An FEA-Based Concept Evaluation Tool For Early Vehicle Body-in-White Development

Master's thesis in Mobility Engineering

Shi Chen
Ziyang Zhou

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's Thesis in Mobility Engineering

An FEA-Based Concept Evaluation Tool For Early Body-in-white Development

Shi Chen
Ziyang Zhou

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Dynamics
Chalmers University of Technology
Gothenburg, Sweden 2025

An FEA-Based Concept Evaluation Tool For Early Body-in-white Development
Shi Chen
Ziyang Zhou

Supervisor: Sandeep Shetty, Volvo Cars Corporation
Supervisor: Jonas Elmered, Volvo Cars Corporation
Examiner: Jim Brouzoulis, Department of Mechanics and Maritime Sciences

Master's Thesis 2025
Department of Mechanics and Maritime Sciences
Division of Dynamics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Three Point Bending in Quasi-static analysis.

Typeset in LaTeX
Gothenburg, Sweden 2025

An FEA-Based Concept Evaluation Tool For Early Vehicle Body-in-White Development

Shi Chen
Ziyang Zhou
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

Traditional crash simulation approaches are typically time-consuming and require highly detailed vehicle models, which are often unavailable during the early stages of product development. To address this limitation, Volvo Cars Corporation (VCC) employs an internal concept evaluation tool that allows designers to perform preliminary structural analyzes and ensure designs meet subsystem-level requirements at an early phase. This thesis focuses on the further development of this concept evaluation tool by addressing its existing drawbacks, thereby enhancing its versatility, efficiency, and accuracy to better support early-stage design.

By pairing Python-driven automation with LS-DYNA, the proposed framework transforms repetitive routine into a streamlined, standardized workflow. Scripts regenerate meshes, reapply boundary conditions, and relaunch the simulation the moment a design variable is set. The developed system framework uses the three-point bending scenario, identified as one of the most critical load cases in vehicle structural assessments, as a representative example to build an automated preprocessing module.

In addition, an automated error handling layer was integrated into the framework. It effectively intercepts issues such as problematic meshes, incorrect coordinate systems, or solver warnings early in the process, thereby preventing contamination of results. Due to automation, engineers can conveniently preset cross-sectional parameters such as dimensions, materials, boundary conditions, and other essential inputs. Even under rapid development cycles, all required parameters can be applied correctly and consistently.

Finally, all benefits mentioned above are integrated into a unified graphical interface. It is intuitive enough for design engineers to guide themselves through all steps of the simulation setup, paying attention to the most crucial aspects while minimizing exposure to non-essential components. The framework transforms previously manual processes into automated workflow and background computations, enabling faster and more accurate numerical results.

Keywords: Body-in-White, Finite Element Method, LS-DYNA, Python, CAE;

Acknowledgements

First of all, we would like to extend our most sincere gratitude to our supervisor Sandeep Shetty and Jonas Elmered at Volvo Cars Corporation, who have supported us throughout the thesis project with their great dedication, illuminating our path with insightful guidance and encouragement.

We would also like to acknowledge Jim Brouzoulis, our academic examiner, whose expert insights and thoughtful direction have been instrumental in shaping the outcome of our work.

We are deeply grateful to Henrik Ljungqvist, Engineering Manager at Volvo Cars, for providing us with this valuable master thesis opportunity and supporting our integration into the team. We also thank Tommy Sedin from CAE IT Support for his dedicated assistance with Damida software, significantly facilitating our technical progress.

Finally, we truly like to express our special appreciation to Tobias Eskilsson, Josef Dagström, Austen Clark, Sidd Kumaraswamy, and Dean Madura Gerlam for their genuine interest in this work. Their constructive feedback and keen perspectives have enriched this project, enabling us to elevate this work to a higher level.

Shi Chen, Ziyang Zhou, Göteborg, June 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

BEV	Battery Electric Vehicle
BIW	Body In White
CAD	Computer Aid Design
CAE	Computer Aid Engineering
DOF	Degree of Freedom
FEA	Finite Element Analysis
FEM	Finite Element Method
GCS	Global Coordinate System
IGES	Initial Graphical Exchange Specification
LCS	Local Coordinate System
MG	Microgrid
PV	Photovoltaic
RES	Renewable-based Energy Sources
VCC	Volvo Cars Corporation

