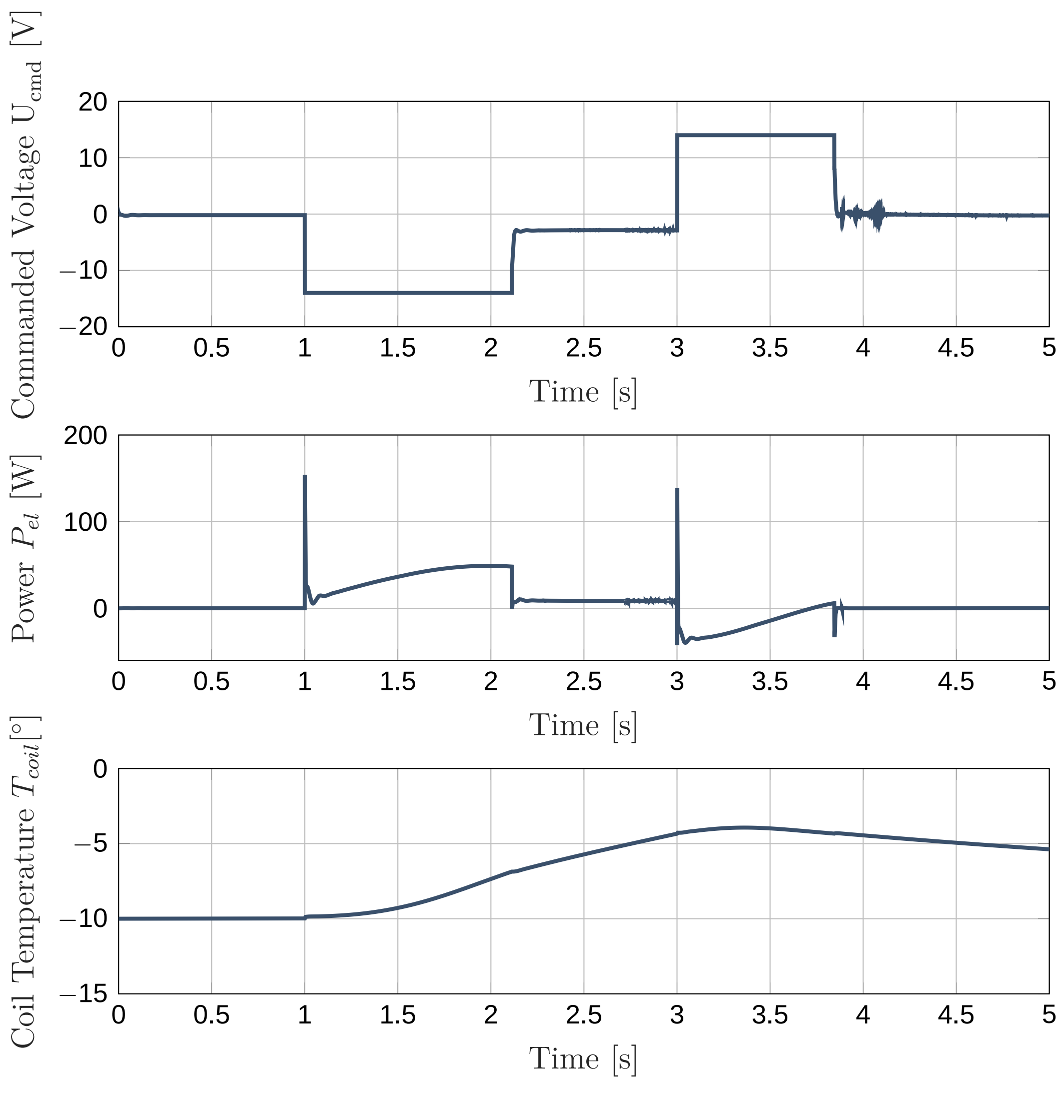


Figure 4.31: *High fidelity simulation results.*

Aircraft Response

For the sake of completeness, the response of the aircraft is also evaluated. The graph in Figure 4.33 represents $(V_K^R)^E$, the kinematic velocity of the aircraft reference point with respect to the *Earth Centered Earth Fixed* (ECEF) frame. Further information on the coordinate frames and notation can be found in [Hol17a].

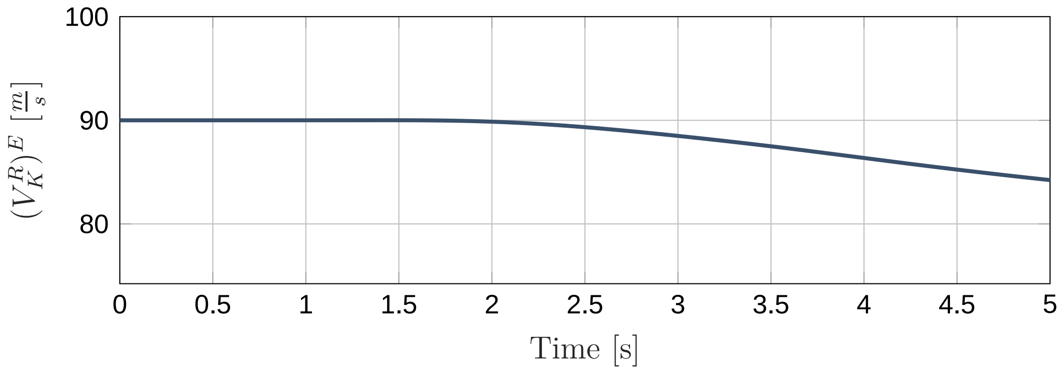


Figure 4.32: *Plot of the airspeed in the high fidelity simulation.*

To ensure that the maneuver is actually flyable, the stall speed has to be reviewed for the most critical flight state, which occurs at $t = 2.44s$ with the maximum load factor. The *Pilot's Operating Handbook* (POH) for this aircraft provides the stall speeds depending on the deflection of the flaps, the aircraft mass, and the bank angle of the aircraft. According to [Hol17a], the correlation between load factor and bank angle μ can be described as

$$n_z = \frac{1}{\cos(\mu)} \tag{4.20}$$

which leads to $\mu = 61.1°$ for the maximum load factor of $2.07g$. Under the given conditions, the POH indicates a stall speed of 90 KIAS, which corresponds to an aerodynamic airspeed of $V_{A,Stall} = 46.3\frac{m}{s}$. Since the wind speed is set to zero in this scenario, the kinematic velocity is equal to the aerodynamic velocity. Accordingly, the airspeed during the maneuver is clearly above the stall speed stated in the POH.

The first graph in Figure 4.33 shows the pitch angle Θ between the *Body Fixed* frame and the *North East Down* (NED) frame. Together with the second graph, which represents the aerodynamic angle of attack α_A between the *Aerodynamic* frame and the *Body Fixed* frame, it can be concluded that the aircraft is flying straight level first. This can also be seen in the last graph of Figure 4.33, where the climb angle γ_K between the NED frame and the *Kinematic* frame is plotted. At $t = 1s$ the actuator movement and thus the moment dynamics set in. The $\Delta\eta$ causes a positive moment around the pitch axis of the aircraft, leading to an angular acceleration and consequently to an angular rate that changes the angle of attack. This is visible in the plots of Θ and α_A until around $t = 1.5s$. While both angles are already rising, γ_K remains more or less unchanged. With a change of the aircraft's angle of attack $\Delta\alpha_A$, the force dynamics then set in, which causes an increase in lift. This in turn leads to a change of the climb angle and, ultimately, flight altitude. The effects of the force dynamics can be seen in the course of γ_K, which starts rising only after $t = 1.5s$.

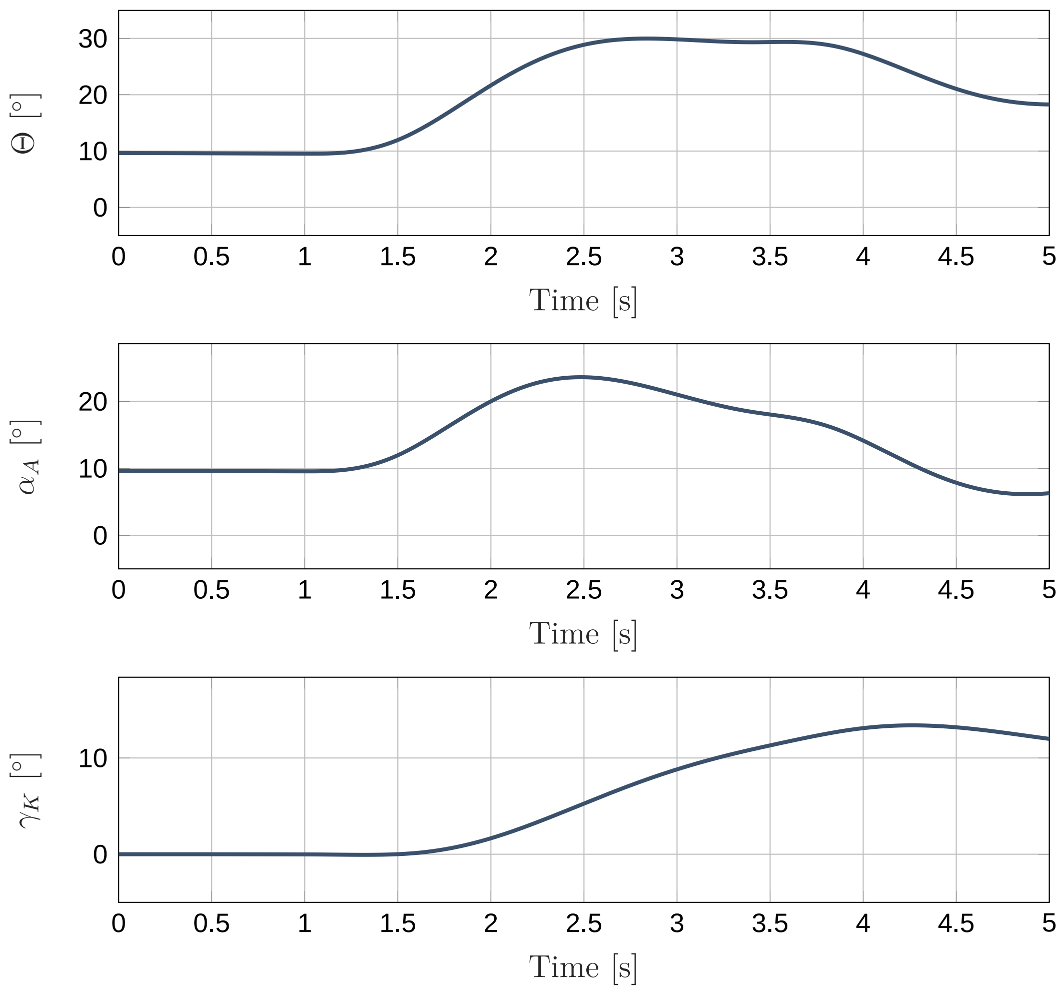


Figure 4.33: *Aircraft response in the high fidelity simulation.*

4.5 Aircraft Handling Qualities

As stated previously, the level of detail of the built simulation model also allows aircraft handling qualities to be reviewed. According to [Coo69], handling qualities refer to "*those qualities or characteristics of an aircraft that govern the ease and precision with which a pilot is able to perform the tasks required in support of an aircraft role*". The following focuses on the stick force per g relationship as part of the airplane's control characteristics. The gradient describes the stick force that is needed to produce a change of the load factor and is an important parameter of the basic handling qualities. In its CS-23 ([Eur15]), EASA provides a minimum elevator control force in maneuvers (CS 23.155), which is defined as the stick force required to achieve the positive limit maneuvering load factor.

For stick controls, this is the greater value between $66.8N$ and

$$F_{Stick,n_{z,max},min} = \frac{W}{14N} \tag{4.21}$$

where W is the maximum aircraft weight in kg. However, it does not need to be greater than $156N$. The reason for the lower limit is to ensure that the aircraft is not overstressed too easily, so that a certain force is necessary to reach the positive limit of the aircraft. On the other hand, there is also a good reason for the upper limit of the parameter, which is in any case not dependent on the aircraft. This limit is supposed to ensure that the aircraft remains controllable without too much effort and that the pilots do not fatigue too quickly. The EASA *Certification Specifications 25* (CS-25) for large aeroplanes does not contain such maneuvering control force requirements.

The minimum stick force per g gradient is also part of the *United States Military Standard*. The *MIL-HDBK 1797* ([U.S97]) calculates the minimum gradient for level 1 aircraft and center stick controllers as

$$\left(\frac{\Delta F_{Stick}}{\Delta n_z}\right)_{min} = \frac{21lb}{n_{z,max} - 1}. \tag{4.22}$$

According to the formula provided in the *MIL-HDBK 1797*, the minimum gradient for the considered aircraft is $\left(\frac{\Delta F_{Stick}}{\Delta n_z}\right)_{min,MIL} = 40.9N$. However, using the formula from the CS-23, the minimum gradient is calculated as $\left(\frac{\Delta F_{Stick}}{\Delta n_z}\right)_{min,CS} = 68.4N$. Since the CS-23 is relevant for the certification of the aircraft, its requirements must be met. Nevertheless, fulfilling the requirements from the *MIL-HDBK 1797* is advisable, as it considers aspects that are not covered by the CS-23.

The stick force of the current FCS can easily be derived using a minor modification of the simulation model used in Section 4.4. For this purpose, an *inverse dynamics* problem is formulated at the control inceptor. Instead of the actuator model governing the motion of the mechanism, the joint connecting the control stick and its fixture is now used. Therefore, its motion is specified in such a way that the control stick deflection is almost identical to the simulation of the previous section. Due to the introduced backlash, it is not possible to reproduce the exact same motion as before. After this modification, *inverse dynamics* algorithms allow one to calculate the torque needed to perform the predefined motion. Together with the length of the control stick, the stick force or required pilot force can be calculated. Figure 4.34 shows the elevator deflection, the load factor, and the pitch angle over time during the maneuver. The actual stick force and the required stick force are shown in Figure 4.35. While the red graph represents the (smoothed) stick force required for the pull-up maneuver, the blue graph shows the minimum stick force

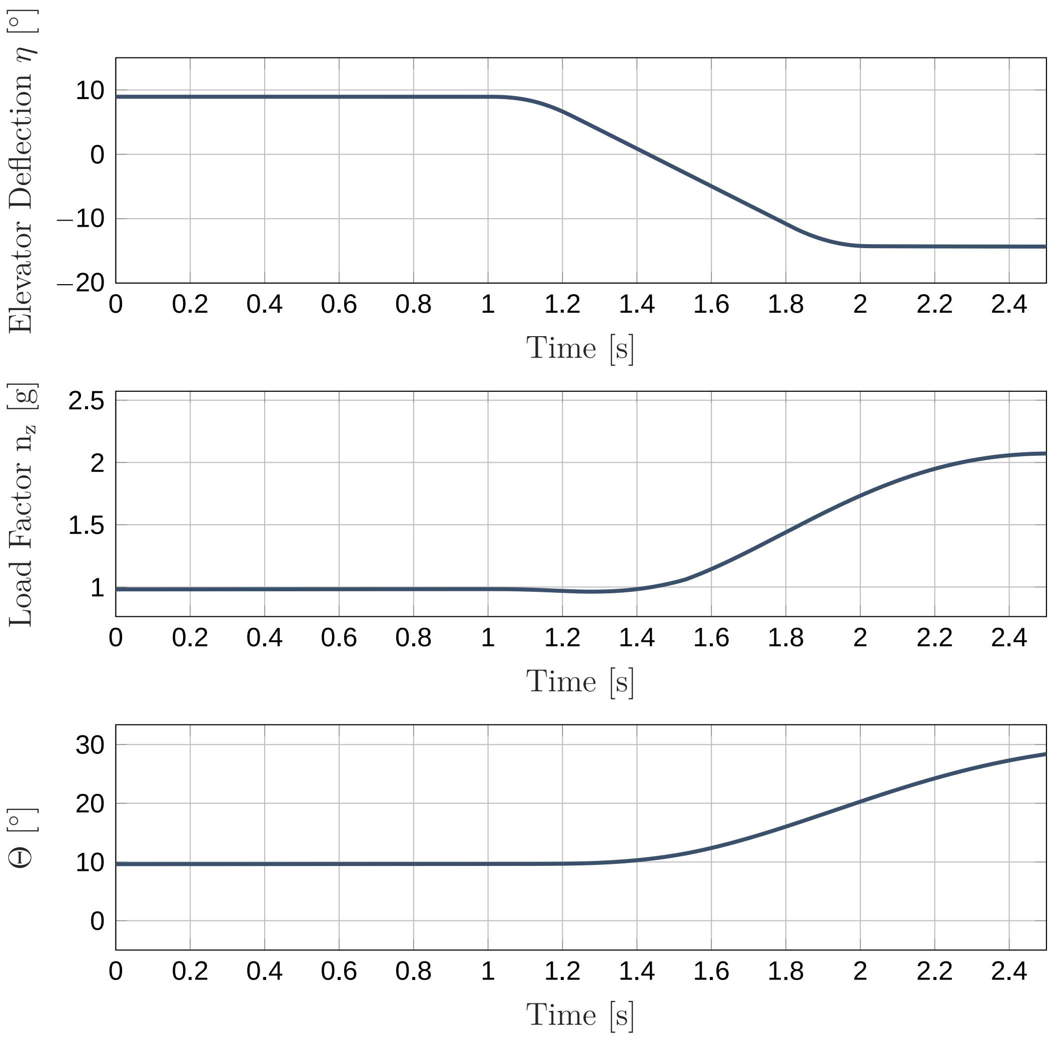


Figure 4.34: *Elevator deflection, load factor, and pitch angle over time.*

required by the CS-23:

$$F_{Stick,min}(t) = \left(\frac{\Delta F_{Stick}}{\Delta n_z}\right)_{min,CS} \cdot n_z(t) = 68.4N \cdot n_z(t) \tag{4.23}$$

As can be seen, the actual stick force is mostly above the required stick force. Only shortly before reaching the maximum load factor at the end of the simulation it falls slightly below the minimum force. This shortfall is probably due to inaccuracies in the models, since the aircraft is already CS-23 certified.

Moreover, the maximum stick force does not occur at the same time as the maximum load factor. Initially, F_{Stick} correlates with the hinge moment that is shown in Figure 4.36. The temporal offset is quite reasonable, since the hinge moment is primarily dependent on the moment dynamics. These are significantly faster than the force dynamics, by which the load factor is largely influenced. Equations 4.12 and 4.13 can also explain the kink in the hinge moment curve at $t = 1.95s$. On the one hand, the dynamic pressure $\overline{q}_t$ decreases as it is proportional to the square of the airspeed, which decreases over the

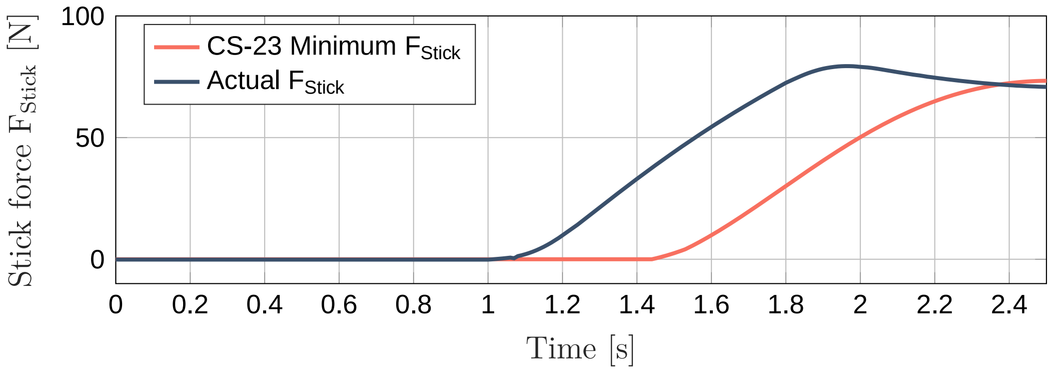


Figure 4.35: *Stick force and minimum stick force over time.*

entire maneuver. On the other hand, the hinge moment coefficient is composed of, among others, the η term, which is the main determinant, and the α_t term, which decreases with an increasing α_t. This expresses the decreasing restoring moment with an increasing angle of attack. Consequently, H_e reaches its peak value as soon as the elevator is fully deflected. After this point, the η term remains unchanged, but the angle of attack still increases, leading to a decreasing α_t term and finally to a decreasing H_e.

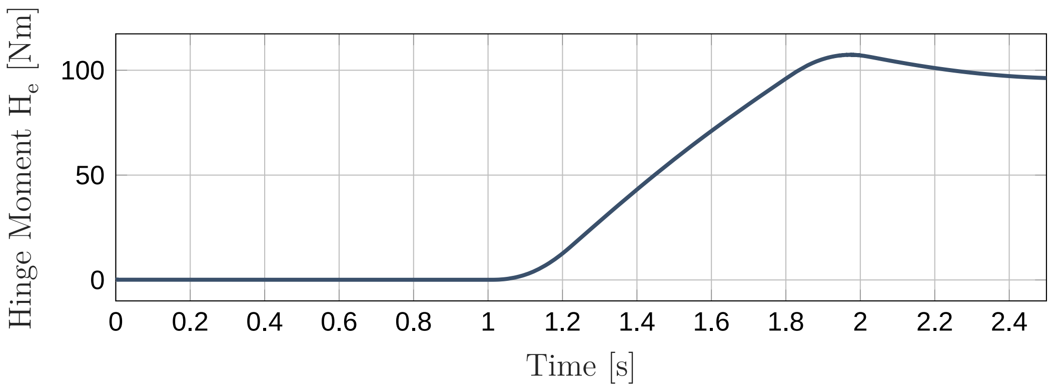


Figure 4.36: *Hinge moment over time.*

In order to bring the model of the mechanical FCS in line with the requirements of the CS-23, the stick force per g gradient is to be adapted. In accordance with [Hol17b], the stick force per g gradient can be calculated from

$$\frac{\Delta F_{Stick}}{\Delta n_z} = G_e \cdot S_e \cdot \overline{c}_e \cdot \frac{\overline{q}_t}{\overline{q}} \cdot \frac{mg}{S} \cdot \frac{C_{h\eta,e}}{C^N_{m\eta}} \frac{x^{GM_{free}}}{\overline{c}} \tag{4.24}$$

where G_e is the mechanical transmission ratio between control stick and elevator surface, $\overline{q}$ is the dynamic pressure, m is the aircraft mass, g is gravitational acceleration, S is the wing area, $C^N_{m\eta}$ is the elevator pitching moment derivative around the aircraft neutral point, $x^{GM_{free}}$ is the distance between the aircraft CG and the stick-free maneuver point, and $\overline{c}$ is the mean aerodynamic chord. Consequently, the main determinants on the gradient are

- the mechanical transmission ratio G_e,
- the elevator wing area S_e,
- the wing loading $\frac{mg}{S}$,
- the aircraft's CG, and the flight altitude (determining $x^{GM_{free}}$).

However, none of the listed determinants shall be used. Instead, the control inceptor is equipped with a bob-weight, a common control system gadget to tailor the stick force per g gradient. In [Ros03], the relationship between a bob-weight and the stick force for relatively shallow pitch angles is given as

$$\Delta F_{Stick} \approx \frac{n_z \cdot g \cdot m_{bw} \cdot l_{bw}}{l_s} \tag{4.25}$$

with the bob-weight mass m_{bw}, the bob-weight lever length l_{bw}, and the stick length l_s. Nevertheless, for higher pitch angles, the stated relationship can no longer be applied. For this purpose, the bob-weight is considered in a dynamic simulation, which enables a much more detailed examination of the bob-weight effects. Figure 4.37 illustrates the installation point of the bob-weight near the rotational axis of the control stick. The greater the distance to the axis, the faster the effective lever arm and thus the bob-weight effects are reduced, as the pitch angle or the stick deflection (in the pull direction) increases.