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Symbol	**Description**
L	Beam element length
w	Height of cross section
t	Thickness of cross section
b	Flange width of cross section
s	Flange thickness of cross section
R	Elemental rotation angle
u	Displacement magnitude
θ	Rotation or angular offset
x, y, z	Global coordinate axes
x', y', z'	Local coordinate axes of each sub-component
O	Coordinate origin (global or local)
δ	Incremental displacement/perturbation distance
$\vec{a}_x, \vec{a}_y, \vec{a}_z$	Directional unit vectors in local coordinate system

Contents

List of Figures

1
Introduction

In recent years, the time to market has been drastically reduced for the automotive industry. The engineering design and optimization process for vehicles is facing tremendous challenges. Adapting to this trend, simulation-based design processes are crucial in shortening the product development cycle. Having Computer Aided Engineering (CAE) methods and tools in the concept development stage to expedite concept evaluation is quite helpful in shortening the development cycle of a product. However, current CAE concept evaluation tools may not be the best solution in many cases during the early concept development phase. For instance, complete Body-in-White (BIW) models, typically required in traditional CAE crash simulations, are often unavailable at this phase. Even when available, such detailed models are highly time and resource-intensive, making them impractical for rapid concept iteration.

1.1 Background

Over the past few decades, the automotive industry has shifted from small workshops to mass production and, more recently, lean manufacturing [1]. Established auto companies now face growing competition from new electric vehicle manufacturers. As a result, traditional auto makers need to improve their competitiveness by improving operational efficiency and reducing costs.

Given the pressure to reduce time-to-market, simulation-based design processes play a vital role in shortening product development cycles. In the initial phases of product development, having accurate methods and resources for assessing various concepts is crucial. Standard CAE evaluations of vehicle body performance are highly inefficient in the early design phase, often requiring hours or even days to complete, while also consuming substantial computational resources. So, it is necessary to develop better tools for concept design that would improve the integration of analysis and design.

During the vehicle design process, the Body-in-White remains the central element and component of the vehicle's structure. The structural characteristics will impact a vehicle's crashworthiness and lightweightness. The body provides structural integrity while serving as the main support for various other systems, such as the engine, suspension, and wiring, among others. Moreover, there exists a certain optimization for the BIW which includes finding a balance between crash energy absorption and body stiffness, as well as finding a compromise between weight and

cost. With emerging technologies, the rapid advancement of new materials and processes has provided engineers with greater flexibility in designs [2]. During the design validation phase, simulations are conducted based on previous analyses, and extensive post-processing is required. This post-processing is particularly complex in structural optimization scenarios, as these analyses typically involve multiple simulation iterations with numerous parametric variations. Therefore, creating a single tool capable of seamlessly combining automation with finite element analysis (FEA) would be invaluable [5]. Such a tool would reduce the analysis time while eliminating human errors to increase analysis efficiency and precision.

The three-dimensional Computer-Aided Design (CAD) geometry is an input for numerous finite element techniques. In most cases, the CAD drawings and numerical analysis done on them are split between multiple departments. The attention of the analysis is usually directed at the BIW as a fully assembled unit instead of conceptual designs in earlier stages.

This disconnect can be troublesome for the design engineers who, at a conceptual stage, need to create and evaluate multiple cross-section sketches to select the one with better overall performance. The traditional Finite Element Analysis (FEA) techniques are too slow for this purpose, which becomes a constraint in the design workflow. The reasons are as follows. Firstly, model-building for simulation tends to be overly reliant on intricate and tedious manual tasks, which impacts the time an expert takes to construct a model, such as geometry cleaning, mesh generation, adding loads, and so on. Secondly, a lack of automated systems for modifying parameters and extracting results creates time-consuming repetitive processes within each simulation cycle. Thirdly, these tools do not have enough adaptive capacity to quickly and accurately respond to design changes, making the process of validating BIW designs excessively complicated and time-consuming. To satisfy this requirement, Volvo Cars developed a concept structure design tool for the BIW that accelerates concept evaluation in the early design stage of the BIW using a proprietary software called Damida. This software allows engineers to quickly verify if a model fulfills the strength requirements using beam 2D section data from CAD as input. This allows design engineers to receive timely feedback on the performance of the chosen cross-section design, while very limited prior knowledge of FEA is required.

1.2 Previous Work

Dr. Nicklas Bylund discussed simulation-based product development in BIW design in his doctoral thesis two decades ago [2]. After this topic, the VCC in-house software, Damida, was created. The software uses scripting techniques to perform simulations and extract results. This has considerably reduced the computation time needed by VCC design engineers to assess the BIW design, particularly the beam design in BIW. The software was created over twenty years ago, primarily written in JAVA. Current Damida focuses on three load cases: moment load, axial

load, and offset axial load. These are sufficient for the comparison of beam section capacity.