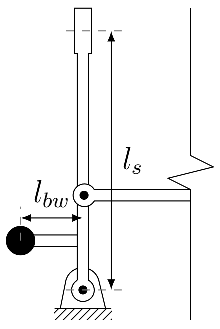


Figure 4.37: *Schematic diagram of the bob-weight installation.*

With $l_s = 1m$, $l_{bw} = 0.4m$, and $m_{bw} = 1.5kg$, the bob-weight is implemented as a point mass $0.05m$ above the rotational axis. The resulting adapted stick force with the bob-weight is shown in Figure 4.38. This is in contrast to the unadapted force of Figure 4.35. The results were obtained by applying the identical motion to the FCS. The actual stick force remains above the minimum stick force according to CS-23. Moreover, it can be seen that the bob-weight causes a constant stick force augmentation of $5.81N$ in a straight-level flight with $n_z = 1g$. Despite the trim pitch angle and trim stick deflection, this is very close to the approximation given by Equation 4.25 with $\Delta F_s \approx 5.89N$. In the subsequent course, the force caused by the bob-weight inertia increases with a rising load factor, but at the same time, the effective bob-weight lever arm decreases with a rising pitch angle and an increasing stick deflection. Since both effects work against each other, this leads to an almost constant force augmentation over the entire simulation.

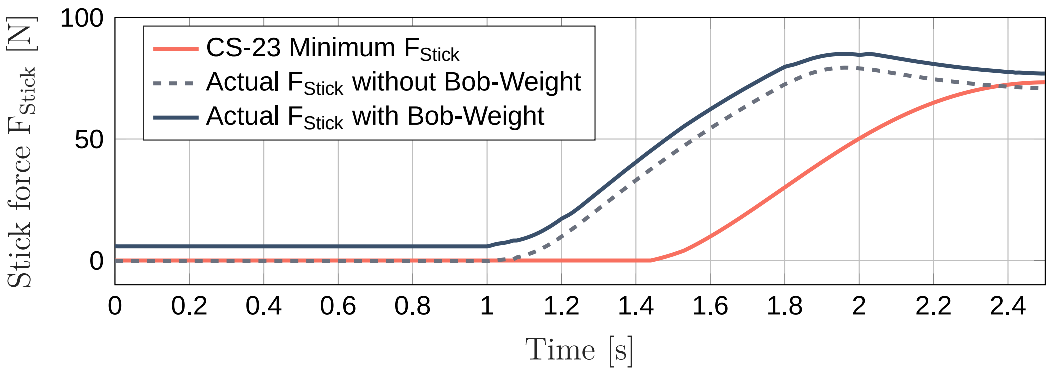


Figure 4.38: *Stick force with bob-weight and minimum stick force over time.*

4.6 Estimation of the Buckling Risk

Column buckling is a subject of continuum mechanics and is, due to its complexity, a separate field of research. EULER's column formula allows one to estimate a critical compression load for columns with homogeneous and isotropic materials, but for more realistic use cases in particular related to FCS with oftentimes anisotropic materials like carbon fibers for rods, the stated relationship reaches its limits. To obtain accurate data, this method is not suitable. However, the critical load, according to EULER, is an initial clue for the risk of buckling. For this reason, it is used in the following simulation to assess the buckling risk for certain rods of the FCS. Therefore, for demonstration purposes, it is assumed that the maximum operational forces occur in this scenario.

The *Simscape Multibody* toolbox offers no functionality to consider buckling. Applying EULER's formula to a model built with this toolbox usually requires a lot of manual work and time. To measure the normal forces, additional *Weld* joint blocks have to be introduced between corresponding parts and joints. The outputs must be logged and further necessary parameters such as Young's modulus, the minimal second moment of area, or the length have to be provided in order to calculate critical loads. The method developed in this thesis, however, allows parts already in the CAD system to be marked for a subsequent analysis. When converting the CAD model into a *Simscape Multibody* model, the required parameters are transferred automatically. They can either be read out from the material data assigned to the parts in the CAD system or be given in the form of user-defined parameters. The additional *Weld* joint blocks are introduced and the data logging is automated as well.

For this purpose, the CAD model is converted again, but the *Simscape Multibody* model creation is triggered with altered parameters. In order to keep simulation costs small, the physical effects friction and backlash in joints are no longer considered, since their influence on buckling is negligible. The FDM still accelerates the entire mechanism and provides the hinge moment. This allows one to not only consider the aerodynamic forces but also the forces caused by inertia.

Apparently, Equation 2.4 contains the relevant parameters for calculating the critical load, which are the second moment of area and Young's modulus. The second moment of area I can be given via user-defined parameters or can be approximated according to Equation 3.26 if the material's density is available. In the following scenario, the material of the rods shall be carbon fiber with a Young's modulus of $E = 8.8 \cdot 10^{10} Pa$. For more precise results, the second moment of area for the hollow cylinders is given using user-defined parameters in *CATIA*. For such cross sections, I is described as

$$I = \frac{\pi}{4}\left(R^4 - r^4\right) \tag{4.26}$$

with R being the outer and r the inner radius. For a cylinder with $R = 10mm$ and $r = 4mm$, the second moment of area is $I = 7.653 \cdot 10^{-9} kgm^2$. The user-defined parameters related to buckling are managed part-wise in the CAD software and are the same for each rod:

```
isBucklingRelevant = 1
I                  = 7.653e-9
E                  = 8.8e+10
```

The critical buckling force F_B is calculated with Equation 2.4 and EULER's second case. Together with the safety factor υ, introduced in Equation 2.5, the maximum allowable compressive force is determined. According to paragraph CS-23.303, the safety factor $\upsilon = 1.5$ is applied to assess the risk of buckling in this use case.

As can be seen in Figure 4.39, the forces are measured twice per part in the built *Simscape Multibody* model using *Weld* joint blocks at both ends of the corresponding rods. The measured total forces (in x-, y-, and z-direction) are then converted to regular *Simulink* signals, which are automatically logged to the *MATLAB* workspace and are finally led in *Terminator* blocks.

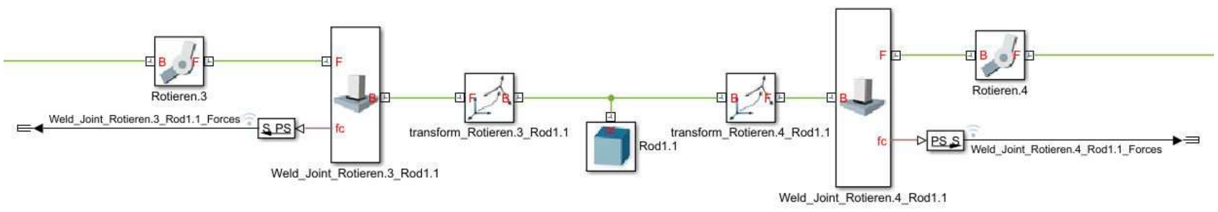

Rotieren.3
B
F
Weld_Joint_Rotieren.3_Rod1.1_Forces
S PS
F
B
fc
Weld_Joint_Rotieren.3_Rod1.1
F
B
transform_Rotieren.3_Rod1.1
Rod1.1
B
F
transform_Rotieren.4_Rod1.1
B
F
fc
Weld_Joint_Rotieren.4_Rod1.1
B
F
Rotieren.4
PS S
Weld_Joint_Rotieren.4_Rod1.1_Forces

Figure 4.39: *Compressive load for each rod.*

The logged data is post-processed in order to obtain the total force by simply applying the *Pythagorean theorem*:

$$F(t) = \sqrt{F_x(t)^2 + F_y(t)^2 + F_z(t)^2}. \tag{4.27}$$

4.6.1 Results

Referring to Figure 4.1, the rods are consecutively numbered starting with the most forward one. The obtained simulation results, which are shown in Figure 4.40, relate to these numbers. As described, the forces are measured twice per part. For this reason, there are two plots, respectively. Naturally, the two blue graphs of each part differ only marginally due to gravitational or inertia forces. The red line, representing the critical load F_B according to Equation 2.4, and the yellow line, representing the maximum allowable load considering the factor of safety, are exactly the same for one and the same part. The results of this particular scenario suggest that most of the rods should not be affected by a risk of buckling, since their compressive load lies well under the yellow line. For Rod6.1, the maximum loads are way above the others, so that they are not shown within the chosen range of the y-axis. After a more precise buckling study, some of these rods, especially Rod2.1 and Rod6.1, may be streamlined to reduce the overall weight. Only Rod3.1, with its long length, exceeds the maximum allowable load slightly, indicating a buckling strength deficit. For further action, this part could be strengthened by increasing its outer radius or by introducing a new lever to reduce the unsupported rod length.

As mentioned at the beginning of this section, such an examination is intended only to give a rough idea of the buckling risk and is by no means suitable to substitute a precise and, of course, necessary buckling strength study.

4.7 Role of the Developed Method in this Use Case

The application of the method developed in this thesis and related advantages regarding the present use case shall be briefly discussed in the following. First and foremost, the method allowed a fully automatic and therefore very fast and flawless conversion of the *CATIA* model to a *Simscape Multibody* model. Subsequently, the bidirectional nature of the developed method allowed the CAD parameters to be optimized in order to minimize nonlinearities in the transfer function of the kinematic between actuator and elevator surface.

Moreover, certain functions developed in this thesis were particularly tailored to mechanical FCSs. For this reason, the nonlinear and complex effects of friction and backlash in revolute joints could be easily and quickly taken into account. The control of these models as well as the parameters required for the simulation of these effects is therefore managed in the CAD system.

The developed method allowed the given FCS to be reviewed in terms of certification aspects and flight safety. Handling qualities were analyzed by accelerating the automatically created model. This way, inertia loads due to flight maneuvers and realistic hinge moments could be considered.

For an estimation of the risk of buckling, the interface provides an automatic calculation of the critical load according to Euler and an automatic determination of the rod loads. The developed feature to read out the necessary material constants from the material assigned to the part as well as the approximation of the parts' second moment of area were not used for accuracy reasons, but were given directly in the form of user-defined parameters in the CAD software.

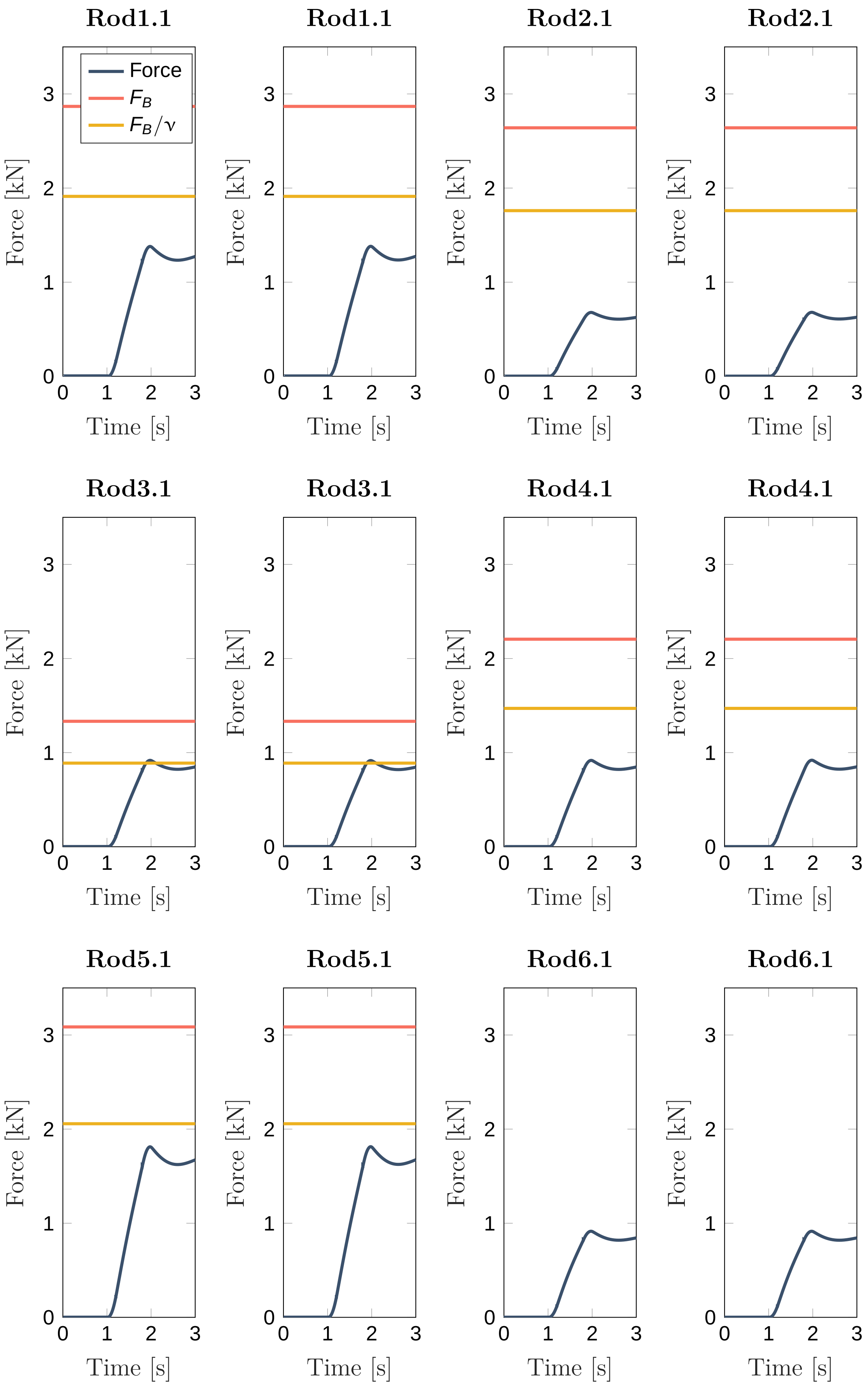


Figure 4.40: *Compressive load for each rod.*

5

Application to a Fly-by-Wire Flight Control System

The use case presented in this chapter shows the application of the methods developed in this thesis within the scope of a project related to a medium-range turboprop airliner. The aircraft manufacturer assigned the institute to perform a model-based requirement validation regarding the yoke control system. The approach of this thesis is applied to automatically create the MBS model, and further models presented previously are integrated to enhance it. Requirements provided by the manufacturer are formalized and test cases verify the simulation model against them. The chapter concludes with a simultaneous optimization of hard- and software parameters while considering formalized requirements.

5.1 Methods and Approach

For development projects of complex systems such as aircraft, requirements management is crucial, as is the quality of the requirements. The classical requirement derivation and validation is a manually driven process. The V-model illustrated in Figure 5.1 is common for the development of safety-critical systems or software. The left side represents capturing the requirements in a top-down manner, where the system is increasingly decomposed downwards. Since the requirements are typically only captured in natural language, the subsequent review has then to be performed manually, as well.

A model-based requirement validation approach, however, uses simulation models to automate the right side of the V-model and support the left side. Therefore, the requirements are not only captured in natural language but are also formalized in order to be able to validate them automatically. Automatic test routines model the requirements and are then able to verify the simulation model against the formalization. The simulation models are therefore not only used for verification but can also be used for the requirement derivation. Requirements at each level can be verified without a specific implementation.

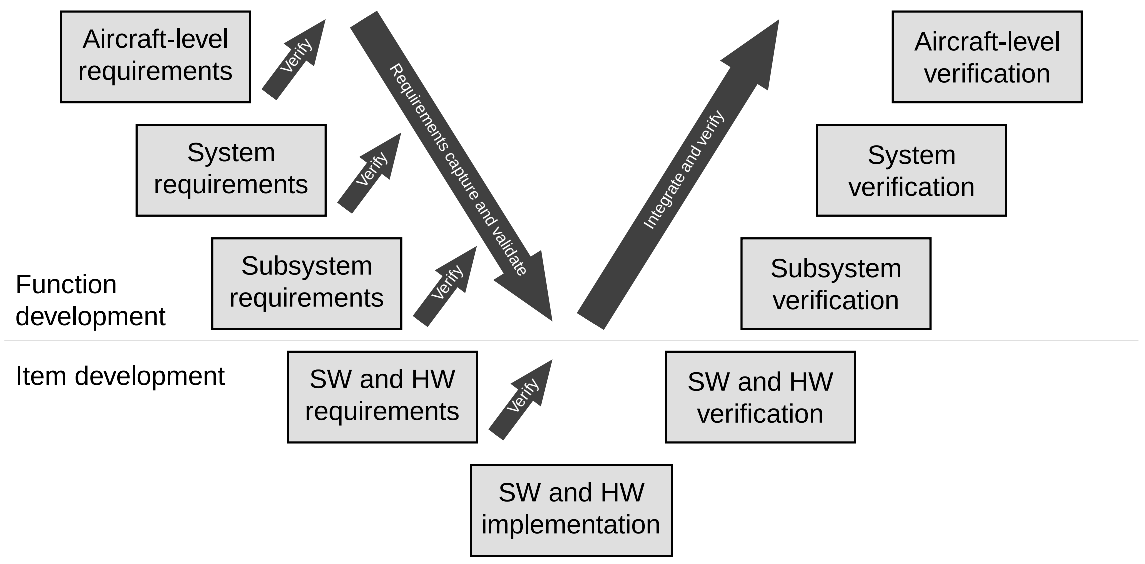


Figure 5.1: *Modified V-model of [Rie13].*

Assuming that the requirements are complete and that the simulation model covers all essential effects, a successful verification of the requirement models corresponds to the validation of the requirements. Another benefit of model-based requirement validation is that, based on the formalization and modeling of the requirements, their verifiability is already ensured during the requirements engineering phase. Requirement verifiability is a common demand in requirements engineering. The incorporation of physical models that are derived automatically from the design and construction world into this established model-based requirement validation process is one of the contributions of this thesis.

In this project, the requirements were provided by the aircraft manufacturer and were captured in the requirements management platform *Polarion*® *REQUIREMENTS*™. Apart from the creation of the simulation model, this use case demonstrates the requirements formalization and the requirements modeling in the form of test cases. Therefore, model inputs and the expected outputs, according to the formalization, are specified in *MATLAB* scripts. A subsequent simulation then produces data that is post-processed to verify the model against the formalized requirements.

In addition, the institute uses a tool called *SimPol*, which manages traceability across the tool borders of *Simulink* and *Polarion*. The tool was developed at the institute and allows one to link requirements and subsystems in *Simulink* models to each other.

5.2 Yoke Control System with Mechanical Control Loading System

The aircraft is to be equipped with a FBW FCS, while the cockpit control inceptor is represented by a yoke control system for two pilots. Due to the irreversible FCS, the yokes are provided with a mechanical *control loading system* (CLS) to supply the pilots with some kind of flight control forces. Essentially, it consists of a spring-damper combination, which produces corresponding counter-forces with increasing deflections and deflection rates.

The yoke of the *Captain* (CPT) and the *First Officer* (FO) are rigidly connected by a friction clutch. This connection can be released with a force exceeding a specified threshold. This ensures the aircraft's controllability in the case that one of both yokes is jammed mechanically. Once the yokes are disconnected, they cannot be reconnected in-flight. A yoke source signal selection system, which is not here considered in detail, manages the signal selection and priority between both yokes.

The trim actuator for the pitch axis is connected to both yokes and alters the zero-force equilibrium point of the spring-damper combinations. In addition to the absolute deflection hard stops, there are CLS hard stops, which move with the trim position. This range is about 20% wider than the absolute hard stops. With the trim in a maximum forward or aft position, the relative hard stops become relevant because in such conditions, the absolute opposite hard stops are no longer reachable. However, if the trim is jammed in the most forward or aft position, safe operation of the aircraft is still guaranteed according to the aircraft manufacturer.

The installed AP actuator is only connected to the CPT yoke and is also used as a stick pusher. The purpose of such a stall identification system is to push the yoke forward whenever the aircraft approaches the stall. Typically, the angle of attack serves as an indicator. The stick pusher is activated as soon as a certain threshold value is reached. Naturally, a linkage only to the CPT yoke results in a loss of autopilot and stall identification if it is jammed.

Analogous to Chapter 4, this use case only focuses on the pitch axis, since the study of the other axis is very similar. The schematic diagram of the mechanics of the pitch axis is illustrated in Figure 5.2. For a clearer visualization, the schematic diagram shows the rotary mechanism in the analogous translational representation.

5.3 Requirements

The requirements to be validated using a model-based approach were provided by the aircraft manufacturer. The relevant ones for the pitch axis are listed below. The requirements were given as translational lengths, which were interpreted as circular arc lengths using the yoke length as the radius. For the relevant requirements, the *MATLAB* code for an automatic requirement validation is listed. For this purpose, the `clsRequirement` class was developed in this thesis. Its implementation can be found in Appendix D. The requirements are assigned a status which can be "passed" (1), "failed" (0), or "no testable condition found" (-1). The used variables originate from the signal logging and contain the corresponding quantity's temporal course.

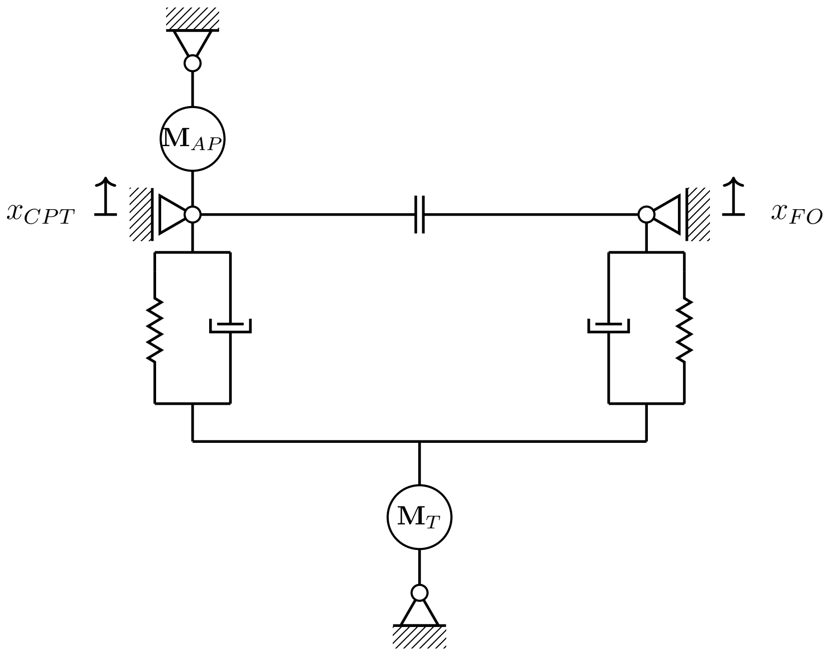


Figure 5.2: *Schematic diagram of the pitch axis mechanics.*

Requirement 1 (REQ-1)

The pitch control shall provide an override function that allows axis control to be kept on one side in case of jamming on the other side.

Requirement 2 (REQ-2)

The pitch control shall mechanically link CPT and FO control.

Requirement 3 (REQ-3)

The column stroke shall be $100mm \pm 1mm$ *in forward and* $-120mm \pm 1mm$ *in aft direction.*

Formalization and Code for Automatic Verification: The following code shows the automatic validation of this requirement. Each yoke is reviewed separately for the upper and lower bound. As such, the boundaries are marginally relaxed by $1mm$, so that minor penetration into the hard stop does not lead to a failed requirement validation.

```
% Create requirement object
objRequirement = clsRequirement();
% Check upper limit for CPT yoke
objRequirement.AddVerification(1, max(x_CPT) <  0.101);
% Check lower limit for CPT yoke
objRequirement.AddVerification(1, min(x_CPT) > -0.121);
% Check upper limit for FO yoke
objRequirement.AddVerification(1, max(x_FO) <  0.101);
% Check lower limit for FO yoke
objRequirement.AddVerification(1, min(x_FO) > -0.121);
```

```
% Evaluate requirement
verification_results(3) = objRequirement.Evaluate();
```

Moreover, the code assumes that, in the test case, a full deflection in both directions was attempted.

Requirement 4 (REQ-4)

The desired breakout force should be $15N \pm 5N$ in the forward and aft directions.

Formalization and Code for Automatic Verification: The idea is to find a point in the simulation results where the pilot force just exceeds $15N$. At this point, the yokes must still be in the trim position. Analogous to REQ-3, the requirement is relaxed by $1.5mm$ mainly to account for deflections due to gravity.

```
% Create requirement object
objRequirement = clsRequirement();
% Check breakout force for positive pilot forces
condition = F_CPT(1:end-1) <=15 & F_CPT(2:end) > 15;
objRequirement.AddVerification(condition, abs(x_wo_trim_CPT(condition)) <
   0.0015);
% Check breakout force for negative pilot forces
condition = F_CPT(1:end-1) >=-15 & F_CPT(2:end) < -15;
objRequirement.AddVerification(condition, abs(x_wo_trim_CPT(condition)) <
   0.0015);
% Evaluate requirement
verification_results(4) = objRequirement.Evaluate();
```

Requirement 5 (REQ-5)

The maximum pitch force shall be $260N \pm 10N$ in the forward direction and $-310N \pm 10N$ in the aft direction.