1.3 Goals

The primary goal for this thesis project has been to extend the scope of the existing in-house FEA-based concept development tool, Damida, by introducing advanced features, thereby updating its capabilities. This master's thesis will develop a more versatile procedure to incorporate additional new load case analysis. The simulation achieves an accuracy level suitable for engineers who may not possess extensive expertise in computational analysis, ensuring ease of use without compromising the reliability of the results. To achieve this, stakeholder perspectives and limitations will require in-depth analysis.
The software should be very user-friendly and provide reasonably accurate results. In most cases, typical users can obtain results with only a few interactions, greatly simplifying the analysis workflow. Experienced users, especially those CAE engineers, can run more advanced modes to interact more with the software. Based on this, the following was the scope of the thesis:

- The initial step is to have a deeper understanding of the BIW and the load cases when designing the system. Talking with different engineers and finding out their needs is also an important part of this step. A clear direction will be given at this stage on where this thesis is going to extend the functionality of the Damida.
- The second part is to code new load cases and implement the new functions into Damida.
- The final part is to validate the newly added functions and see whether they can meet the stakeholders' needs and have a good performance in helping design engineers to do quick comparisons with different concepts.

1.4 Limitations and Demarcations

Due to the limited duration of the thesis project, the scope has been defined with the following constraints:

- *Software Limitation*: Once the settings are configured, Damida will perform the meshing internally and submit the task to the LS-DYNA solver. It is important to note that the source code was developed in Java two decades ago. This master's thesis scope will not involve translating the entire source code into Python, but the new functions of Damida will be written in Python.
- *Physical Validation*: Due to limited time and resources, no new physical structure will be built and tested in comparison with the results of the simulated model. However, some old experimental results can be used in the later validation phase to see whether the newly developed function is robust enough.
- *Beam Analysis Only*: A modern car body usually comprises many different sheet metal parts welded together to form an integrated structure. The focus of the software is solely on the analysis of beams. The Damida is designed

to analyze various types of beams for both local axial buckling and plastic bending. Car body joints and panels are analyzed using another in-house software, Adrian. Damida and Adrian were originally developed within the same package, but this thesis will focus only on Damida. Therefore, joints and panels analysis will not be included in this thesis.
- *Accuracy*: One thing that needs to be mentioned is that the software is only used for vehicle concept design, specifically BIW. It gives engineers an approach to quick analysis and comparison with the capacity of another design concept. Complete vehicle analysis in LS-DYNA is still needed for the most accurate simulation results.

1.5 Research questions

Based on the background description and the purpose of this thesis, the following questions are answered during this 20-week project:
- Which new loadcases need to be implemented to make the tool more useful?
- Which function in Damida needs to be improved?
- Can future development of Damida be easier?

1.6 Resources

The following software and resources were used in this master's thesis topic:
- CATIA to generate 2D sections.
- ANSA as pre-processor.
- LS-DYNA as solver.
- METApost for checking results files.
- Visual Studio Code as the editor of the new Damida function source code.
- Python will be the language to develop all those new functions.

1.7 Other Considerations

This thesis has no direct connection to issues concerning the environment or the sustainable development of society. However, saving computational resources will reduce certain energy consumption to some extent. Weight optimization of the vehicle structure will also help the environment.

2
Theory

In the pursuit of a simulation framework that is both robust and industrially applicable, this chapter sets out to establish the theoretical underpinnings of the methods used throughout the thesis.

To this end, the chapter opens with an exploration of Finite Element Analysis (FEA), paying special attention to simulation methodologies, solver schemes, and element behaviour—all of which critically influence the predictive quality and computational cost of crash simulations. Additionally, the structural logic and simulation philosophy of Body-in-White (BIW) assemblies are reviewed, drawing from industry practice at Volvo Cars. Discussions then extend to core safety attributes like crashworthiness and structural load paths, particularly in three-point bending scenarios. Lastly, the study of the Damida simulation platform is presented to frame the relevance and uniqueness of the methods developed in this thesis.