Formalization and Code for Automatic Verification: With the pilot force as input to the model, it is not trivial to validate the maximum pitch forces. In this case, the maximum forces are validated in both directions through checking the minimum force occurring in the full deflection positions. In addition, the rigid connection between both yokes must still be intact. Otherwise, control loads are halved, since only the remaining CLS of the CPT yoke has an effect. If they are within the required forces, the requirement is validated. Of course, a combination of a certain pitch force course and the inertia of the system may lead to a falsified validation. This must be considered in this particular test case.

```
% Create requirement object
objRequirement = clsRequirement();
% Check nominal maximum pilot force for min forward deflection
```

```
condition = x_CPT >= 0.1 & bDisconnected == 0;
objRequirement.AddVerification(condition, min(F_CPT(condition)) < 260);
% Check nominal minimum pilot force for max aft deflection
condition = x_CPT <= -0.12 & bDisconnected == 0;
objRequirement.AddVerification(condition, max(F_CPT(condition)) > -310);
% Evaluate requirement
verification_results(5) = objRequirement.Evaluate();
```

Requirement 6 (REQ-6)

The pitch trim speed shall be $5.5mm/s \pm 10\%$ *with no load on the column.*

Formalization and Code for Automatic Verification: This requirement is only checked for time steps where the command for the trim system does not equal zero. In order to focus on significant motion, the logged data is first filtered using a second-order transfer function. Afterwards, velocities below $1mm/s$ are removed from the dataset. The requirement is valid if the mean velocity of the remaining data does not deviate by more than $0.5mm/s$ from the target value.

```
% Resample data
t = 0:ts:t_end;
v_trim = interp1(timeVector, v_trim, t);
% Filter data to remove spikes
w = 1000;
G = tf(w, [0.1 2*sqrt(w) w]);
v_trim = lsim(G, v_trim, t);
% Remove slow velocities
v_trim(abs(v_trim)<0.001) = [];
% Create requirement object
objRequirement = clsRequirement();
% Check trim speed for trim position commands other than zero
condition = x_trim_cmd ~= 0;
objRequirement.AddVerification(condition, abs(abs(mean(v_trim)) - 0.0055)
    < 0.0005);
% Evaluate requirement
verification_results(6) = objRequirement.Evaluate();
```

Requirement 7 (REQ-7)

The pitch trim stroke shall be $50mm \pm 1mm$ *in the forward direction and* $-60mm \pm 1mm$ *in the aft direction.*

Formalization and Code for Automatic Verification: Analogous to REQ-3, the requirement is validated as long as the required upper and lower bound is not exceeded.

For reasonable results, this requires the test case to command a full deflection of the trim actuator in both directions. In addition, the requirement is relaxed by $1mm$.

```
% Create requirement object
objRequirement = clsRequirement();
% Check upper limit for trim system
objRequirement.AddVerification(1, max(x_trim) <  0.501);
% Check lower limit for trim system
objRequirement.AddVerification(1, min(x_trim) > -0.601);
% Evaluate requirement
verification_results(7) = objRequirement.Evaluate();
```

Requirement 8 (REQ-8)

The required force to open the friction clutch between both yokes shall be $446N \pm 10N$.

```
% Create requirement object
objRequirement = clsRequirement();
% Check friction clutch dissolving force
condition = abs(F_CPT(1:end-1)) <=446 & abs(F_CPT(2:end)) > 446;
objRequirement.AddVerification(condition,
   ~isempty(bDisconnected(condition) == 1));
% Evaluate requirement
verification_results(8) = objRequirement.Evaluate();
```

Requirement 9 (REQ-9)

When jammed, the maximum pitch force shall be $\leq 500N$.

```
% Create requirement object
objRequirement = clsRequirement();
% Check maximum pitch force if FO yoke is jammed
condition = x_wo_trim_CPT(bDisconnected == 1)>=0.10 |
   x_wo_trim_CPT(bDisconnected == 1)<=-0.12;
objRequirement.AddVerification(condition, ~isempty(abs(F_CPT(condition))
   <= 500));
% Evaluate requirement
verification_results(9) = objRequirement.Evaluate();
```

5.4 CAD Model

The CAD model of this use case is fairly straightforward. One reason for this is that the CLS, with its spring and damper parts, as well as the details of the friction clutch for the rigid connection of both yokes are not represented in the CAD model. However, all other important parts for a pitch axis study, such as both yokes, their connecting parts, the installation points of the trim actuator, and the AP actuator are present. Nevertheless, according to the CAD model information, both yokes are not yet connected to each other and are independently moveable. Only in the MBS model is the rigid connection by the friction clutch established with the implementation of the corresponding logic and behavior. The same applies to the spring-damper combinations for both CLSs.

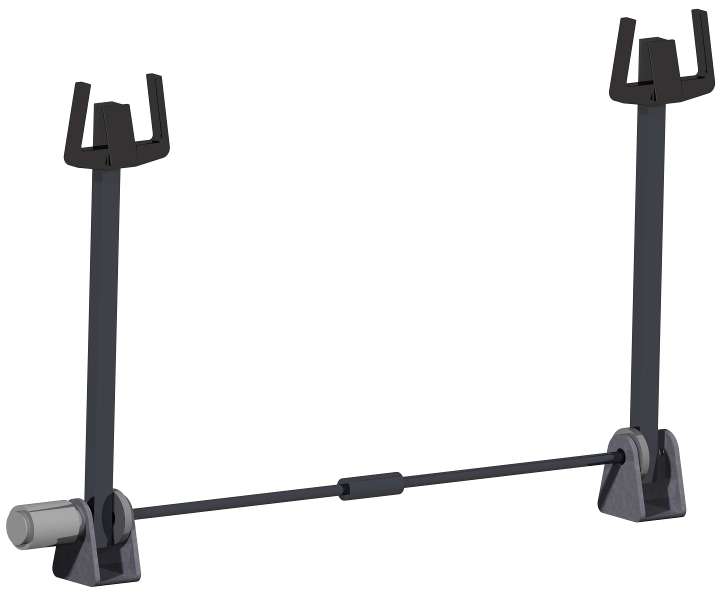


Figure 5.3: *3D CAD model of the yoke system.*

5.5 Simscape Multibody Model

To convert the CAD model to an MBS model, the interface developed in this thesis is used. Afterwards, details like the friction clutch or models for the AP and trim actuator are added to the *Simscape Multibody* model manually.

5.5.1 Nomenclature

The data and requirements provided by the aircraft manufacturer refer to translational movements. However, since the motion is a rotation, which is represented in the simulation model, a relationship between the rotary deflection δ_e and the translational movement x must be made. This is briefly introduced in the following for a better comprehension in later sections. As already mentioned, x is interpreted as the length of the circular arc, which is described by the rotating pilot handle. As such, a positive sign is defined for the push-direction. The relation is then

$$\delta_e = -\frac{x}{l} \tag{5.1}$$

with a yoke length of $l = 0.72m$ between the rotary axis to the pilot handle. The sign change is necessary because rotational deflections in the push direction have negative signs.

5.5.2 Friction Clutch

The force or torque transmission by the friction clutch is modeled as a very stiff spring connecting the rotary DoF of both yokes. With the difference between the rotary deflection of the CPT yoke $\delta_{e,CPT}$ and the FO yoke $\delta_{e,FO}$,

$$\Delta\delta_e = \delta_{e,CPT} - \delta_{e,FO} \tag{5.2}$$

the torque applied by the clutch can be described as

$$T_{clutch} = \begin{cases} \Delta\delta_e \cdot k_{clutch}, & \text{if clutch is closed} \\ 0Nm, & \text{if clutch is open} \end{cases} \tag{5.3}$$

The clutch remains closed until T_{clutch} exceeds a threshold value, which results in opening the clutch. Once it is opened, it stays open until the end of the simulation. The threshold value can be derived from REQ-8 using the yoke length to convert the pilot force to a torque in the clutch. The actual connection between the implementation of the clutch packed in a subsystem and the MBS model is shown in Figure 5.4. Of course, forward dynamics are used to introduce the torque, which is calculated according to the presented equations, into both joints.

5.5.3 Control Loading System

The CLS for each yoke is represented with the help of the internal mechanics, which are available by default for each joint in *Simscape Multibody*. This feature allows one to define a spring stiffness, a damper coefficient, and an equilibrium position for any joint. Accordingly, the two corresponding joints are equipped with a spring-damper combination with spring stiffness k_{CLS} and damper coefficient d_{CLS}. The parameter estimation feature of *Simulink* is used for the estimation of the spring stiffness and the damper coefficient. The measurement data required for this is obtained by a second order transfer function, which was provided by the aircraft manufacturer in addition to the requirements presented in Section 5.3. The transfer function describes the desired relation between introduced force and resulting yoke deflection for the pitch axis. Due to the non-disclosure agreement,

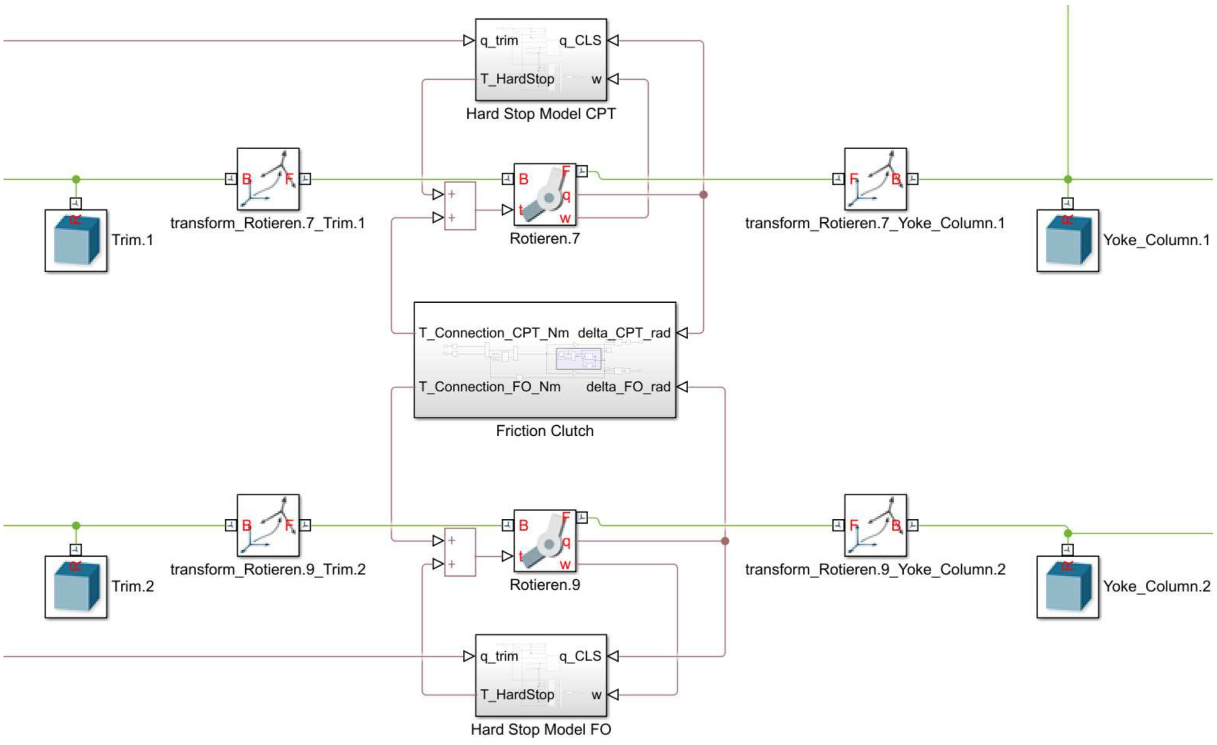


Figure 5.4: *Friction clutch connecting both yokes.*

the actual values are not given:

$$\frac{x}{F_{in}} = \frac{0.339 \cdot b}{s^2\ a + s\ b} \tag{5.4}$$

The time domain response of this transfer function to the step function $H(t)$ (Equation 5.5), applying the maximum pull force at $t = 5$, is plotted next to the response of the final MBS model in Figure 5.5. As can be seen, the estimated parameter lead to a decent fit.

$$H(t) = \begin{cases} 0, & \text{for } t < 5 \\ -310, & \text{for } t \geq 5 \end{cases} \tag{5.5}$$

5.5.4 Breakout Force

According to the specified requirements, the yoke system shall exhibit breakout forces from the trim position, which must be overcome in order to start a movement of the control surfaces. Such means attempt to suppress unintentional commands from the pilot.

In the model, the mechanism to generate a breakout force is not simulated, but its consequences are considered. The breakout force is realized by a *Dead Zone* block, which effectively outputs zero within a specified range. For values outside of this zone, the

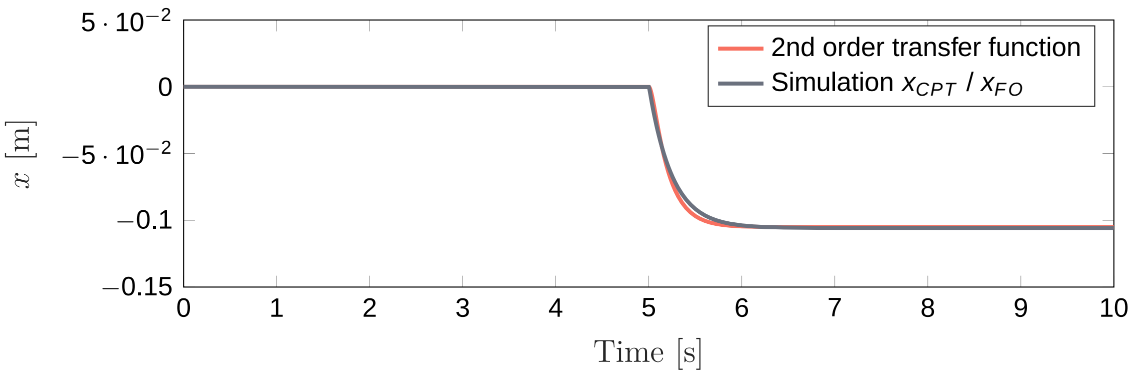


Figure 5.5: *Step response of the transfer function and the simulation.*

output is a linear function of the input.

$$F_{in}(F_{Pilot}) = \begin{cases} F_{Pilot} - F_{bf,min}, & \text{for } F_{Pilot} < F_{bf,min} \\ 0, & \text{for } F_{bf,min} \leq F_{Pilot} \leq F_{bf,max} \\ F_{Pilot} - F_{bf,max}, & \text{for } F_{Pilot} > F_{bf,min} \end{cases} \tag{5.6}$$

The forces exerted by the pilots, which are fed into the model, are thus reduced to forces F_{in} that are perceived by the mechanism. These perceived forces are then applied to the x-axis of the particular yoke via *External Force and Torque* blocks. This means that the applied forces always act perpendicularly to the yoke.

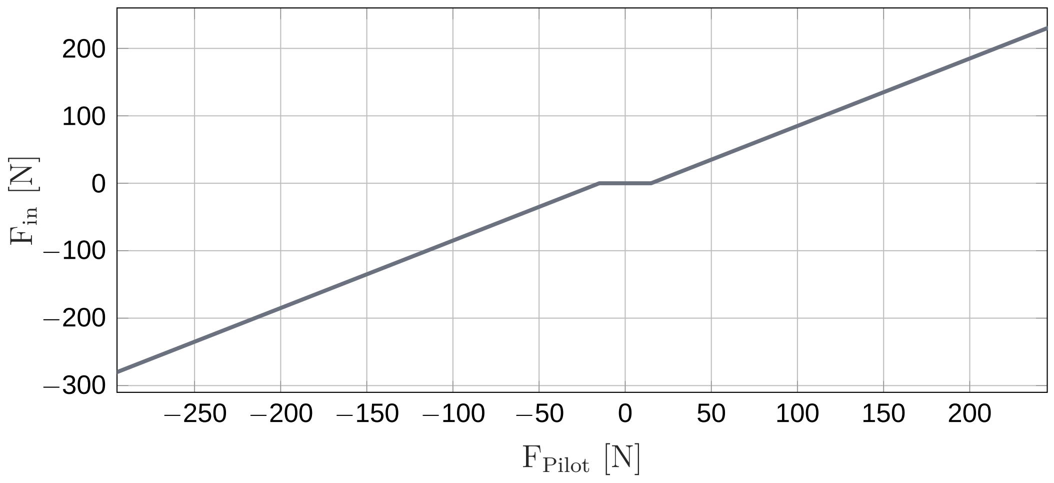


Figure 5.6: *Consideration of the breakout forces via Dead Zone block.*

5.5.5 Actuators

The mechanism represented in the simulation model contains two actuators: one for the trim system and another one for the AP or stick pusher. The actuator models are based on the model presented in Subsection 2.4.1.

5.5.6 Stick Pusher

With regards to the stick pusher actuator model, a simulation of the thermal behavior is neglected, since no data was available in the course of this project, apart from its maximum torque of $253.33Nm$. For this reason, generic data is used for the remaining parameters, in order to reach a corresponding maximum torque (see Table 5.1). Furthermore, the stick pusher actuator operates in a binary fashion, meaning it provides either its maximum torque by commanding the maximum voltage of $28V$ or it is off. In fact, this behavior was specified by the aircraft manufacturer. The forward dynamics actuator model is connected to the corresponding joint, which link the solid of the CPT yoke and the solid of the stick pusher actuator. Naturally, the joint must then be specified as an inverse dynamics problem.

Table 5.1: *Parameters of the stick pusher actuator.*

Name	Symbol	Value
Phase-to-phase inductance	L	$0.55mH$
Phase-to-phase resistance	R	1.05Ω
Voltage constant	k_e	$0.095Vs$
Torque constant	k_t	$0.095Nm/A$
Friction coefficient	k_f	$4 \cdot 10^{-7}Nms$
Moment of inertia	J	$7.5 \cdot 10^{-6}kgm^2$
Gear ratio	G_r	100
Gearbox efficiency	η	1
Viscous friction coefficient	η_v	$1 \cdot 10^{-4}$

5.5.7 Trim System

For the trim actuator, a parameter set is used that satisfies the requirement regarding maximum torque, which was given by the aircraft manufacturer. The parameters are listed in Table 5.2.

In this particular case, the trim actuator model is connected to the two joints connecting each yoke with its fixture. In addition, it is provided with the torque measured in both joints and supplies the resulting trim actuator motion to both. Another modification is the altered gearbox efficiency. Typically, the trim actuator or the gearbox is mostly self-locking, meaning the system cannot be driven by the output side. This is necessary to avoid a continuous load as well as a continuous correction of the position. This is usually achieved by high gearbox ratios and low gearbox efficiencies. In order to account for this effect, the gearbox efficiency η is reduced to 0.1. The maximum rate of the actuator is limited by the controller.

Table 5.2: *Parameters of the trim actuator.*

Name	Symbol	Value
Phase-to-phase inductance	L	$0.55mH$
Phase-to-phase resistance	R_{coil}	1.25Ω
Voltage constant	k_e	$0.055Vs$
Torque constant	k_t	$0.055Nm/A$
Friction coefficient	k_f	$4 \cdot 10^{-7}Nms$
Inertia tensor	J	$12.0 \cdot 10^{-6}kgm^2$
Gear ratio	G_r	100
Gearbox efficiency	η	0.1
Viscous friction coefficient	η_v	$1 \cdot 10^{-4}$

5.5.8 Hard Stops

Contrary to the previous examples, this model requires the modeling of hard stops to consider the limited trim and yoke deflections. Otherwise, due to the nested limits, system states might be reached that are not possible with the real system. In particular, three different hard stops have to be modeled:

- Absolute limits of the yoke deflection (REQ-3)
- Absolute limits of the trim actuator stroke (REQ-7)
- Limits of the CLS relative to the trim position (20% wider than REQ-3)

All hard stops are represented by a spring-damper combination, which is modified based on the model proposed by Hunt and Crossley in [HC75]:

$$T_{HardStop} = k\beta + |k\beta|\, c\dot{\beta} \tag{5.7}$$

with β being the angle exceeding the hard stop limit, defined as

$$\beta = \min\{\delta_e - \delta_{e,min}, 0\} + \max\{\delta_e - \delta_{e,max}, 0\} \tag{5.8}$$

With $\delta_{e,min}$ as the lower (forward) bound and $\delta_{e,max}$ as the upper (aft) bound, it can be seen that the damper term is in this case dependent on the absolute value of the spring force. This results in a continuous contact force, which is both more realistic and more beneficial for the simulation. However, Hunt and Crossley present a complex formula for the damping coefficient which is not used in this case. In addition, only the absolute value of the spring force influences the damper term, since in this use case there are hard stops for negative deflections.

The hard stop models for the CLSs can be directly connected to the corresponding blocks. For the trim system, the model is directly integrated into the actuator model. To limit the absolute deflection of the yokes, the position measurements of the trim and the CLS are required to determine the absolute yoke deflection (see Figure 5.4).

5.5.9 Yoke Jam

The hard stop model just described is also used in this use case to simulate jammed yokes. A boolean signal therefore triggers the forward bound $\delta_{e,min}$ and the rearward bound $\delta_{e,max}$ of the hard stop model to be overwritten with the current yoke deflection at that particular time step. These altered bound values are held until the boolean signal becomes false again.

5.6 Requirements Validation and Discussion

The test cases for the verification of the requirements compliance are grouped into three major groups. Table 5.3 gives an overview of the test cases whose results are discussed in detail in the following. Note that the applied pilot force always refers to the CPT yoke and its x-axis. If relevant, the clutch state is given, which indicates a working connection if it is equal to one or a disconnect if it is equal to zero.

Table 5.3: *Overview of the test cases.*

Test Case No.	Description
Nominal behavior	
10	Nominal achievable yoke deflection
11	Nominal achievable yoke deflection with trim in maximum forward position
12	Nominal achievable yoke deflection with trim in maximum aft position
Jammed FO yoke	
20	FO yoke jammed in neutral position
21	FO yoke jammed in maximum forward trim position
22	FO yoke jammed in maximum aft trim position
Stick pusher activation	
30	Stick pusher activation with neutral trim position
31	Stick pusher activation with maximum aft trim position

5.6.1 Test Case 10: Nominal Achievable Deflection

This test case attempts to validate the nominal achievable yoke deflection (REQ-3) with the trim system in neutral position. Therefore, a pilot force according to the maximum forces of REQ-5 is applied to the CPT yoke. The maximum short term forces according to Paragraph 25.143 of the CS-25 are applied afterwards to verify the resulting deflection. For a more realistic and pilot-like loading, the forces are not introduced as discontinuous steps, but the transition between two force levels always takes $500ms$.