2.1 Finite Element Analysis

Over the past several decades, Finite Element Analysis has evolved from a niche computational technique into a mainstream engineering tool, particularly in the realm of vehicle safety simulations. The versatility and extensibility of FEA enable engineers to explore everything from static simulation to highly dynamic crash events. Typically, there are three different analysis types:

- Static analysis refers to evaluating the long-term response of a structure under a certain type of load. It can be understood as the response when the loading speed is negligibly small [6].
- For dynamic analysis, inertial forces are very significant, for example, in impacts or high-speed loading scenarios. The influence of acceleration is pronounced, and it greatly affects the structural response [3].
- Quasi-static analysis is a degenerated form of dynamic analysis. When the velocity in a dynamic simulation is reduced, the influence of dynamics (inertia) also decreases. As the velocity continues to decrease, the dynamic effects become negligible, and the process can be considered quasi-static [10].

There are generally two types of CAE (Computer-Aided Engineering) time integration methods: Explicit and Implicit. LS-DYNA has the ability to solve both implicit and explicit problems [4]. Explicit solvers are based on dynamic equations, where the displacement at the current time step depends on the velocity and displacement from the previous step, eliminating the need for iteration during the solution process.

In contrast, implicit solvers can solve both dynamic and static problems and typically require iterative computation [7]. For problems exhibiting mild nonlinearity, the implicit method can utilize relatively large time steps compared to the explicit method, often resulting in reduced overall computation time. However, when dealing with highly nonlinear problems, the implicit solver may require significantly smaller time steps to maintain convergence, which can lead to longer runtimes—making it less efficient than the explicit approach in such scenarios [8].

In the present work, while the simulation employs a dynamic explicit solver, the physical scenario, namely a three-point bending test, often lies between quasi-static and fully dynamic, depending on load speed and material properties.

2.2 Structural mechanics and crashworthiness of Body-in-White

Vehicle safety has always been central to the business philosophy of Volvo Cars. As Gustaf Larson, co-founder of Volvo, famously stated, "Cars are driven by people. The guiding principle behind everything we make is, and must remain, safety" [11]. Today, with advancements in computational algorithms, software, and sensor technology, automotive companies increasingly invest in active safety systems — technologies designed to prevent or mitigate collisions during the pre-crash phase. Nevertheless, passive safety continues to hold significant importance within the automotive industry as the final line of defense during accidents.

Effective passive safety heavily relies on the design and structural integrity of the vehicle Body-in-White (BiW). Lawmakers around the world establish basic regulations that define how car manufacturers should design vehicles from a safety perspective. And consumer rating agencies will develop independent testings to verify the actual car's safety performance. However, Volvo typically will go beyond these baseline requirements, with the aim of creating a safer vehicle through specialized crash tests and internal standards. Therefore, study of crashworthiness and vehicle structure would be necessary prior to determining which load case should be incorporated into Damida.

2.2.1 Crash tests and internal tests

Safety standards and consumer rating protocols vary globally. A vehicle designed to meet international market demands must comply with minimum regulatory requirements while achieving favorable ratings in consumer crash tests (e.g., Euro NCAP, IIHS, C-IASI). During the product development phase, Volvo vehicles undergo extensive internal crash testing in addition to mandatory regulatory and consumer evaluations. These stringent internal tests are intended to ensure the highest level of occupant protection.

2.2.2 Load cases and load path

During a frontal collision event—such as full-frontal impact, frontal offset collision, or frontal pole impact—the initial impact energy is absorbed by the vehicle's front structures, namely the bumper beam and crash box. Due to Volvo's progressive deformation design and reinforced safety cage structure, the impact load gradually transfers through multiple structural components, including side members, subframes, engine, firewall reinforcement structures, and eventually through the safety cage (as illustrated in Figure 2.1). This sequential activation of structural elements ensures controlled deformation and optimized energy dissipation.