Results and Discussion

As can be seen in the course of x in Figure 5.7, using the maximum pitch forces of REQ-5 does not result in the maximum yoke deflections stated in REQ-3. However, this could

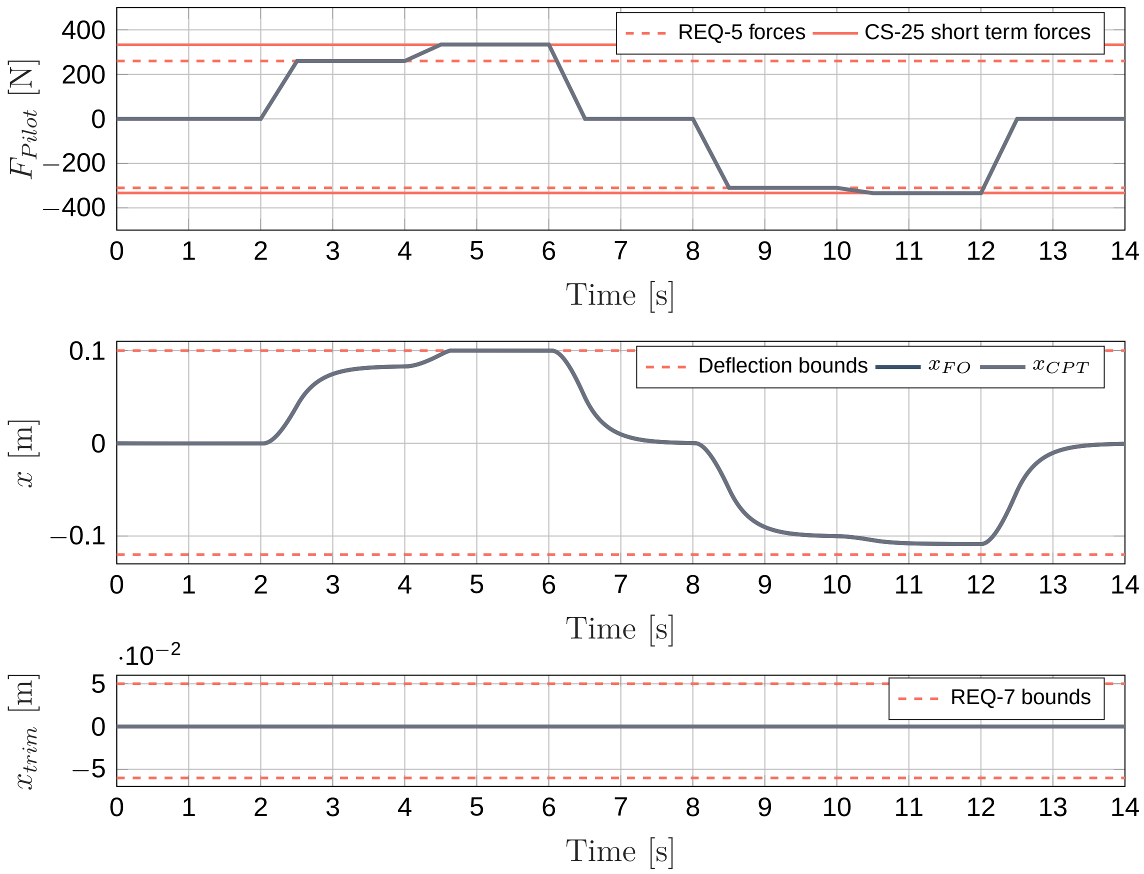


Figure 5.7: *Test Case 10: Nominal achievable yoke deflection.*

be predicted from the given transfer function (Equation 5.4). Even when applying the maximum short term forces of the CS-25, only the forward hard stop cannot be reached while the maximum aft position still needs a higher force. Moreover, with both yokes deflecting, the graph shows an intact connection between them. This is also indicated by the plotted clutch state. Consequently, REQ-2 can be seen as validated.

The results of the automated requirements validation are shown in Table 5.4. They also indirectly indicate a violation of the maximum pitch forces. Although REQ-5 is marked as non-testable, it can be seen as violated since the conditions (maximum deflections) were not found under the maximum forces. The requirements for the trim velocity and for jamming cases are also marked as non-testable, as expected.

Table 5.4: *Results of the automated requirements validation (Test Case 10).*

Requirement	REQ-3	REQ-4	REQ-5	REQ-6	REQ-7	REQ-8	REQ-9
Result	passed	passed	-	-	passed	-	-

5.6.2 Test Case 11: Nominal Achievable Deflection with Max. Forward Trim

In addition to the trim in neutral position, the achievable yoke deflection with the trim in its maximum positions must also be reviewed. For this purpose, the trim actuator is commanded the maximum forward position first. Subsequently, the maximum force of REQ-5 and the maximum short term force of the specifications is applied in the pull direction, to evaluate the reached deflections. In the push direction, only the maximum force of REQ-5 is introduced.

Results and Discussion

With the trim position in maximum forward position, and thus a shortened way to the forward hard stop from REQ-3, it is easily reached with the maximum pitch force. Nevertheless, the rear hard stop is far from being reached, although it is no longer represented by the absolute hard stop but by that of the CLS, which moves forward with the trim position in this case. This can clearly be seen between $t = 6s$ and $t = 10s$. Even the maximum CS-25 short term forces were not sufficient, as there is still a separation of $36mm$ to the lower bound. This suggests possible higher pitch forces when the yoke is deflected to the rear hard stop. This test case scenario also shows the achievable yoke deflections with a pitch trim runaway in the forward direction. The requirement regarding the trim speed (REQ-6) can hardly be manually validated using Figure 5.8. It takes a little less than $10s$ to travel $5mm$, but a more precise assertion can only be done using the raw data. In this context, the automated requirements validation results (Table 5.5) are of particular interest. These show that the pitch trim speed lies between the required limits. As before, the automated requirements validation could not verify REQ-5 because not all test conditions were met.

Table 5.5: *Results of the automated requirements validation (Test Case 11).*

Requirement	REQ-3	REQ-4	REQ-5	REQ-6	REQ-7	REQ-8	REQ-9
Result	passed	passed	-	passed	passed	-	-

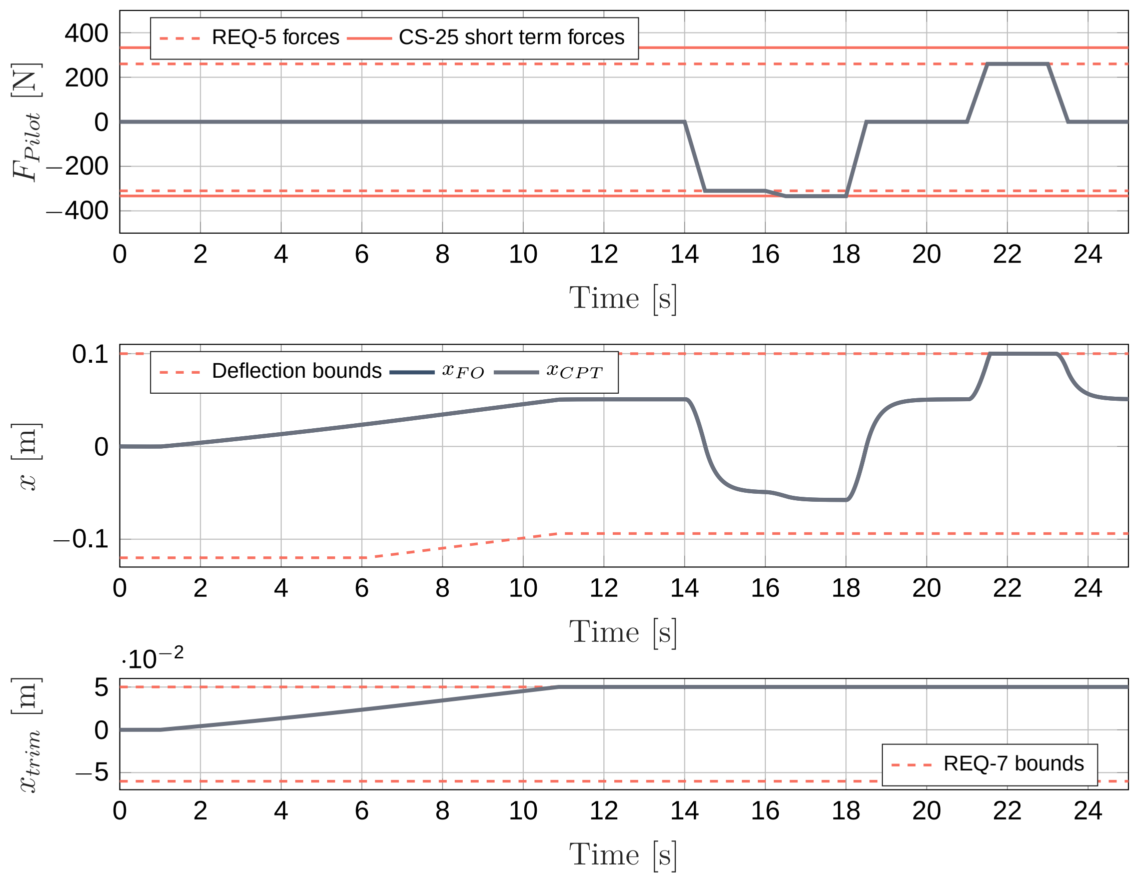


Figure 5.8: *Test Case 11: Nominal achievable yoke deflection with maximum forward trim.*

5.6.3 Test Case 12: Nominal Achievable Deflection with Maximum Aft Trim

Analogous to the previous test case, now the achievable yoke deflections with the trim system in maximum aft position are reviewed.

Results and Discussion

The simulation results are very similar to the ones of Test Case 11. Between $t = 5s$ and $t = 10s$, the reduction of the possible forward stroke can be seen, which is associated to the rearward moving pitch trim. As the plot in Figure 5.9 reveals, the forward hard stop is not reached in this test case, neither by applying the forces of REQ-5 nor by applying the CS-25 forces. However, the deficiency is now only $13mm$. The achievable yoke deflections also apply to a pitch trim runaway scenario in the aft direction. The results of the automatic requirements validation are identical to those of Test Case 11 and are listed again in Table 5.6 for consistency.

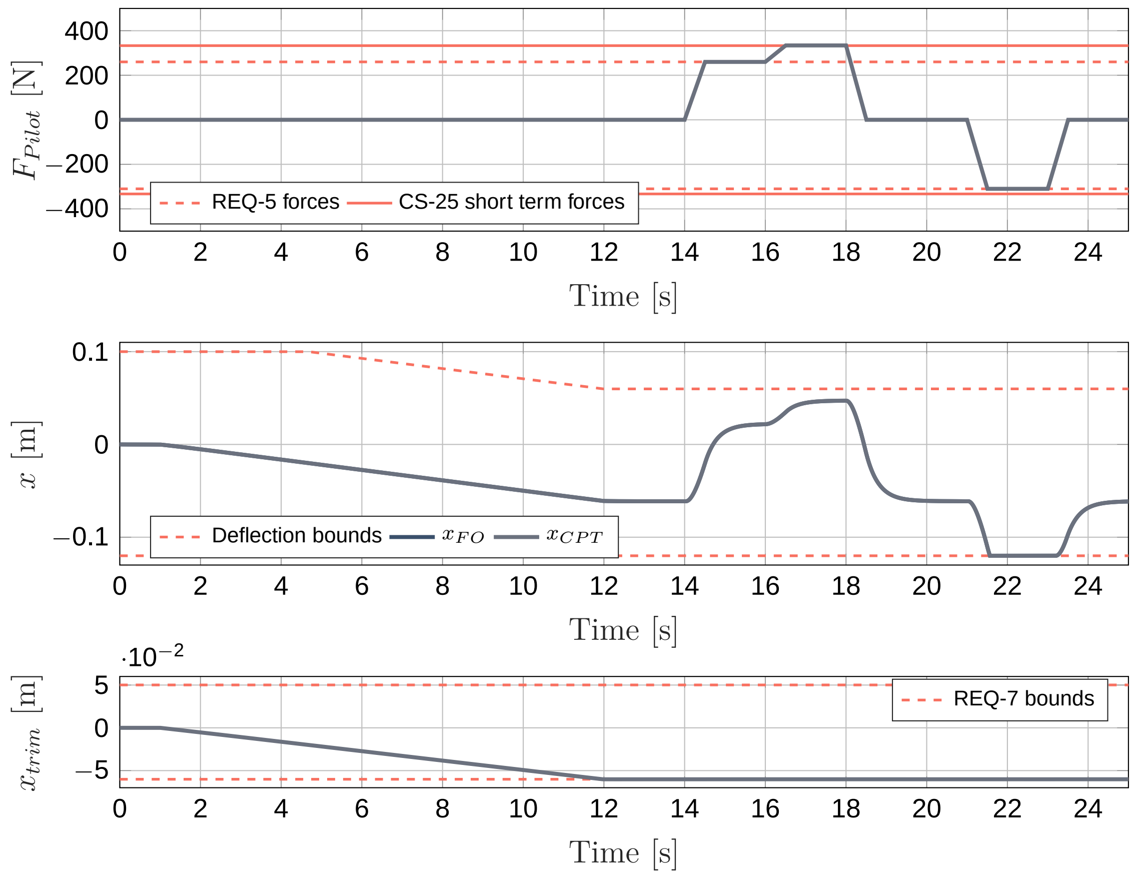


Figure 5.9: *Test case 12: Nominal achievable yoke deflection with maximum aft trim.*

Table 5.6: *Results of the automated requirements validation (Test Case 12).*

Requirement	REQ-3	REQ-4	REQ-5	REQ-6	REQ-7	REQ-8	REQ-9
Result	passed	passed	-	passed	passed	-	-

5.6.4 Test Case 20: FO Yoke Jammed in Neutral Position

The decoupling of both yokes in the case that the FO yoke jams in the neutral position is verified in this test case. Furthermore, the resulting behavior of the operative CPT yoke is reviewed. At the beginning of the simulation, the FO yoke jam is triggered in the neutral position. This jam is held until the end of the simulation.

Results and Discussion

First, the pilot force introduced over the CPT yoke is increased to $500N$, which leads to a release of the mechanical connection, as expected. This can be seen twice in Figure 5.11, namely in the graph representing the clutch state, which drops from 1 (connected) to 0 (disconnected) at around $t = 2.5s$. In addition, the graph displaying the deflection of both yokes reveals a decoupled motion with x_{FO} remaining at zero while x_{CPT} shows a deflection. Furthermore, it is noticeable that the disconnect occurs at the very end of the pilot force rise and not at the beginning. This indicates a validation of the requirement

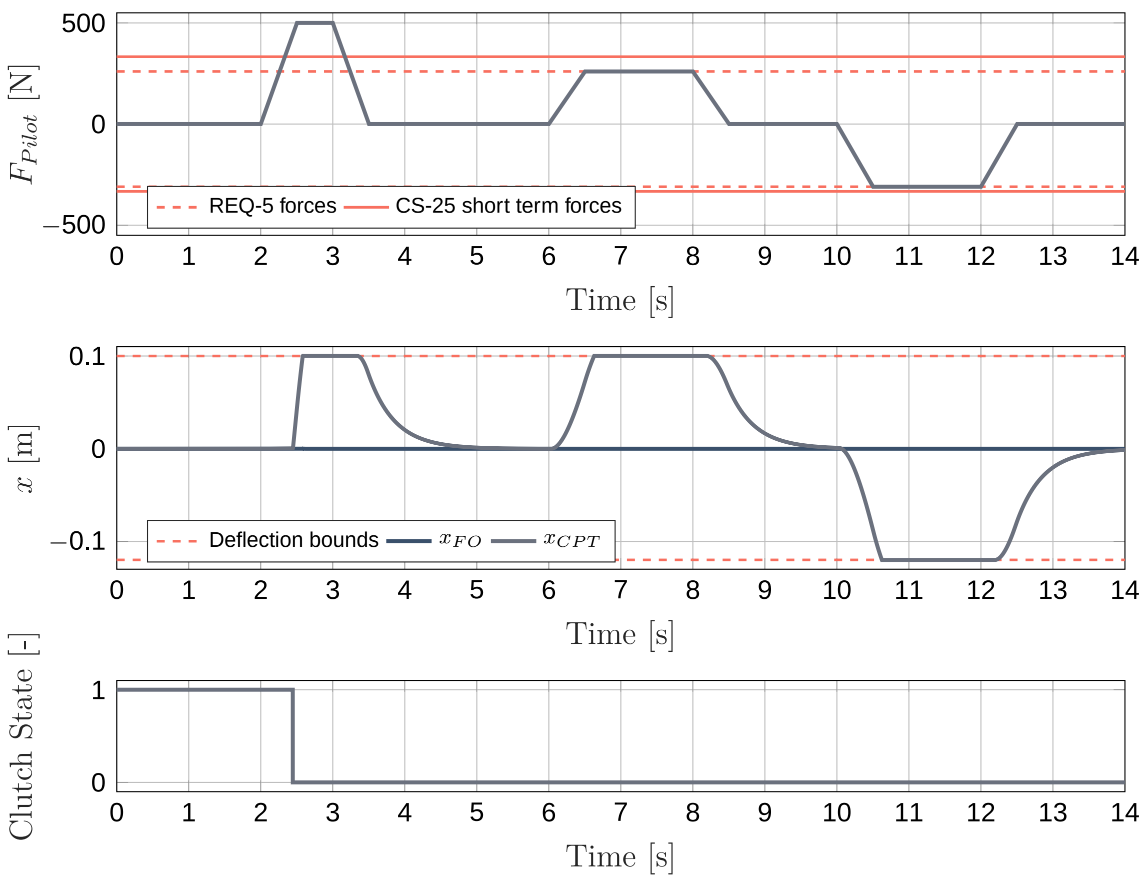


Figure 5.10: *Test Case 20: FO yoke jammed in neutral position.*

specifying the force to open the clutch. Within 130ms after the disconnect, the yoke leaps forward to the hard stop. Taking human factors into account brings the study to an interdisciplinary level and leads to the conclusion that such a behavior can be hazardous. Such short time spans are usually below the human response time when it comes to auditory, visual, or tactile stimuli. Although the time to react to tactile stimuli is faster according to [NC12] compared to the auditory or visual ones, it still takes at least 150ms for the neuromuscular system to react ([LSW54]). Consequently, the pilot hardly has a chance to disconnect both yokes without pushing the yoke forward to its hard stop. Moreover, it is important to consider that always both yokes have to be pushed or pulled as long as it is not obvious which yoke is jammed.

After returning to the neutral position, the maximum pitch forces are applied to the CPT yoke. In contrast to the previous test cases, the hard stops are reached easily this time. The reason for this is the halved loading from the CLS. Due to the disconnect, the CLS from the FO yoke has no effect on the movements of the CPT yoke. The reduced restoring force is also visible when returning to the neutral position, which takes considerably longer than with an intact mechanical connection.

At this point, the automatic requirements validation has verified all but two irrelevant requirements. Since REQ-5 requires connected yokes and REQ-6 requires a trim movement, both requirements are marked as non-testable. In contrast, REQ-8 and REQ-9 could be verified as valid for the first time.

Table 5.7: *Results of the automated requirements validation (Test Case 20).*

Requirement	REQ-3	REQ-4	REQ-5	REQ-6	REQ-7	REQ-8	REQ-9
Result	passed	passed	-	-	passed	passed	passed

5.6.5 Test Case 21: FO Yoke Jammed in Maximum Forward Trim Position

In this test case, the FO yoke is jammed after moving the trim system to its maximum forward position. To disconnect the yokes, the CPT yoke is pushed forwards with $500N$.

Results and Discussion

The simulation results of this test case show that, after the disconnect of both yokes in the most forward trim position, the operative yoke hits the forward hard stop after only $70ms$, which is far below the time a pilot could react. This shortened time span is based on the minimized possible forward deflection of the yoke. Analogous to the previous test case, the yoke acts after the disconnect only against its own CLS, which results in a full aft deflection using the maximum pitch forces of REQ-5. As Table 5.8 shows that, now also the requirement on the trim speed (REQ-6) is verified as valid again. The other results are the same as in Test Case 20.

Table 5.8: *Results of the automated requirements validation (Test Case 21).*

Requirement	REQ-3	REQ-4	REQ-5	REQ-6	REQ-7	REQ-8	REQ-9
Result	passed	passed	-	passed	passed	passed	passed

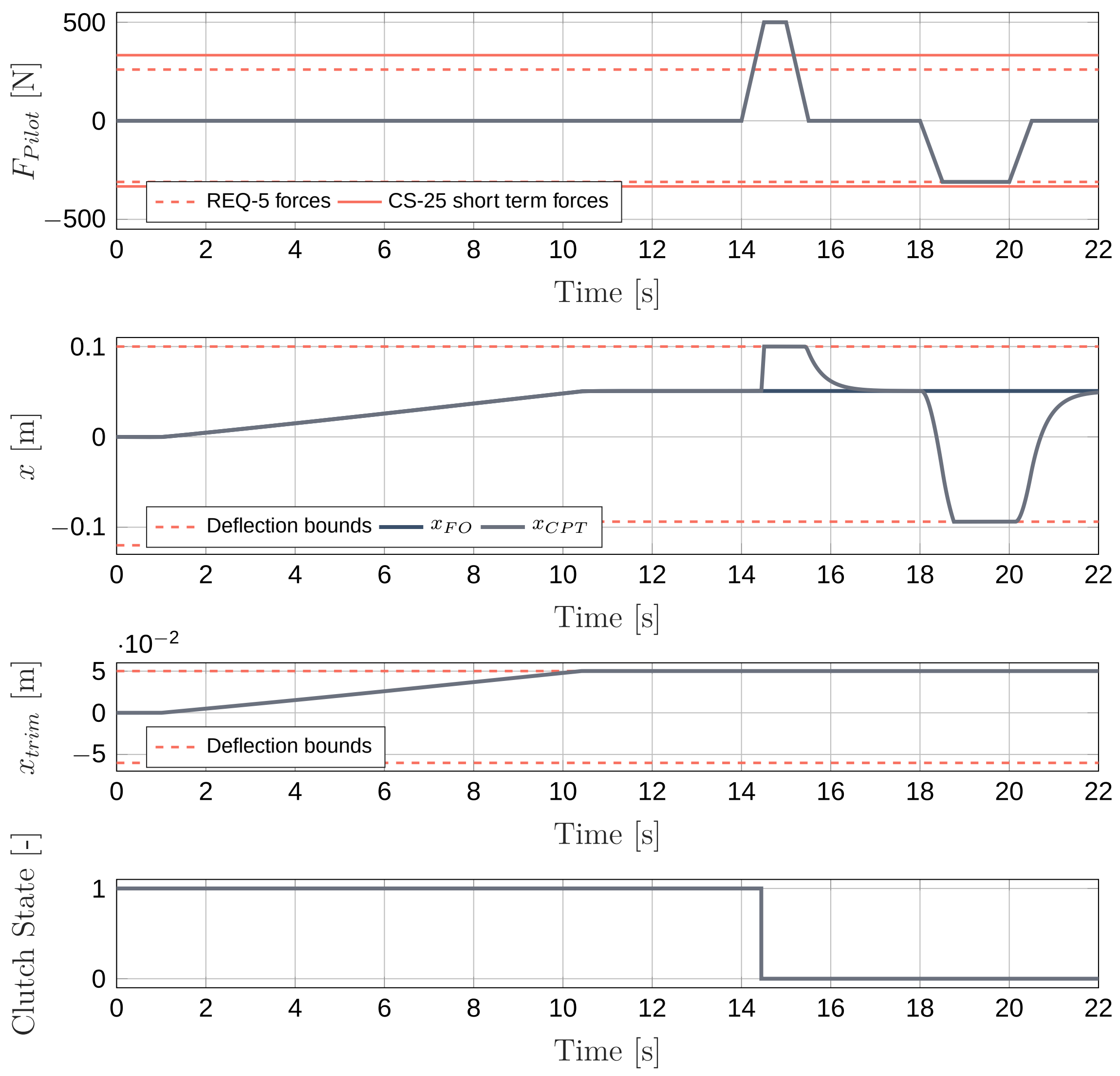


Figure 5.11: *Test Case 21: FO yoke jammed in maximum forward trim position.*

5.6.6 Test Case 22: FO Yoke Jammed in Maximum Aft Trim Position

The subject matter of this test case is a FO yoke jam in the most aft trim position. As before, the disconnect is caused by the forward directed maximum pitch force in a jamming case (REQ-9).

Results and Discussion

The plotted deflection of both yokes in Figure 5.12 shows that, after the disconnect, the CPT yoke also reaches the maximum forward deflection. However, with the greater distance to the hard stop, it takes 170ms, which is 20ms above the minimum response time to tactile stimuli presented in [LSW54]. As long as the prevailing flight state allows it, it can be beneficial to disconnect the yokes by a push or pull force, depending on which hard stop is further away. As in the other two jam test cases, the remaining operative

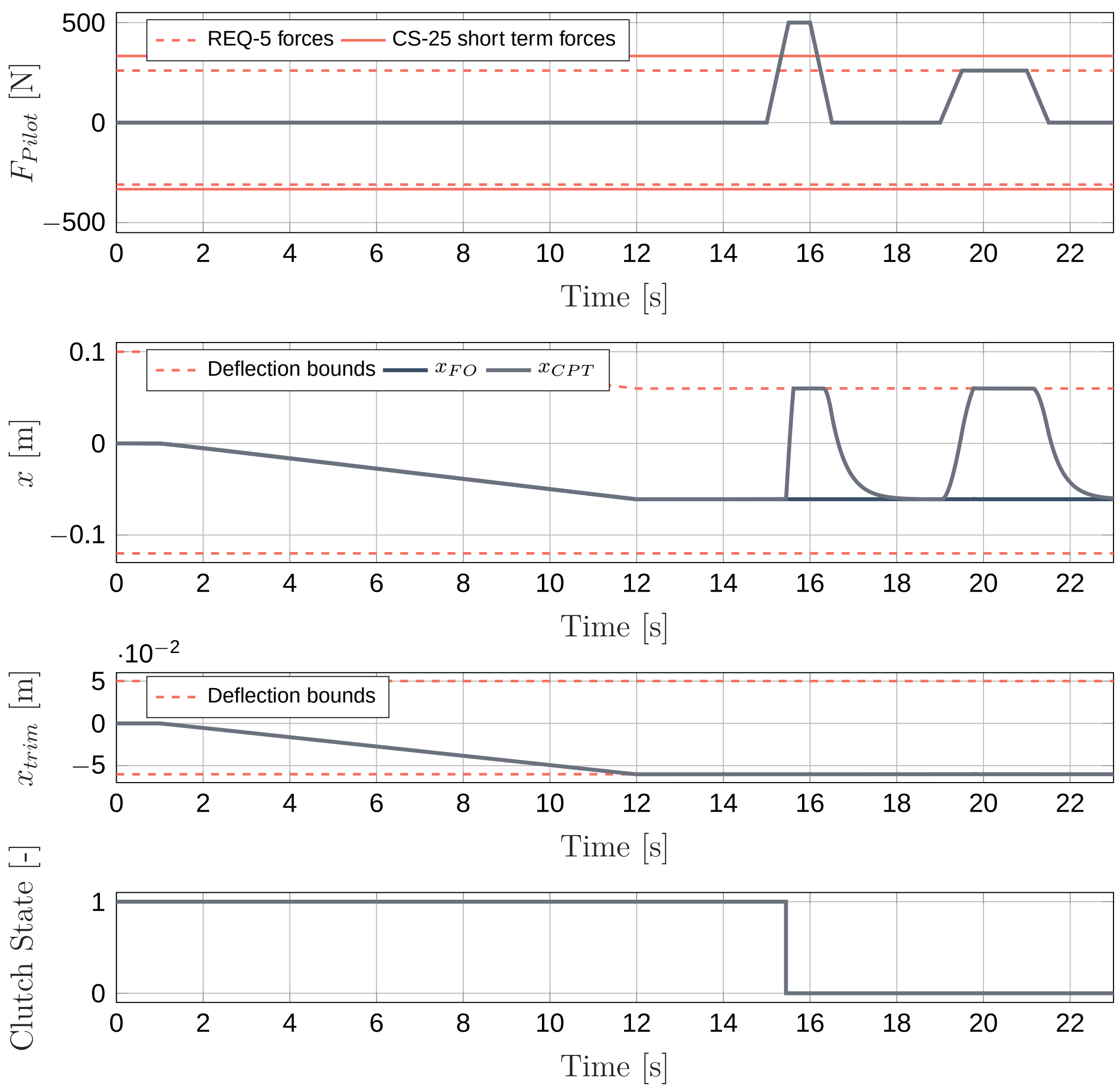


Figure 5.12: *Test Case 22: FO yoke jammed in maximum aft trim position.*

yoke reaches the opposite hard stop using the maximum pitch forces of REQ-3.

Table 5.9: *Results of the automated requirements validation (Test Case 22).*

Requirement	REQ-3	REQ-4	REQ-5	REQ-6	REQ-7	REQ-8	REQ-9
Result	passed	-	-	passed	passed	passed	passed

5.6.7 Test Case 30: Stick Pusher Activation With Neutral Trim Position

This test case focuses on the effect of the stick pusher on the yoke system with the trim system in neutral position. While in the first phase there is no pilot force, the maximum pitch forces of REQ-3 and the maximum maneuver forces of the CS-25 are applied afterwards to review a possible flightcrew interaction. According to [Fed12], it should be unlikely for a flight crew member to prevent oder delay the stick pusher's nose-down control input.

Results and Discussion

The stick pusher is triggered at $t = 1s$, which immediately leads to a full forward deflection of both yokes within half a second. However, the later applied pilot forces restore the yoke

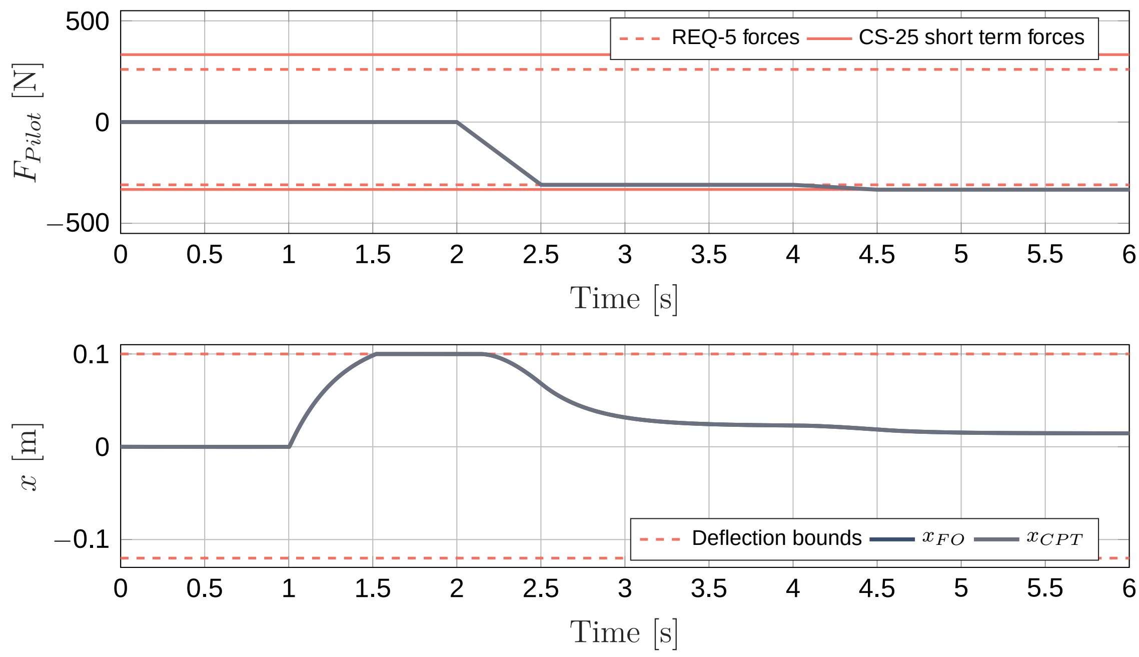


Figure 5.13: *Test Case 30: Stick pusher activation with neutral trim position.*

system close to its neutral position. As stated in [Fed12], such a behavior is undesirable. In this case, most of the stick pusher's torque acts against both CLSs so that only a small amount is available to counteract pilot inputs.

Since most of the requirements of the automated validation are not applicable to the tested scenario, its results are not reviewed.

5.6.8 Test Case 31: Stick Pusher Activation With Maximum Aft Trim Position

Similar to Test Case 30, the stick pusher performance is reviewed in this test case. However, instead of the trim in neutral position, it is now commanded to the aft trim hard stop. Naturally, this leads to an increased possible forward deflection, which has to be overcome by the actuator.

Results and Discussion

After reaching the most rear trim position, the stick pusher is activated at $t = 13s$. As

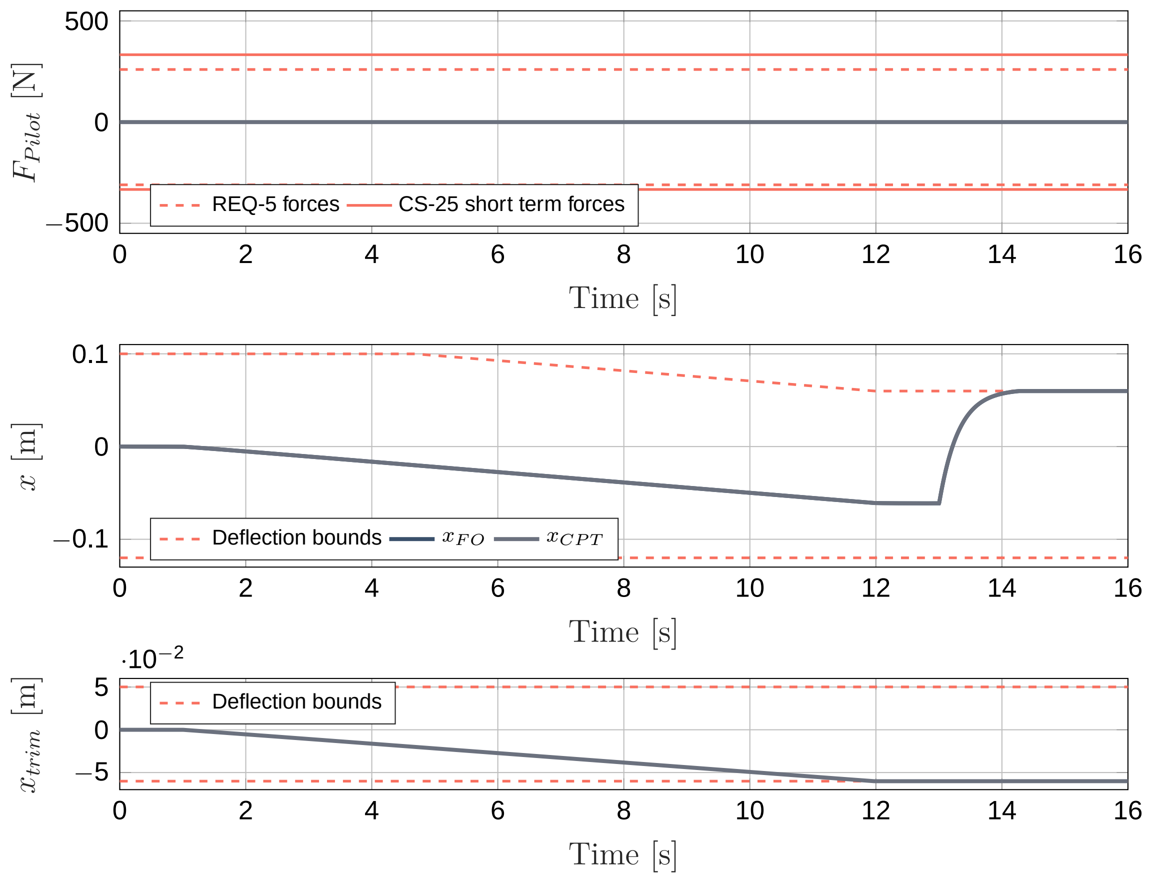


Figure 5.14: *Test Case 31: Stick pusher activation with neutral trim position.*

the deflection plot in Figure 5.14 shows, the actuator only just pushes the yokes to the front hard stop. As such, the flight crew could easily override the stick pusher deflection.

5.6.9 Conclusion

The conducted tests revealed that the system meets all but one requirement. With the trim system in neutral position, the forward or aft hard stops cannot be reached by the application of the maximum pitch forces, which are stated in REQ-5. This deficit will be rectified in the next chapter by the optimization of selected system parameters in accordance with system requirements.

5.7 Simultaneous Optimization of Hard- and Software Parameters

Modern aircraft are very complex and truly multidisciplinary systems, whose development involves countless number of specialized software tools. On account of couplings between domains or disciplines, the optimization of such systems requires the evaluation of the effect of parameter changes in one discipline on the entire system. The incorporation of multiple disciplines into an optimization is known as MDO. Typically, MDO refers in aerospace engineering to different physical domains, such as aero-structural problems, where the best trade-off between aerodynamics and structures is sought. Software parameters, such as controller gains or control strategies, are rarely part of these optimizations. There are methods that consider physical systems and their controllers [AN10, FPU04], but they use either sequential, nested, or distributed approaches of decomposition methods. In this thesis, an integrated architecture is presented that solves the MDO problem using a strategy to simultaneously find the optimal system design. The simultaneous or all-at-once optimization generally provides better results than the sequential strategy, where problems are first decomposed to subproblems and then are optimized sequentially. This often leads to non-optimal system designs due to neglected couplings [FRPU01]. Moreover, none of these approaches considers the integration of a CAD system. For this reason, this thesis presents an approach for a simultaneous optimization of software and hardware parameters, where CAD parameters and the effects of their changes on the overall system are taken into account.

5.7.1 Objective and Approach

In this use case, an optimization problem is formulated that simultaneously involves hard- and software parameters in order to minimize the maximum power consumption of the stick pusher in accordance with the system requirements. On the one hand, energy is somehow a limited resource aboard. On the other hand, flight safety is taken into account, since the stick pusher shall apply a significant force ([Fed12] states 50 to 80 lbs), but it shall be possible to override it in the case of a system fault or of an erroneous trigger signal. A minimization of the required power allows one to limit the voltage source and with that the maximum torque. To prevent overheating and possible damage, this torque must not be greater than the actuator's maximum continuous stall torque. The method developed in this thesis is therefore used to additionally optimize a CAD parameter - the yoke length - and consider resulting mass, inertia, CG, and position and orientation changes in the optimization. For the purpose of optimization, a closed loop i-controller (integral) for the stick pusher with the external load as feedback is added to the simulation model to control the stick pusher on the basis of a reference torque. As such, the gain of the i-controller represents the software parameter to be optimized. Moreover, the developed optimization framework also allows one to include *MATLAB* variables in the optimization. As a means of increasing the potential for improvement, the spring stiffness, and the transmission ratio of the actuator gearbox - both model parameters in the *MATLAB* workspace - are included into the optimization as well. The application of IPOPT, an algorithm for constrained nonlinear optimization, allows one to include the requirements on the system in the optimization. They are implemented in the form of inequality constraints and thus are made visible to the optimizing algorithm. In this use case, only the requirements are verified which are directly affected by the parameter modifications. Therefore, a predefined pilot force $F_{Pilot}(t)$ is applied to the CPT yoke and the stick pusher is activated while the requirements are verified at certain time steps.

5.7.2 Simulation Inputs

The inputs for the simulation model as well as the final time are not dependent on the optimization parameters and thus do not change over the iterations. The simulation starts with a pilot force according to the maximum pitch force requirement (REQ-5) of $260N$ pushing the yokes forward. This allows one to verify that the maximum pitch force results in a full forward deflection (see requirement verification in Subsection 5.7.7). After a short holding period, the pilot force is instantly released so that the yokes return to their neutral position. The same applies for an aft deflection. Afterwards, the stick pusher is activated until the end of the simulation. Both input signals are plotted in Figure 5.15. Significant time steps for the requirement verification are

- the force release at $t_1 = 1.6s$,
- after a zero pilot force time span while the CLS had enough time to center the yoke system at $t_2 = 3s$, and
- after the stick pusher has been active for $1.5s$ at $t_f = 7.5s$.

5.7.3 Parametrization of the CAD Model

For optimization purposes, a length of the yoke system CAD model is parameterized. The parameter l defines the length of the yoke column rotary axis (for pitch commands) to the yoke handle rotary axis (for roll commands). The deflection range is not angular-based but dependent on the yoke handle translation. Consequently, according to Equation 5.1, larger values for l result in smaller maximum deflection angles for a constant maximum translational handle movement, and vice versa.

5.7.4 Stick Pusher Controller

In order to control the stick pusher force-based, a closed-loop i-controller is implemented and added to the model. The control function for the voltage command can be expressed

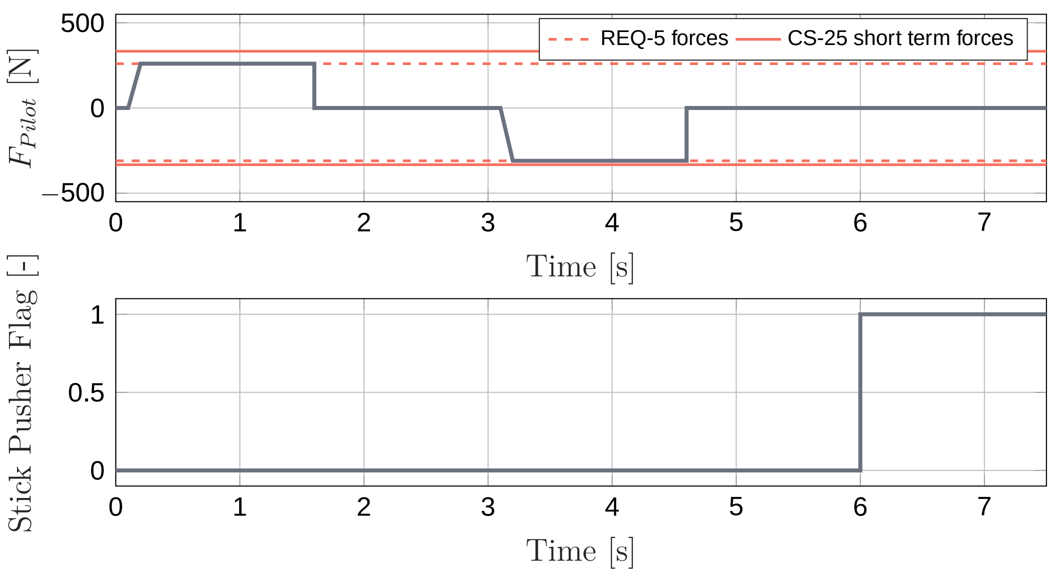


Figure 5.15: *Inputs for the simulation.*

as

$$u_{cmd} = K_i \int_0^t e(\tau)d\tau \tag{5.9}$$

where K_i is the integral gain which is optimized, τ is the variable of the integration, and $e(t)$ is the error from the measured actuator load to the reference. One requirement under which the optimization is performed states that the stick pusher shall generate a power reserve at the yoke handle of $222N$ at the forward hard stop. This means that the controller reference r is constant during the simulation but depends on the yoke length l and the spring stiffness k_{CLS}. It is calculated by

$$r = \underbrace{-2\left(k_{CLS}\frac{0.1\cdot 10^3}{l}\right)}_{\text{Spring forces of the CLSs}} \underbrace{-222\cdot l\cdot 10^{-3}}_{\text{Torque reserve}} \tag{5.10}$$

and represents the minimal torque to overcome the spring forces of both CLSs at the forward hard stop and additionally to build up the specified power reserve. The negative sign is needed for the correct direction of rotation.

5.7.5 Constrained Optimization Problem Setup

The interface of the developed optimization framework allows one to use also optimization tools in addition to the ones that are provided by *MATLAB*. As already stated, in this use case IPOPT is supposed to be used. The constrained optimization problem can be formulated as

$$\begin{aligned} \min_{\mathbf{z}\in\mathbb{R}^n} & J_{Cost}(\mathbf{z}) \\ \text{s.t. } & \mathbf{g}(\mathbf{z}) \leq 0 \end{aligned} \tag{5.11}$$

where $J_{Cost}(\mathbf{z})$ is the cost or objective function and $\mathbf{g}(\mathbf{z})$ represents the inequality constraint. Like *fmincon*, IPOPT is a gradient-based optimizer. Analogous to the previous use case, the gradients cannot be derived analytically but are determined via central finite differences (see Equation 4.5). The vector $\mathbf{z}$, which contains all optimization parameters, is defined as

$$\mathbf{z} = \begin{bmatrix} k_{CLS} \\ l \\ G_{Pusher} \\ K_i \end{bmatrix} \tag{5.12}$$

The spacing, the lower and upper bounds, and the initial values for each parameter are listed in Table 5.10. The limits regarding l are chosen due to the available space and regarding G_{Pusher} with respect to common ratios.

Table 5.10: *Parameters, initial values, limits, and finite differences spacing.*

Parameter	Origin	Initial Value	Lower Bound	Upper Bound	Spacing
k_{CLS}	Simulink	$752.81\frac{Nm}{rad}$	$0\frac{Nm}{rad}$	∞	10^{-3}
l	CAD	$720mm$	$500mm$	$1250mm$	1
G_{Pusher}	Simulink	100	1	200	10^{-3}
K_i	Simulink	1	0	∞	10^{-3}

5.7.6 Cost Function

The objective of this MDO is to minimize the maximum electrical power required by the stick pusher actuator, which can be calculated by

$$P_{Pusher}(t) = u(t) \cdot i(t) \tag{5.13}$$

where $u(t)$ is the voltage and $i(t)$ is the current in the actuator. Consequently, the costs are then calculated by

$$J_{Cost}(\mathbf{z}) = \max_t \left(P_{Pusher}(t, \mathbf{z})\right) \tag{5.14}$$

However, when using gradient-based optimization algorithms, these kinds of of min-max optimization problems are typically reformulated, in order to achieve better results. In this case, the problem is transformed to a TSCHEBYSCHEFF problem formulation by introducing an additional optimization parameter α and inequality constraint

$$\begin{aligned} \max_t \left(P_{Pusher}(t, \mathbf{z})\right) &\leq \alpha \\ J_{Cost}(\mathbf{z}) &= \alpha \end{aligned} \tag{5.15}$$

This definition of the cost function is mathematically equivalent to Equation 5.14. Details on the transformation may be found in [Ger12].

5.7.7 Nonlinear Constraints (Requirements)

As stated, a contribution of this thesis is the simultaneous optimization of hard- and software parameters in accordance with defined system requirements. The parameters listed in Table 5.10 mostly influence the pilot force - deflection relation as well as the stick pusher behavior or performance. The effect on the trim speed, hard-stops, or on the disconnect behavior is negligible. For this reason, the requirements are formulated in the form of nonlinear inequality constraints to ensure that

- the forward hard stop is reached with a $1mm$ tolerance when applying the maximum pilot force of $260N$ in push-direction,
- the spring is strong enough to center the yokes with a $1mm$ tolerance, and
- the stick pusher reaches the forward hard stop and generates a power reserve of $222N$ at the yoke handle within $1.5s$. According to [Fed12], an instantly applied stick pusher force of $222N$ to $355N$ is suggested. This requirement guarantees a constant power reserve of $222N$ at the yoke handle until a full forward deflection.

For the automatic verification, these requirements are first formalized and then implemented in a function that evaluates the nonlinear constraints and is called by the optimization algorithm. The mathematical formulations of the constraints are

$$g_1(\mathbf{z}) = 0.099 - x_{CPT}(t_1) \tag{5.16}$$

$$g_2(\mathbf{z}) = x_{CPT}(t_2) - 0.001 \tag{5.17}$$

$$g_3(\mathbf{z}) = (0.1 - x_{CPT}(t_f)) \cdot 10^4 - \frac{T_{HardStop,CPT}(t_f) + T_{HardStop,FO}(t_f)}{z(2)} \cdot 10^3 + 222 \tag{5.18}$$

where the time points refer to Subsection 5.7.2. The first term of g_3 does not actually represent a requirement but helps the optimizer to determine proper gradients in cases where the stick pusher does not reach the forward hard stop. The additional constraint due to the Tschebyscheff problem formulation is not listed above.

5.7.8 Complete Optimization Problem Statement

The complete optimization problem including the TSCHEBYSCHEFF formulation is summarized in form of a problem statement below:

$$\min_{\alpha,\mathbf{z}} \ \alpha$$
$$\text{s.t. } \mathbf{g}(\mathbf{z},\alpha) \le 0$$

with

$$\begin{aligned}
\mathbf{z} &= \begin{bmatrix} k_{CLS}, & l, & G_{Pusher}, & K_i \end{bmatrix} \\
g_1(\mathbf{z},\alpha) &= 0.099 - x_{CPT}(t_1) \\
g_2(\mathbf{z},\alpha) &= x_{CPT}(t_2) - 0.001 \\
g_3(\mathbf{z},\alpha) &= (0.1 - x_{CPT}(t_f)) \cdot 10^4 \\
&\quad - \frac{T_{HardStop,CPT}(t_f) + T_{HardStop,FO}(t_f)}{l} \cdot 10^3 + 222 \\
g_4(\mathbf{z},\alpha) &= \max_t \left(P_{Pusher}(t,\mathbf{z})\right) - \alpha
\end{aligned}$$

5.7.9 Results

The initial configuration and the converged solution of the simultaneous optimization are presented and discussed in the following.

The original system represented the starting point for the optimization. Since it does not meet the requirement for the maximum pitch forces (REQ-5), this leads consequently to an infeasibility at the beginning. The initial parametrization leads to a maximum power consumption of the stick pusher of $P_{Pusher,max} = 1771.68W$. Admittedly, the power consumption depends strongly on the integral gain, whose initial value is more-or-less randomly chosen. For this reason, an optimization of only K_i was undertaken for a better comparability. This led to $K_i = 0.2887$ and a maximum power consumption of $P_{Pusher,max} = 1536.02W$. This configuration requires a voltage source providing at least $40.16V$.

Table 5.11: *Optimal parameter values.*

Parameter	Initial Value	Lower Bound	Upper Bound	Optimized Value
k_{CLS}	$752.81\frac{Nm}{rad}$	$0\frac{Nm}{rad}$	∞	$1161.62\frac{Nm}{rad}$
l	$720mm$	$500mm$	$1250mm$	$1043.06mm$
G_{Pusher}	100	1	200	200
K_i	1	0	∞	0.1480

The optimal values found by IPOPT are shown in Table 5.11. With the listed parameter set, the stick pusher requires a maximum power of $P_{Pusher,max} = 588.47W$, which is about 65% less than the initial value and roughly 60% less than with only K_i optimized. Moreover, with the optimal parameter set, the required voltage is reduced to only $24.87V$. As can be seen, only the transmission ratio of the stick pusher gearbox ran into its upper bound, while the other parameters found their optimal values between their limits. Naturally, the optimizer aims for at least a local minimum, so that the value found for spring stiffness clearly cannot be realized in the precision listed. Moreover, setting the manufacturing tolerance of the yoke length to less than millimeters is not practical. But even major parameter adjustments still result in a significant improvement compared to the initial system parameterization.

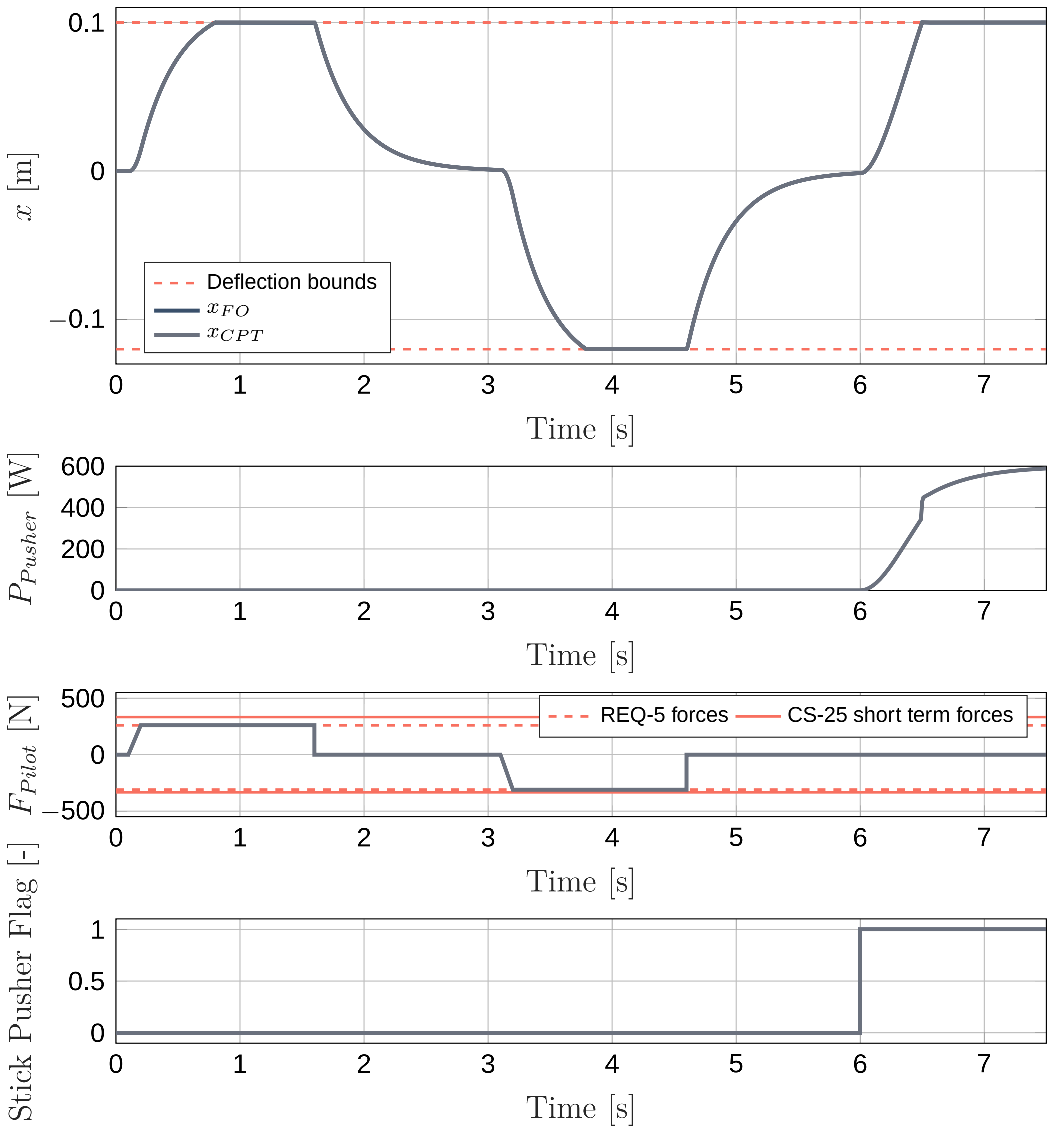


Figure 5.16: *Behavior of the optimized yoke system.*

A solution has been found with a maximum constraint violation of less than 10^{-6}. Therefore it can be concluded that the requirements are still satisfied. Apart from the inputs, the yoke deflection and the electrical power required by the actuator are plotted over the time in Figure 5.16. It is clearly visible that the forward and aft hard stop are now reached when applying the maximum forces of REQ-5. Furthermore, the spring is strong enough to center both yokes within a tolerance of less than $1mm$. The plot of the stick pusher power consumption shows that the actuator is inactive until $t = 6s$. With activation, the stick pusher draws power when acting against the CLS to push the yokes forward. The significantly flattening curve at the end of the simulation indicates a strong reduction of the control deviation and thus the generation of the demanded power reserve.

The method developed in this thesis allowed for the inclusion of hard- and software parameters in *Simulink* as well as CAD parameters in a simultaneous optimization with respect to formalized requirements. As such, the simultaneous strategy considers all parameter couplings and thus enables to find an optimal system design. By means of the inclusion of CAD software, the gap to the design and construction world was bridged and effects of a part modification on the dynamic system behavior could be evaluated.

6 Summary and Outlook

In this thesis, an integrated method for including physical components in the design and optimization of *flight control systems* (FCSs) is presented. The essential part is the development of a method for bidirectional data exchange between CAD software tools and *MATLAB / Simscape Multibody*, which closes the prevalent tool-chain gap between design and simulation. In the following, the most important aspects of each chapter are summarized. Furthermore, a brief outlook for possible future work is given.

The first chapter focuses on the process layer and initially presents a common state-of-the-art development process of mechatronic products. Such processes are very much characterized by tool-chain gaps, causing a lot of unnecessary manual and time-intensive work, sources of error, and costs. As explained in Chapter 1, closing the gap has a substantially positive effect on the development process, leading to shorter development times, more detailed models and, with that, more knowledge about the system in early development phases. As long as a bidirectional data transfer is available, better system design through optimization possibilities across tool-borders can ultimately be achieved. The closure of further tool-chain gaps and the incorporation of disciplines like FEM or CFD may be the subject of future work.

In Chapter 2, typical components of, in particular, mechanical FCSs are introduced. Since the *Simscape Multibody* toolbox does not provide all required components for building high fidelity models, mathematical models are presented, which either represent essential physical effects or entire systems. These models are afterwards used in the application examples and are partly integrated in the developed method for automated usage. Since friction and backlash in joints can have a significant effect on the system behavior, particularly with regard to FCSs, the sophisticated LUGRE friction model is described and a model for simulating backlash in revolute joints is developed.

The solution to the key issue is the subject of Chapter 3, which describes the development of the *MATLAB*-based method for a bidirectional data exchange with CAD tools and the automated building of *Simscape Multibody* models. First, an overview of commercial off-the-shelf interfaces together with a comparison to the approach of this thesis is given, which clearly shows the demand for a more comprehensive method. In

Section 3.3, the requirements for such a method are listed. According to them, the software architecture is specified in Section 3.4. For a most modular approach, a general interface is defined that allows for outsourcing CAD software specific parts to DLLs that are later embedded in the core framework. This ensures an easy integration of further CAD software tools. Proof of concept has already been attained with the help of two bachelor theses which created DLLs for the application of the developed method with *SolidWorks® 2014* (*Dassault Systèmes*) and *NX™ 10.0* (*Siemens*). The implementation of the key methods of the object-orientated framework core in *MATLAB* is covered in Chapter 3.5. The proposed application of quaternions for describing relative orientations and the derivation of synthetic coordinate frames is explained. Moreover, an applied technique from graph theory for the programmatic creation of reasonable models is described. In addition, the inclusion of the friction and backlash model as well as the method for the strength analysis are introduced. In order to enable an optimization of CAD parameters on the basis of the simulation results, an additional framework especially tailored for simulation-based system design optimization is developed and presented in Chapter 3.6. It supports the formulation of nonlinear constrained optimization problems and allows the application of *MATLAB* but also of external optimization algorithms. The use cases in Chapter 4 and Chapter 5 utilize this framework to solve optimization problems, incorporating CAD parameters. Furthermore, strategies implemented for increasing the efficiency of gradient-based optimizations with certain problem formulations are described, such as TSCHEBYSCHEFF, which include CAD parameters and apply finite differences for gradient approximation.

Although this thesis provides a comprehensive approach, there are interesting aspects that can be covered in future work. This includes the integration of an automated derivation of joints based on only constraints data or the automated modeling of flexible bodies. Two methods for implementing the latter are described in [MSVW17]. Moreover, the optimization framework also offers room for advancement. The parallelization of costly simulations and updates of CAD data, or the automated modification of *Simscape Multibody* generated code to make parameters tunable, offer considerable potential to increase the performance of optimization processes.

Chapter 4 presents the first use case, where the method of this thesis is applied to the elevator axis of a mechanical FCS of a general aviation aircraft within the scope of a research project. After describing the mechanism and the parametrization of the 3D CAD model, kinematic analyses are conducted to determine necessary actuator deflections and nonlinearities in the transfer function. The subsequent optimization of CAD parameters, which describe the integration mechanism of the actuator, reduced these nonlinearities by more than half for the same elevator surface deflection. For mechanical analyses, the LUGRE friction model is added to joints and an actuator model takes electrodynamic effects into account. Different combinations of actuator supply voltage and gearbox ratio are examined and discussed for a certain flight maneuver. The most promising configuration is then again enhanced with the inclusion of backlash in joints and with the flight dynamic model of the aircraft, which is used to accelerate the mechanism according to the aircraft motion. This allows for not only assessing the dynamic aircraft behavior, and thus the performance of the entire FCS, but also considering forces caused by inertia during the maneuver. To take into account the handling qualities of the aircraft, the model is used to assess the stick force per g characteristic, which is eventually improved by means of a bob-weight. The chapter concludes with an estimation of the buckling risk of the push-pull rods using the methods developed for strength analysis.

The application of the presented method to the pitch axis of a yoke control system of a commercial fly-by-wire aircraft is demonstrated in Chapter 5. After a short introduction to model-based requirement validation, the yoke control system and its requirements are described. The method of this thesis is then used to automatically generate a *Simscape Multibody* model on the basis of the developed CAD model. Thereafter, models for components that cannot be represented in the CAD software, such as the friction clutch, the spring-damper control loading system, the hard stops, or the breakout forces, are presented and integrated into the multibody simulation model. The requirements validation is eventually achieved by verifying the model against the requirements using test cases, whose results are discussed in detail. Finally, a simultaneous optimization of hard- and software parameters with respect to system requirements in form of nonlinear inequality constraints is performed. The optimal system design found, consisting of CAD, hard- and software parameters, meets the specified requirements and is conclusively described.

The methods presented in this thesis allow the inclusion of physical components in the design and optimization of FCSs and thus contribute to a significant improvement of the development process. The more integral process is not limited to the aerospace industry but can be applied to the development of any mechatronic product. The approach for the simultaneous optimization of hard- and software parameters with respect to system requirements contributes to better designs of mechatronic systems. However, especially the aerospace industry with its long development cycles and the state-of-the-art tools *MATLAB* and *CATIA* can directly benefit from the methods described in this thesis.

Appendix A

Manual

The code of the developed method is available for the employees and students of the Institute of Flight System Dynamics. The following guide shall help to operate the method of this thesis appropriately and gives a first impression how easy the resulting work beyond the tool boundaries is.

A.1 Requirements on the CAD Model

In order to convert a CAD model to a *Simscape Multibody* model using the presented approach, it must be built according to a few rules, of course. This includes the exclusive use of the following joints supported by the method:

- Rigid / weld joint
- Revolute joint
- Prismatic joint
- Cylindrical joint
- Planar joint
- Spherical joint

For further information about the particular DoFs refer to Figure 3.11. This also implies the need to use joints and not only constraints. Regarding *CATIA V5* this means, for instance, that connections between parts have to be created using the *DMU Kinematics Toolbox*.

A.1.1 Limitations Regarding CATIA V5

Since *CATIA V5 R20* is the predominant CAD software tool at the institute, the method was used intensively in this combination and a respective experience has been gained.

Therefore, limitations resulting from restrictions of the *CATIA* API are listed below. There may be similar restrictions for combinations with other CAD systems, but they have not been determined so far.

Weld Joints

If multiple parts are connected via a weld joint, none of these parts must be fixed in space, as otherwise the API stops providing further parts as soon as reaching one of the parts involved in the joint. A possible workaround is to either combine all parts to one single one or to fix all parts in space without using a weld joint.

Spherical Joints

When using spherical joints, the API is not able to obtain the joint position. For this reason, the two points of the individual parts, which are later used for the definition of the congruence condition, must be created in advance using the *Generative Shape Design Toobox*. Defining a spherical joint using these two points allows then to extract the position information.

A.2 Convert a CAD Model to a Simscape Multibody Model

To begin with a model conversion, the path to the CAD model product or assembly file must be known. An object of the type `iCADModel` can then be created using the following command:

```
path    = 'C:\Exemplary_Path\Example_Product.CATProduct';
myModel = iCADModel(path);
```

This establishes a connection to the CAD system, opens the corresponding model, and creates the object in the *MATLAB* workspace. For capturing and transferring the CAD data, the created object provides the `GetCADData` method:

```
myModel.GetCADData();
```

Depending on the model complexity, this process might take a while. Therefore, the progress is indicated with a progress bar. At the end, all required data from the CAD model is extracted, transferred, and stored within the object.

The next step automatically processes this data and builds an equivalent *Simscape Multibody* model. The `CreateSimulinkModel` method of the object triggers the model building:

```
myModel.CreateSimulinkModel();
```

Since the implemented features of the interface may influence the outcome of this step, additional arguments can be passed to the method. The following name-value pairs may be used:

- `'STL' - {true | false (default)}` Enables the automatic export the STL files of each part for a visual 3D representation in the *Mechanics Explorer*. It should be noted, that this slows down the model building process considerably.
- `'JPG' - {true | false (default)}` Enables the automatic photography of each part and a subsequent masking of the particular block. This allows to create clearer diagrams. This feature also slows down the model building process considerably.
- `'Color' - {true (default) | false}` Extracts and transfers colors assigned to parts by default.
- `'Material' - {true (default) | false}` Extracts and transfers data of the materials assigned to parts by default.
- `'Friction' - {true | false (default)}` Determines, if a potential parameterization for friction in joints shall be considered.
- `'Backlash' - {true | false (default)}` Determines, if a potential parameterization for backlash in joints shall be considered.
- `'Buckling' - {true | false (default)}` Determines, if a potential parameterization for strength analyses shall be considered.

All attributes regarding *Solid*, *Joint*, or *Rigid Transform* blocks are parametrized and refer to data of the object of type `iCADModel` in the *MATLAB* workspace.

A.3 Change CAD Parameters from MATLAB

As stated in this thesis, a major feature of the interface is the bidirectionality. In this case, it allows for the modification of user-defined CAD parameters from *MATLAB*.

These parameters are either a member of parts or products. To change a particular parameter value, the location in the `iCADModel` object must be known. For this purpose, the object can be examined in the *Variables* editor. Then, the modification can be undertaken using the `Set` method of parameter objects as follows:

```
% Modification of a parameter located in the root product
myModel.RootProduct.Parameters(2).Set(450);
% Modification of a parameter located in a part
myModel.RootProduct.Elements{3}.Parameters(1).Set(60);
```

Some CAD tools like *CATIA* require an update trigger to make the parameter changes effective. This can be started using the `Update()` method.

```
myModel.Update();
```

The method must not be confused with the following one for obtain an update of CAD data.

A.4 Get an Update of the CAD Data

After changing a parameter either from *MATLAB* or directly in the CAD system, the effective changes can be considered in the model by updating the `iCADModel` object with the `UpdateCADData` method.

```
myModel.UpdateCADData();
```

Similar to the model building, this method allows for passing additional arguments. Primary intention is a time efficient update by skipping unnecessary data, as this method is also used in the optimization framework. The method accepts the following name-value pairs:

- `'STL' - {true | false (default)}` Enables the automatic export the STL files of each part for a visual 3D representation in the *Mechanics Explorer*. It should be noted, that this slows down the model building process considerably.
- `'Color' - {true (default) | false}` Extracts and transfers colors assigned to parts by default.
- `'Material' - {true (default) | false}` Extracts and transfers data of the materials assigned to parts by default.
- `'Parameters' - {true (default) | false}` Extracts and transfers user-defined parameters.

A.5 Model Handling and Reconnect

Since essential data of *Simscape Multibody* models created with the method of this thesis is in the `iCADModel` object in the *MATLAB* workspace, the model and the object must be treated as a couple. This means, if the multibody simulation model is stored for later use, the object containing the data must also be saved in a .mat file, for instance. Otherwise, important data will be lost.

However, the embedding of the DLL and thus the connection to the CAD system is a component of the object which is lost in this process. Consequently, reloading it causes the following warning which can be ignored:

```
Warning:  Cannot load an object of class 'CatiaLinkLibrary':
Its class cannot be found.
```

To reconnect the object with the CAD system and enable an interaction of both tools again, the `Reconnect` method is provided:

```
myModel.Reconnect();
```

A.6 Application of the Optimization Framework

In Capter 3.6, the developed optimization framework is outlined. It facilitates to set-up simulation-based optimizations which incorporate CAD parameters. For this purpose, an object of the type `Problem` must be created, which embeds the previous discussed `iCADModel` object and by that inherits a connection to the CAD software tool:

```
myProblem = Problem(myModel);
```

Afterwards, settings, functions, and parameters for the optimization can be specified. In terms of settings, an algorithm and its particular options must be provided in form of:

```
% Selection of the optimizer
myProblem.Optimizer          = 'fmincon';
% Algorithm-specific settings
myProblem.Settings.Algorithm = optimoptions( ...
                               'fmincon', ...
                               'Algorithm', 'interior-point', ...
                               'Display', 'iter-detailed', ...
                               'SpecifyObjectiveGradient', true, ...
                               'SpecifyConstraintGradient', true, ...
                               'OptimalityTolerance', 1e-6);
% Setting type of gradients
myProblem.Settings.Gradients = 'central';
```

Valid values for the `Optimizer` field are currently

- `fmincon` - for the application of *MATLAB*'s gradient-based fmincon algorithm,
- `ga` - for the application of a genetic algorithm of the *Global Optimization Toolbox*,
- `patternsearch` - for the application of
- `ipopt` - for the application of IPOPT.

The `Gradients` field is only necessary for gradient-based optimization algorithms. At the moment is only supports finite differences for gradient approximation and thus can either be `'forward'` (default) or `'central'`. The objective or cost function and the function for nonlinear constraints are created by the user and then just passed to the framework via a function handle:

```
myProblem.CostFunction                 = @costfun;
myProblem.NonlinearConstraintsFunction = @nonlinconfun;
```

For a proper operation, the definition of these functions must match a certain predefined set of arguments and return values. Passed arguments and expected return values are respectively:

```
function [F] = costfun(z, simulationMeasurements)
...
end

function [c, ceq] = nonlinconfun(z, simulationMeasurements)
...
end
```

The simulation results, in other words the signals logged in *Simulink*, are passed to both functions and are of type `Dataset`.

The optimization parameters are introduced using the `AddParameter` method. It allows to specify the parameter type (*MATLAB* or CAD) and its initial, minimum, and maximum values:

```
% Add MATLAB parameter
myProblem.AddParameter('Type', 'Matlab', ...
                       'VariableName', 'k_cls', ...
                       'Initial', 100,
                       'Min', 0, ...
                       'Max', inf, ...
                       'DiffChange', 1e-6, ...
                       'ScaleFac', 1);
% Add CAD parameter
myProblem.AddParameter('Type', 'CAD',
                       'ParentName', 'Yoke_Column', ...
                       'ParameterName', 'YokeLength', ...
                       'Initial', 500,
                       'Min', 250, ...
                       'Max', 1000, ...
                       'DiffChange', 1e-3, ...
                       'ScaleFac', 1);
```

Depending on the parameter type, a different set of additional arguments is required:

- For *MATLAB* parameter, only the particular name of the variable in the workspace is required and passed in the `VariableName` argument.
- In terms of CAD parameters, two additional arguments are required to find the corresponding parameter object in the `iCADModel` object. Therefore, the name of the parameter itself as well as the instance name of the parent object are needed.

Optional arguments allow to specify the spacing for a finite differences gradient approximation (use `DiffChange`) and to scale the parameter. In this case, all numerical parameter values are then divided by `ScaleFac` before handed over to the optimizer. Naturally, before they are sent to the CAD system or the *MATLAB* workspace, the output of the algorithm is again multiplied by the scaling factor.

After setting up the optimization problem, the algorithm can be started by:

```
[x, fval] = myProblem.Solve
```

Appendix B

Implementation of the Class-Based Framework

The presentation of the framework in Chapter 3 focuses rather the applied methodology than the implementation itself. For this reason the following intends to give an insight into the implementation of the framework. Therefore, each class definition and important methods are presented. However, minor *helper* methods that wrap useful functionality just for internal processes are not included. Furthermore, a description of the `iCADSystem` class is intentionally omitted, since only data is processed and passed to the framework core. For an overview of the interaction of the classes, please refer to Figure 3.7.

iCADModel

Central element of the entire framework is the `iCADModel` class. From a hierarchical point of view, it represents the root object which contains all other objects. At the same time, it is basically the interface to the user and handles the interaction. Therefore, this class is large in scale compared to most of the others.

Objects of this class instantiate respectively an object of the `iSimulinkModel-Builder` (for model building capabilities), `iCADSystem` (for a connection to the CAD system), and `iProduct` (for the representation of the root product) type. The property `Id` is a unique string which helps later to find the object in the *MATLAB* workspace.

The `GetCADData` and the `UpdateCADData` method use functionality of the `CAD-System` object to subsequently transfer data regarding product and parts, joints, constraints, and parameters. Yet to enable an efficient update process, the scope of information is reduced:

- Joint names or associated parts and constraints are not transferred, since CAD parameter changes should not affect the model topology
- Optional: The extraction of color or material data can be suppressed by the user (see Appendix A.4)

- Optional: The repeated transfer of parameter values can be suppressed (see Appendix A.4)

In order to build a proper data structure, the class contains methods that divide the transferred array data into meaningful blocks and create corresponding objects.

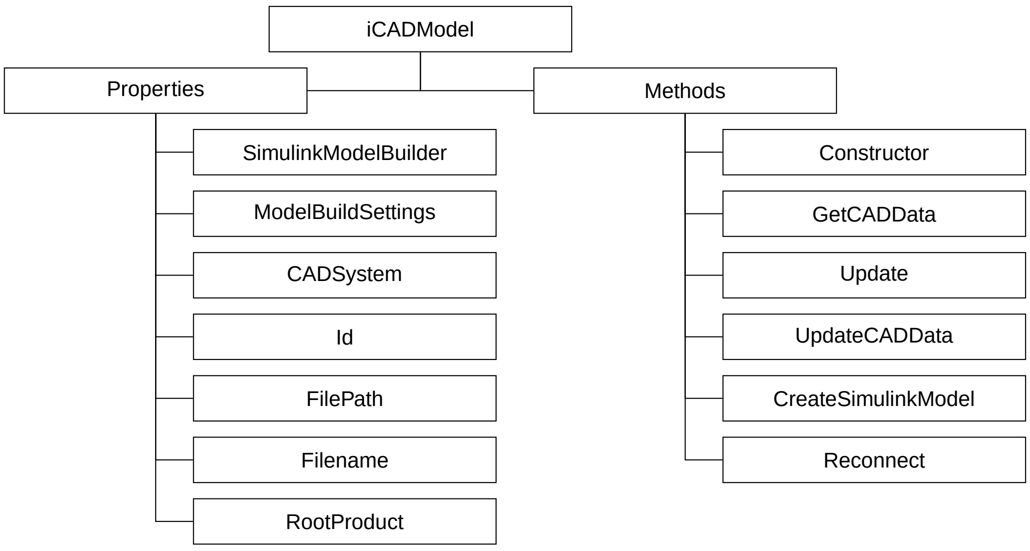


Figure B.1: *Structure of the* `iCADModel` *class.*

iSimulinkModelBuilder

As the name implies, the `iSimulinkModelBuilder` class takes care of the model building. Thereby, the access to the transferred data is granted by the `Parent` property which points to the creating `iCADModel` object.

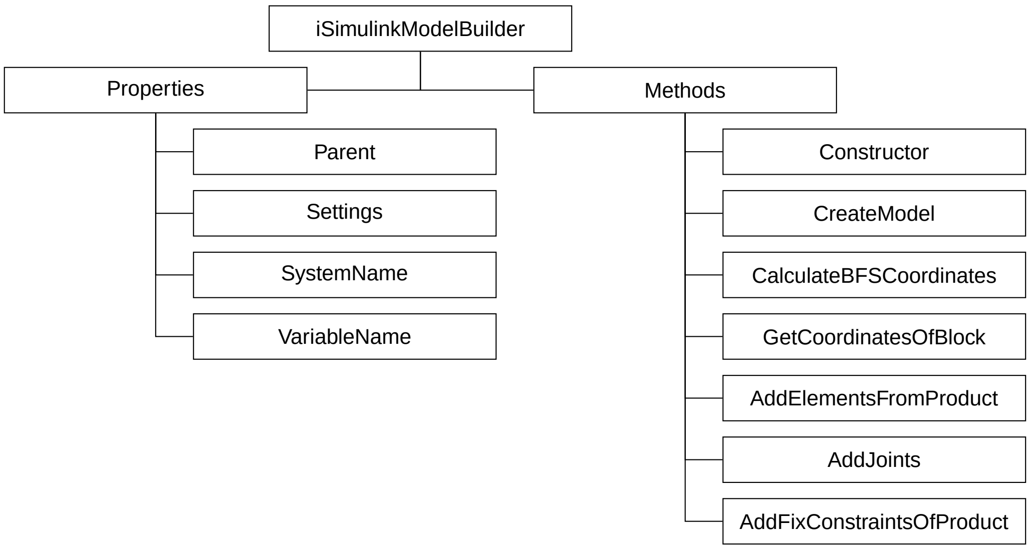


Figure B.2: *Structure of the* `iSimulinkModelBuilder` *class.*

The `SystemName` property is an equivalent but in any case valid *MATLAB* identifier of the root product name of the CAD model. Moreover, the variable name of the associated `iCADModel` object in the *MATLAB* workspace is identified via its unique id and stored in the `VariableName` property. This is used for a proper parametrization of the programmatically added diagram blocks.

Actually, all the listed methods in Figure B.2 are major *helper* methods, with the exception of the `Constructor` and the `CreateModel` method. Before a new model is created, the coordinates for each block in the model are calculated by means of the BFS algorithm (for details, refer to Chapter 3.5.4). After the mandatory set of blocks (*Solver Configuration, Mechanism Configuration, World Frame*) has been prepared, all *Solid* blocks are added to the diagram and configured. The particular coordinates are derived via the `GetCoordinatesOfBlock` method. Afterwards, blocks for representing the corresponding joints and *Rigid Transform* blocks are attached. By processing parts in the first step, all adjacent blocks are available when adding a joint. Therefor, respective parts are already connected to each other by creating *Physical Signals.* In the last step, parts which are fixed in space are linked to the *World Frame* whereby a *Rigid Transform* block with the part's orientation and origin data ensures for a consistent kinematic loop.

iProduct

Products or assemblies are represented using objects of type `iProduct`. The `Root-Product` of the `iCADModel` class, thus the root product of the CAD model, is of this type too.

As Figure B.3 shows, a `iProduct` object saves typical product parameters such as instance name, product name, or filename. In addition, these objects also store all children parts or products in the `Elements` property, which allows for representing a hierarchical data structure. The `Parent` property of `iProduct` and `iPart` (see next section) also enables an upwards navigation in the hierarchy up to the root object. The `CADModel` property allows a direct access.

The `CalculateTransformation` method is used to calculate the quaternion for the relative orientation and the relative translation to the global coordinate frame. It is always triggered when these fields are updated. The results are hold by the `Transform-Data` property.

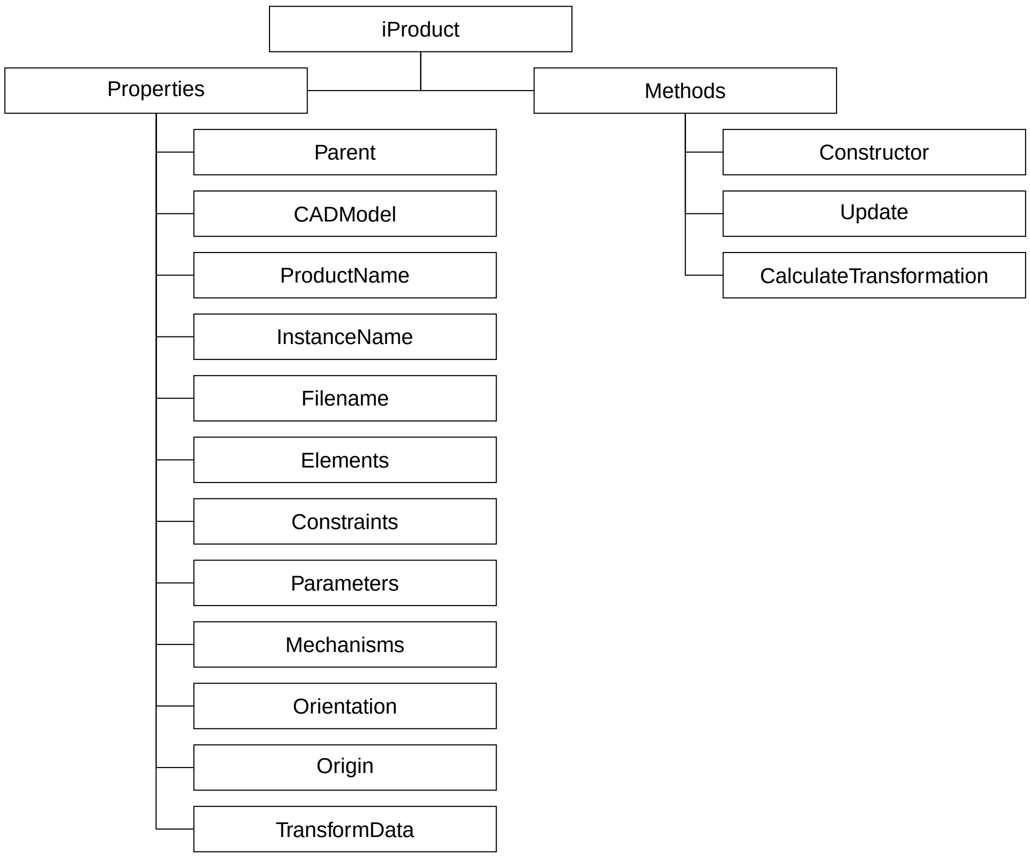


Figure B.3: *Structure of the* `iProduct` *class.*

iPart

Information about parts is stored in objects of the `iPart` class. Instances of this type are always children of `iProduct` objects, thus part of their `Elements` property. Besides general information the `iPart` class also provides physical properties.

When the model builder adds the parts to the model in the form of *Solid* blocks, the parameterization refers to these properties. The `TransformData` property is used for the quaternion and relative translation to the global coordinate frame.

In addition, if a model for strength analysis is created, the `IsBucklingRelevant` method determines, whether the respective part must be considered. The method searches for corresponding parameter names in the `Parameters` property of the object.

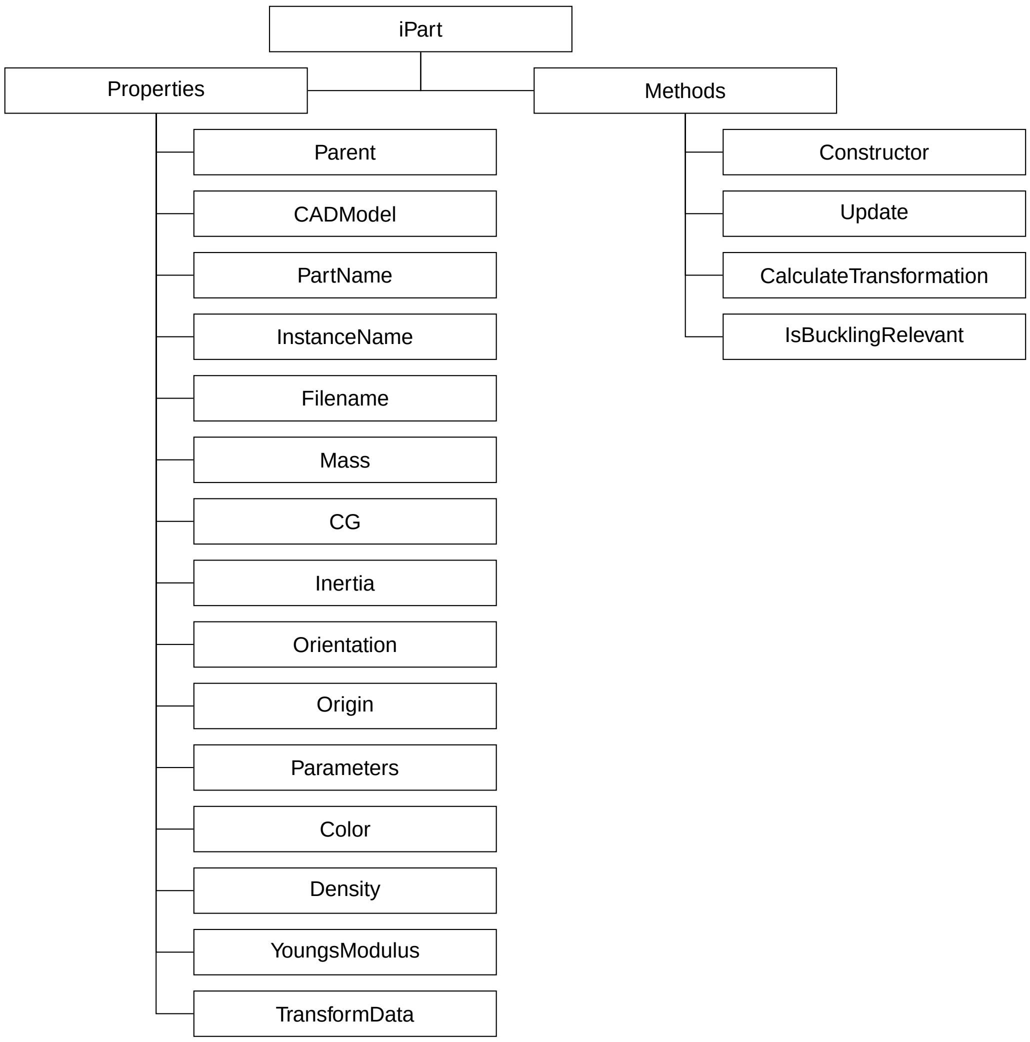


Figure B.4: *Structure of the `iPart` class.*

iConstraint

Constraints basically define the relative motion of two coordinate frames or respectively parts. Since there are no corresponding constraint blocks, constraints can not be added to a *Simscape Multibody* model directly. Yet a combination of constraints or sometimes even one single constraint represents a particular joint. As stated in the concluding chapter of this thesis (Chapter 6), future work may include the determination of joints derived from constraint data.

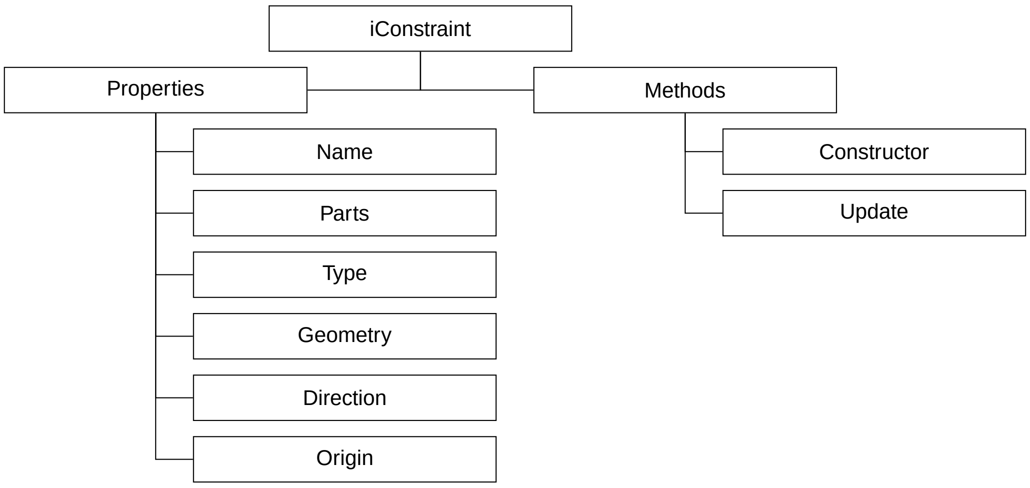


Figure B.5: *Structure of the `iConstraint` class.*

iMechanism

A mechanism basically subsumes all joints of one product. For each product that contains joints, a corresponding mechanism object is created which collects all it's joints. All mechanism objects are stored in the root product and not in the corresponding child products. In the context of this thesis, mechanisms originated from *CATIA*'s tree structure.

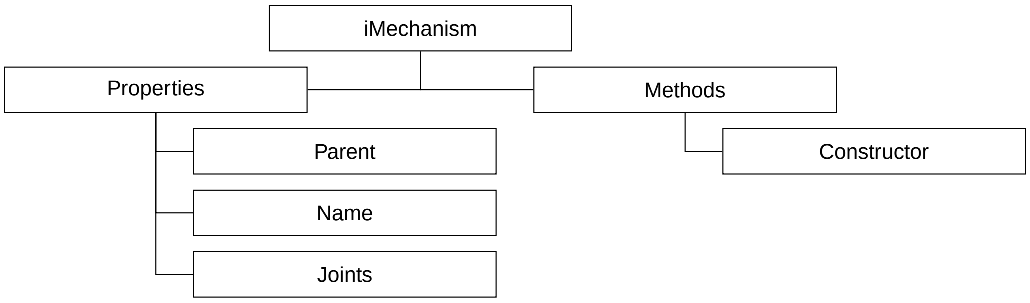


Figure B.6: *Structure of the `iMechanism` class.*

Similiar to constraints, mechanisms can not be added to a *Simscape Multibody* model but the `iMechanism` class is used to structure the CAD data similar to *CATIA*.

iJoint

The aforementioned description of the `iConstraint` class already introduces that joints represent only bundles of constraints. In terms of *Simscape Multibody*, joints are one of the most essential modeling components since they link *Solids* to each other.

The `Type` property provides information about the joint type, thus its DoFs, and refers to Figure 3.11. Consequently, this determines the appropriate block from the comprehensive joints library of *Simscape Multibody*. Under `Parts` and `Constraints`, links to the associated objects are captured. In case of joints, the `CalculateTransformation` method determines the quaternion and relative translation of the joint and the respective parts. Thereby, the data is notated is the corresponding part coordinate frame.

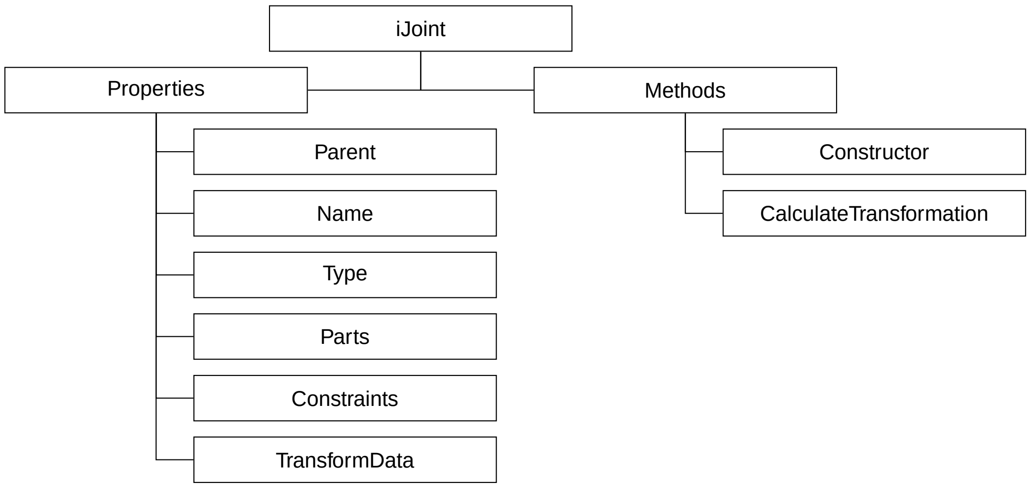


Figure B.7: *Structure of the `iJoint` class.*

iParameter

Any user-defined CAD parameter, whether of parts or products, is represented by an object of the type `iParameter`. The `Parent` property enables access to the parent object which is either of type `iProduct` or `iPart`. Thus access to the CAD software tool is granted by their `CADModel` property which is needed for the `Set` method to change parameter values.

In order to address the correct parameter object in the CAD model, the path from the root product to the parameter is required. It is determined by the `GetProductPath` method once and then hold by the `ParentPath` property.

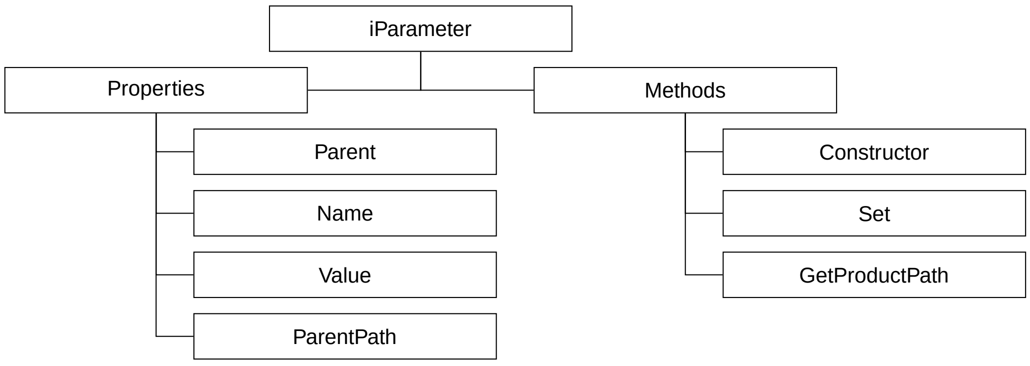


Figure B.8: *Structure of the `iParameter` class.*

iTransformData

Many of the classes presented in the previous sections use the `iTransformData` class to save a quaternion and a translation vector to a corresponding component. Although this data does not necessarily require a class to store it, the `iTransformData` class allows for a better structuring.

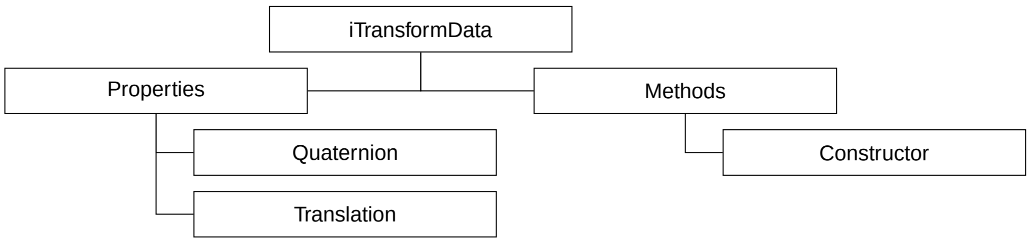


Figure B.9: *Structure of the* `iTransformData` *class.*

Appendix C

Definition of the General Interface

As stated in this thesis, CAD-specific code is outsourced to DLLs in order to obtain a very modular concept. This requires the definition of a general interface in order to standardize the communication with the *MATLAB* framework core. Accordingly, the method structure of each DLL has to match the specification hereinafter.

C.1 Method Overview

The standardized communication requires a minimal set of methods. The following list provides an overview:

- Start
- Close
- OpenFile
- SetVisibility
- Update
- GetParts
- GetConstraints
- GetJoints
- GetParameters
- GetParameterValue
- SetParameterValue
- CreateSTL
- CreateJPEG

C.2 Definition

This chapter specifies each method with its required input arguments and return values.

Start

Establishes a connection to the CAD software and starts it.

Close

Closes the CAD tool.

OpenFile

Opens the passed file.

Syntax

`OpenFile(Filename)`

Input Arguments

- `Filename` - String - Full path (including file name and extension) to the assembly file

SetVisibility

Sets the visibility of the CAD system.

Syntax

`SetVisibility(Visibility)`

Input Arguments

- `Visibility` - Boolean - Full path (including file name and extension) to the assembly file; Default value is true

Update

Updates the CAD model. This is necessary for some CAD tools to make parameter changes effective.

GetParts

Extracts all product an part information from the CAD model and returns data in form of a nested array.

Syntax

`GetParts(ColorReadout, MaterialReadout)`

Input Arguments

- `ColorReadout` - Boolean - Extract color information of parts; Default value is true
- `MaterialReadout` - Boolean - Extract data of materials assigned to parts; Default value is true

Return Values

The data of each component, whether part or product, is respectively stored in an array with two elements. The first element contains all data of the particular component, while the second element contains an array with the information of all children. If there are no children, the element remains empty. The return scheme is illustrated in Figure C.1.

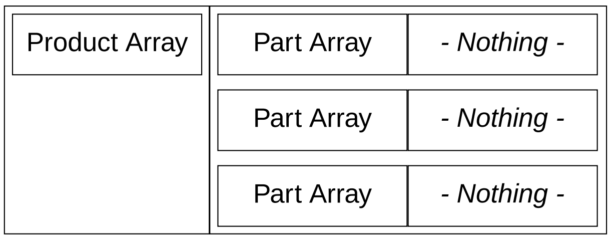


Figure C.1: *Return scheme of hierarchical product and parts data.*

For products, the sequence of the elements is:

- Product name (String): Name of the product
- Instance name (String): Instance name of the product
- Filename (String): Filename of the CAD model (with file extension)
- Orientation (Array(9) of Double, in *mm*): Orientation of the product coordinate frame; Sequence: x-, y-, z-axis; Notated in the global coordinate frame
- Origin (Array(3) of Double, in *mm*) Position of the product coordinate frame; Notated in the global coordinate frame

Data regarding parts is structured as follows:

- Part name (String): Name of the part
- Instance name (String): Instance name of the part
- Filename (String): Filename of the part (with file extension)
- Mass (Double, in kg): Mass of the part
- Center of gravity (Array(3) of Double, in mm): Position of the part's center of gravity; Notated in the part coordinate frame
- Inertia tensor (Array(9) of Double, in $kg \cdot m^2$): Moment of inertia tensor of the part; Sequence is $I_{xx}, I_{xy}, I_{xz}, I_{yx}, I_{yy}, I_{yz}, I_{zx}, I_{zy}, I_{zz}$
- Orientation (Array(9) of Double, in mm): Orientation of the part coordinate frame; Sequence: x-, y-, z-axis; Notated in the global coordinate frame
- Origin (Array(3) of Double, in mm) Position of the part coordinate frame; Notated in the global coordinate frame
- Color (Array(3) of Double): Assigned color normalized to 1; Sequence: red, green, blue
- Material data (Array(2) of Double): First element is the density of the assigned material in $\frac{kg}{m^3}$; Second element if Young's modulus of the assigned material in $\frac{kg}{m \cdot s^2}$

GetConstraints

Extracts all data regarding the constraints of the model.

Return Values

The method returns the constraint data in form of an array where each element itself contains a second array with the particular constraint information, where the scheme is as follows:

- Parent product (Array of String): Top-down path of product names
- Constraint name (String): Name of the constraint
- Constraint type (Integer): Type of the constraint (0: fixed in space, 1: distance, 2: on, 3: concentricity, 99: fixed together)
- Constraint geometry (String): `'plane', 'line'` or `'point'`; for constraint type 99 ″
- Parts (Array of String): Provides the path of instance names to the corresponding parts; For each part there is an element containing the path also in form of an array;

- Direction (Array(3) of Double, in *mm*) If constraint geometry is `plane`: x, y, z of surface normal; If geometry is `line`: x, y, z of direction; If geometry is `point`: 0, 0, 0; In case of no geometry, this element is not provided

- Origin (Array(3) of Double, in *mm*) Position of the constraint; Notated in the global coordinate frame; In case of no geometry, this element is not provided

GetJoints

Provides all mechanisms and joints data in form of a nested array.

Return Values

The return scheme is illustrated in Figure C.2. The upper level contains information about the mechanisms and has the particular joints data nested. The mechanism data is

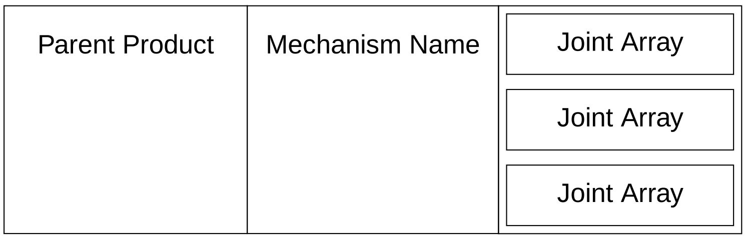


Figure C.2: *Return scheme of mechanisms and joints data.*

given as:

- Parent product (Array of String): Top-down path of product names

- Mechanism name (String): Name of the mechanism

The structure of a joint array is:

- Joint name (String): Name of the joint

- Joint type (Integer): Type of the joint; For rigid joints: 1; For revolute joints: 2; For prismatic joints: 3; For cylindrical joints: 4; For planar surface joints: 5; For spherical joints: 6

- Constraints (Array of String): Contains all associated constraint names

GetParameters

Provides all user-defined parameters of the CAD model.

Return Values

The method returns an array where each element contains a nested array with the parameter information according to the following structure:

- Parent name (String): Name of the parent object
- Parameter name (String): Name of the parameter
- Parameter value (Double): Value of the parameter

GetParameterValue

Reads the current value of the passed parameter.

Syntax

```
GetParameterValue(ParameterName)
GetParameterValue(ParameterName, ParentInstanceName)
```

Input Arguments

- `ParameterName` - String - Name of the parameter
- `ParentInstanceName` - String - Optional - Instance name of the parent object; Can be omitted if parameter is in the root product

Return Values

- Parameter value (Double): Value of the parameter

SetParameterValue

Sets a parameter to an user-defined value.

Syntax

```
SetParameterValue(Path, Name, Value)
```

Input Arguments

- `Path` - String - Top-down path of the instance names to the parameter parent object, separated by a slash
- `Name` - String - Name of the target parameter
- `Value` - Double - Parameter target value

CreateSTL

Exports the STL file for the given part to the same directory.

Syntax

```
CreateSTL(Path, File)
```

Input Arguments

- `Path` - String - Path to the target file (concluding with a slash)
- `File` - String - File name with extension

CreateJPEG

Photographs the given object and save the picture as a jpg file to the same directory.

Syntax

```
CreateJPEG(Path, File)
```

Input Arguments

- `Path` - String - Path to the target file (concluding with a slash)
- `File` - String - File name with extension

Appendix D

Code for the Fly-by-Wire Use Case Requirement Validation

In Chapter 5.3, the requirements of the yoke control system of a FBW aircraft, their formalization, and the corresponding code for an automated verification are presented. For this purpose a class is used which has been developed in this thesis. Its implementation is described below.

```
classdef clsRequirement < handle
    properties
        items = [];
    end

    methods
        function obj = Requirement()
        end

        %% Add verification
        function AddVerification(obj, condition, verification)
            if find(condition > 0)
                % Check if there is a test condition
                obj.items = [obj.items; 1, verification];
            else
                % Mark verification as non-testable
                obj.items = [obj.items; 0, -1];
            end
        end

        %% Evaluate added verifications
        function valid = Evaluate(obj)
            if find(obj.items(:,2) < 0)
                % If verification is non-testable, mark requirement as
                   non-testable
                valid = -1;
            elseif find(obj.items(:,2) == 0)
```

```
                % Else if there is any violation, mark requirement as
                   violated
                valid = 0;
            else
                % Else mark requirement as valid
                valid = 1;
            end
        end
    end
end
```

Appendix E

Publications

The author of this thesis published the following publications during the period at the Institute of Flight System Dynamics at Technical University of Munich. Besides, the author contributed to the lecture notes for the lecture *MATLAB Simulink for Computer Aided Engineering* and to the lecture notes for the corresponding lab course *MATLAB Simulink for Computer Aided Engineering.*

- Schmiechen, Kevin; Hochstrasser, Markus; Rhein, Julian; Schropp, Christopher; Holzapfel, Florian: *"Traceable Model-Based Requirements Derivation, Simulation and Validation with Matlab Simulink and Polarion ALM"* In: 2019 AIAA SciTech Forum. San Diego, USA. 2019

- Nezhat, Sam; Jewoh, Silvain; Schropp, Christopher; Miller, Steve: *"Computergestützte Simulationsschnittstelle - Optimierte Systementwicklung in der Mechatronik"* In: SAXSIM 2017. Chemnitz, Germany. 2017

- Jewoh, Silvain; Nezhat, Sam; Schropp, Christopher; Miller, Steve: *"Optimierte Systementwicklung"* In: VDI Bewegungstechnik 2016. Nürtingen, Germany. 2016

- Schropp, Christopher: *"Development of an Interface between CAD-Systems and SimMechanics"* In: MATLAB EXPO 2014. Munich, Germany. 2014.

Appendix F

Supervised Student Theses

The author of this thesis supervised or co-supervised the following students during the period at the Institute of Flight System Dynamics at Technical University of Munich.

- Gartner, Heitor; *"Entwicklung und Simulation einer alternativen Full-Flight-Simulator Kinematik für VTOL Flugzeuge"*. Garching, Germany. 2018
- Kirsch, Michael; *"Entwicklung einer Simulink Simscape Multibody Erweiterung zum Berechnen und Optimieren von Kontaktkräften anhand von CAD-Daten"*. Garching, Germany. 2018
- Marowsky, Christoph; *"Konzeptstudie zu möglichen Ansteuerungskinematiken für einen Flugsteuerungsroboter"*. Garching, Germany. 2018
- Zhong, Bingzhuo; *"Dynamic Simulation of Contact Forces Based on Model Transformations between Siemens NX and MATLAB"*. Garching, Germany. 2017
- Weinbuch, Moritz; *"Mü 32 R̈eißmeister": Modellierung und Auslegung der Steuerung für ein Segelkunstflugzeug"*. Garching, Germany. 2017
- Kochdumper, Niklas; *"Entwicklung einer Schnittstelle zu SolidWorks in Form einer Dynamic Link Library (DLL)"*. Garching, Germany. 2015
- Su, Jizong; *"Entwicklung einer Schnittstelle zu PTC Creo in Form einer Dynamic Link Library (DLL)"*. Garching, Germany. 2015
- Staudenmaier, Adrian; *"Entwicklung einer Schnittstelle zu Siemens NX in Form einer Dynamic Link Library (DLL)"*. Garching, Germany. 2015
- Nezhat, Sam; *"Entwicklung einer funktionalen Interoperabilität zwischen CATIA V5 und MATLAB/SimMechanics"*. Garching, Germany. 2013

Bibliography

[Air93] Airbus Industrie ; Airbus Industrie (Hrsg.): *Service Bulletin SB A300-22-6021: A300-600.* 1993

[AN10] Allison, James T. ; Nazari, Sam: Combined Plant and Controller Design Using Decomposition-Based Design Optimization and the Minimum Principle. In: *Volume 1: 36th Design Automation Conference, Parts A and B*, ASME, 2010. – ISBN 978–0–7918–4409–0, S. 765–774

[Arn04] Arnold, Martin: *Report / Martin-Luther-Universität Halle-Wittenberg, Fachbereich Mathematik und Informatik.* Bd. No. 2004,27 : Reports of the Institute of Numerical Mathematics: *Simulation algorithms in vehicle system dynamics.* Halle : Univ., Fachbereich Mathematik und Informatik, 2004

[BB15] Böge, Alfred ; Böge, Wolfgang: *Technische Mechanik.* Wiesbaden : Springer Fachmedien Wiesbaden, 2015

[Bet10] Betts, John T.: *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming.* Society for Industrial and Applied Mathematics, 2010

[BFS10] Biahmou, Alain ; Fröhlich, Arnulf ; Stjepandic, Josip: Improving Interoperability in Mechatronic Product Developement, 2010

[Boe14] Boeing 787–8 Critical Systems Review Team: *Boeing 787-8 Design, Certification, and Manufacturing Systems Review.* 2014

[Bos98] Bossak, Maciej A.: Simulation based design. In: *Journal of Materials Processing Technology* 76 (1998), Nr. 1-3, S. 8–11. – ISSN 09240136

[Bos06] Bossche, Dominique van d.: The A380 Flight Control Electrohydrostatic Actuators, Achievements And Lessons Learnt. In: *ICAS 2006 proceedings.* Edinburgh : Optimage, 2006. – ISBN 9780953399178

[BRIA14] Bause, Katharina ; Radimersky, Aline ; Iwanicki, Marinette ; Albers, Albert: Feasibility Studies in the Product Development Process. In: *Procedia CIRP* 21 (2014), S. 473–478. – ISSN 22128271

[BSMC18] Bhattacharya, Partha ; Suyam, Nick-Ange ; Makanaboyina, Ranga ; Chimalakonda, Adithya: Integration of CATIA with Modelica. (2018)

[CADa] CADNexus Inc.: *CAPRI CAE Gateway Overview*. `http://www.cadnexus.com/index.php/capri.html`

[CADb] CADNexus Inc.: *Visualize / Analyze / Optimize your CAD designs in MATLAB*. `http://www.cadnexus.com/index.php/matlab-connector.html`

[CE08] Cooper, R. G. ; Edgett, S. J.: Maximizing Productivity in Product Innovation. In: *Research Technology Management* (2008), S. 47–58

[CH05] Clark, John ; Holton, Derek A.: *A first look at graph theory*. Repr. Singapore : World Scientific, 2005

[CJ06] Chang, Kuang-Hua ; Joo, Sung-Hwan: Design parameterization and tool integration for CAD-based mechanism optimization. In: *Advances in Engineering Software* 37 (2006), Nr. 12, S. 779–796. – ISSN 09659978

[COAL95] Canudas de Witt, Carlos ; Olsson, Henrik ; Aström, Karl J. ; Lischinsky, Pablo: A new model for control of systems with friction. In: *IEEE Transactions on Automatic Control* (1995). – ISSN 0018–9286

[Coo69] Cooper, G.E. and Harper. R.P: *The Use of Pilot Rating in the Evaluation of Aircraft Handling Qualities*. 1969

[Das] Dassault Systèmes: *CATIA V5: Service Pack 7, Build Number 20, Hot Fix 54*

[DWH03] Deremaux, Yann ; Willcox, Karen ; Haimes, Robert: Physically-Based, Real-Time Visualization and Constraint Analysis in Multidisciplinary Design Optimization. In: *33rd AIAA Fluid Dynamics Conference and Exhibit*. Reston, Va. : American Institute of Aeronautics and Astronautics, 2003. – ISBN 978–1–62410–095–6

[Dzo08] Dzomo, Prudence C.: *Modellierung eines Mapping in einer multidisziplinären Umgebung zur effizienten Cross-Skill Engineering Kollaboration*. Darmstadt, 2008

[EBPF03] Engelson, Vadim ; Bunus, Peter ; Popescu, Lucian ; Fritzson, Peter: *Mechanical CAD with Multibody Dynamic Analysis Based on Modelica Simulation*. 2003

[Eur15] EUROPEAN AVIATION SAFETY AGENCY: *CS-23 Certification Specifications and Acceptable Means of Compliance for Normal, Utility, Aerobatic, and Commuter Category Aeroplanes: Amendment 4.* 2015

[Eur16] EUROPEAN AVIATION SAFETY AGENCY: *CS-25 Certification Specifications and Acceptable Means of Compliance for Large Aeroplanes.* 2016

[Fed09] FEDERAL AVIATION ADMINISTRATION: *Advanced Avionics Handbook.* Washington, D.C. : U.S. Department of Transportation, Federal Aviation Administration, Flight Standards Service, 2009

[Fed12] FEDERAL AVIATION ADMINISTRATION: *Advisory Circular 25-7C.* 2012

[Fis07] FISCHER, E.: Standard multi-body system software in the vehicle development process. In: *Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics* 221 (2007), Nr. 1, S. 13–20. – ISSN 1464–4193

[FPU04] FATHY, H. K. ; PAPALAMBROS, P.Y. ; ULSOY, A.G.: On combined plant and control optimization. (2004)

[FRPU01] FATHY, H. K. ; REYER, J. A. ; PAPALAMBROS, P. Y. ; ULSOY, A. G.: On the coupling between the plant and controller optimization problems. In: *Proceedings of the 2001 American Control Conference.* New York : IEEE, 2001. – ISBN 0–7803–6495–3, S. 1864–1869 vol.3

[Gen16] GENERAL ATOMICS AERONAUTICAL: *Predator B RPA.* `http://www.ga-asi.com/predator-b`. Version: 2016

[Ger12] GERDTS, Matthias: *Optimal control of ODEs and DAEs.* Berlin and Boston : De Gruyter, 2012 (De Gruyter graduate)

[GM11] GUJARATHI, G. P. ; MA, Y.-S.: Parametric CAD/CAE integration using a common data model. In: *Journal of Manufacturing Systems* 30 (2011), Nr. 3, S. 118–132. – ISSN 02786125

[Gro08] GROVES, Paul D.: *Principles of GNSS, inertial, and multisensor integrated navigation systems.* Boston : Artech House, 2008 (GNSS technology and applications series)

[GY12] GROŸEKATTHÖFER, Karsten ; YOON, Zizung: *Introduction into quaternions for spacecraft attitude representation.* Berlin, 2012

[Hac15] HACKL, Christoph M.: Dynamische Reibungsmodellierung: Das Lund-Grenoble (LuGre) Reibmodell. In: SCHROEDER, D. (Hrsg.): *Elektrische*

Antriebe – Regelung von Antriebssystemen. Springer-Verlag, 2015, S. 1615–1657

[Has06] Haskins, C.: *Systems Engineering Handbook v. 3.* San Diego, CA: INCOSE., 2006

[HC75] Hunt, K. H. ; Crossley, F. R. E.: Coefficient of Restitution Interpreted as Damping in Vibroimpact. In: *Journal of Applied Mechanics* 42 (1975), Nr. 2, S. 440. – ISSN 00218936

[Hol17a] Holzapfel, Florian: *Flight System Dynamics I.* München, 2017

[Hol17b] Holzapfel, Florian: *Flight System Dynamics II.* München, 2017

[Hol17c] Holzapfel, Florian: *Flight System Identification.* München, 2017

[Hon15] Honeywell International Inc.: *Aerospace and Defense: Sensors and Switches Product Range Guide.* 2015

[HSPH13] Heller, Matthias ; Schuck, Falko ; Peter, Lars ; Holzapfel, Florian: *Hybrides Flugsteuerungssystem für zukünftige Kleinflugzeuge (Future Small Aircraft).* Bonn : Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V, 2013

[JNSM16] Jewoh Zekeyo, Silvain ; Nezhat, Sam ; Schropp, Christopher ; Miller, Steve: *Computergestützte Simulationsschnittstelle: Optimierte Systementwicklung in der Mechatronik.* 18. VDI-Getriebetagung Bewegungstechnik 2016, 20.09.2016

[KMPJ03] Kim, Hyung M. ; Michelena, Nestor F. ; Papalambros, Panos Y. ; Jiang, Tao: Target Cascading in Optimal System Design. In: *Journal of Mechanical Design* 125 (2003), Nr. 3, S. 474. – ISSN 10500472

[Kub08] Kubisch, Matthias: *Modellierung und Simulation nichtlinearer Motoreigenschaften*, Humboldt-Universität zu Berlin, Studienarbeit, 2008

[LAT15] Louhichi, Borhen ; Abenhaim, Gad N. ; Tahan, Antoine S.: CAD/CAE integration: Updating the CAD model after a FEM analysis. In: *The International Journal of Advanced Manufacturing Technology* 76 (2015), Nr. 1-4, S. 391–400. – ISSN 0268–3768

[LHH] Lauffs, Patrick J. ; Hochstrasser, Markus ; Holzapfel, Florian: Real-time simulation of nonlinear transmission behavior in electro-mechanical flight control systems. In: *2014 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology (ICARES)*, S. 39–47

[LPS13] LPS® Laboratories: *Technical Data Sheet - LPS 2® Heavy-Duty Lubricant.* 4647 Hugh Howell Road, Tucker, GA 30084, 2013

[LSW54] Lele, P. P. ; Sinclair, D. C. ; Weddell, G.: The reaction time to touch. In: *The Journal of physiology 123.1.* 1954

[MGM06] Mastinu, Giampiero ; Gobbi, Massimiliano ; Miano, Carlo: *Optimal Design of Complex Mechanical Systems.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2006

[MKLC09] Maier, A. M. ; Kreimeyer, M. ; Lindemann, U. ; Clarkson, P. J.: Reflecting communication: A key factor for successful collaboration between embodiment design and simulation. In: *Journal of Engineering Design* 20 (2009), Nr. 3, S. 265–287. – ISSN 0954–4828

[ML13] Martins, Joaquim R. R. A. ; Lambe, Andrew B.: Multidisciplinary Design Optimization: A Survey of Architectures. In: *AIAA Journal* 51 (2013), Nr. 9, S. 2049–2075. – ISSN 0001–1452

[MMGG12] Mefteh, Wafa ; Migeon, Frederic ; Gleizes, Marie-Pierre ; Gargouri, Faiez: Simulation based design. In: *2012 International Conference on Information Technology and e-Services*, IEEE, 2012. – ISBN 978–1–4673–1166–3, S. 1–6

[Mod] `https://www.modelica.org/`

[MSVW17] Miller, S. ; Soares, T. ; Van Weddingen, Y. ; Wendlandt, J. ; The MathWorks, Inc. (Hrsg.): *Modeling Flexible Bodies with Simscape Multibody Software: An Overview of Two Methods for Capturing the Effects of Small Elastic Deformations.* 2017

[NC12] NG, Annie W. Y. ; CHAN, Alan H. S.: Finger response times to visual, auditory and tactile modality stimuli. In: *Proceedings of the international multiconference of engineers and computer scientists* Bd. 2. 2012

[Ols96] Olsson, H.: *Control systems with friction.* Lund, 1996

[PBFG07] Pahl, Gerhard ; Beitz, Wolfgang ; Feldhusen, Jörg ; Grote, Karl-Heinrich: *Engineering Design.* London : Springer London, 2007

[PK01] Pytel, Andrew ; Kiusalaas, Jaan: *Engineering mechanics: Dynamics.* 2nd ed. Pacific Grove : Thomson, 2001

[PW] Papalambros, Panos Y. ; Wilde, Douglass J.: *Principles of optimal design: Modeling and computation.* 3rd ed.

[RC93] Raymond, E. T. ; Chenoweth, C. C.: *Aircraft flight control actuation system design.* Warrendale, PA : Society of Automotive Engineers, 1993

[Rie13] Rierson, Leanna: *Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance.* Taylor & Francis Inc, 2013

[Ros03] Roskam, Jan: *Airplane Flight Dynamics and Automatic Flight Controls: Part I.* DARcorporation, 2003

[RS5] Richard, Hans A. ; Sander, Manuela: *Technische Mechanik: Lehrbuch mit Praxisbeispielen, Klausuraufgaben und Lösungen : Grundlagen - effektiv und anwendungsnah.* Einzelbände in versch. Ausg. Wiesbaden : Vieweg, 2005- (Viewegs Fachbücher der Technik)

[RS08] Richard, Hans A. ; Sander, Manuela: *Technische Mechanik: Lehrbuch mit Praxisbeispielen, Klausuraufgaben und Lösungen.* 2., erw. Aufl. Wiesbaden : Vieweg+Teubner / GWV Fachverlage, Wiesbaden, 2008 (Vieweg Studium)

[RW99] Reinhart, G. ; Weissenberger, M.: Multibody simulation of machine tools as mechatronic systems for optimization of motion dynamics in the design process. In: *Proceedings.* Piscataway, NJ : IEEE Service Center, 1999. – ISBN 0–7803–5038–3, S. 605–610

[Sch10] Schröder, Valentin: *Prüfungstrainer Strömungsmechanik.* Wiesbaden : Vieweg+Teubner, 2010

[SPK00] Sinha, R. ; Paredis, C.J.J. ; Khosla, P. K.: Integration of mechanical CAD and behavioral modeling. In: *Proceedings 2000 IEEE/ACM International Workshop on Behavioral Modeling and Simulation*, IEEE Comput. Soc, 2000. – ISBN 0–7695–0893–6, S. 31–36

[SPLK01] Sinha, Rajarishi ; Paredis, Christiaan J. J. ; Liang, Vei-Chung ; Khosla, Pradeep K.: Modeling and Simulation Methods for Design of Engineering Systems. In: *Journal of Computing and Information Science in Engineering* 1 (2001), Nr. 1, S. 84. – ISSN 15309827

[The16a] The MathWorks, Inc.: *MATLAB: R2016a.* 2016

[The16b] The MathWorks, Inc.: *R2016a Documentation.* Natick, Massachusetts, 2016

[The16c] The MathWorks, Inc.: *Simscape Multibody: R2016a.* 2016

[The16d] The MathWorks, Inc.: *Simscape: R2016a.* 2016

[The16e] The MathWorks, Inc.: *Simulink: R2016a.* 2016

[Tro07] Tron, Xavier L.: *A380 Flight Controls Overview.* Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal Oberth e.V., 2007

[U.S97] U.S. Department of Defense: *MIL-HDBK-1797 Flying Qualities of Piloted Aircraft: Handbook.* 1997

[VDI93] VDI - Verein Deutscher Ingenieure: *Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte.* 1993

[Ver98] Veryzer, R.: Discontinuous Innovation and the New Product Development Process. In: *Journal of Product Innovation Management* 15 (1998), Nr. 4, S. 304–321. – ISSN 07376782

[VKV04] Vaculín, Ondřej ; Krüger, Wolf R. ; Valášek, Michael: Overview of coupling of multibody and control engineering tools. In: *Vehicle system dynamics* (2004). – ISSN 0042–3114

[Way18] Wayne Stout, PhD: *Aerospace Flight Control Systems.* DesignAerospaceLLC, 2018

[WL13] Wickert, Jonathan A. ; Lewis, Kemper E.: *An introduction to mechanical engineering.* 3rd ed. Stamford, CT : Cengage Learning, 2013

[ZKN+17] Zimmermann, Markus ; Königs, Simon ; Niemeyer, Constantin ; Fender, Johannes ; Zeherbauer, Christian ; Vitale, Renzo ; Wahle, Martin: On the design of large systems subject to uncertainty. In: *Journal of Engineering Design* 28 (2017), Nr. 4, S. 233–254. – ISSN 0954–4828

[ZMRMZ17] Zare, Amir ; Michels, K. ; Rath-Maia, L. ; Zimmermann, M.: On the design of actuators and control systems in early development stages. In: Pfeffer, Prof. Peter E. (Hrsg.): *8th International Munich Chassis Symposium 2017.* Wiesbaden : Springer Fachmedien Wiesbaden, 2017 (Proceedings). – ISBN 978–3–658–18458–2, S. 337–352

[ZSMH18] Zollitsch, Alexander W. ; Schatz, Simon P. ; Mumm, Nils C. ; Holzapfel, Florian: Model-in-the-Loop Simulation of Experimental Flight Control Software. In: *2018 AIAA Modeling and Simulation 2018.* 2018