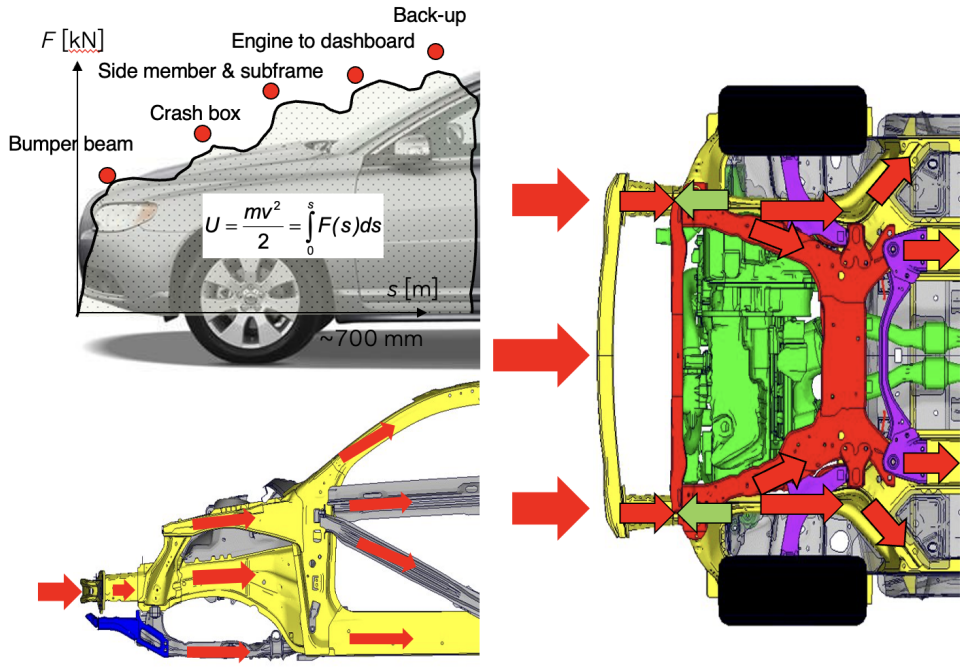

Figure 2.1: Frontal crash load path and structural engagement sequence in a vehicle body.

In side-impact scenarios, occupant protection presents a greater challenge due to limited deformation space between the passenger compartment and the impact source. Recognizing this, Volvo developed the world's first Side Impact Protection System (SIPS) in 1991, significantly optimizing side structural performance and load path management. Modern SIPS integrates a strategically designed safety cage, reinforced structural components—including robust A-pillars, B-pillars, door sills, and optimized door beams—and a carefully coordinated load distribution path. Upon side impact, energy is initially absorbed by the doors, followed by the sequential engagement of the sills, B-pillars, and roof cantrails. Ultimately, the impact forces are distributed efficiently throughout the vehicle structure via lateral cross-members embedded within the main floor and roof assemblies.

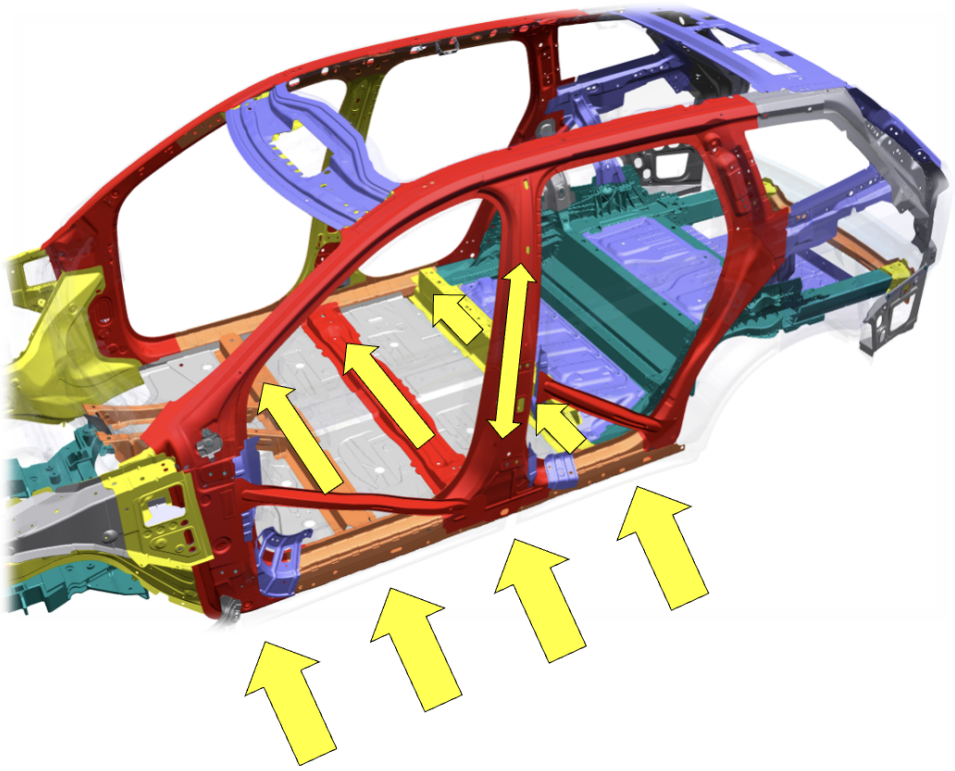

Figure 2.2: The side crash load path for BIW.

Based on these frontal and side crash scenarios, several structural components emerge as critically stressed zones, including the bumper beam, crash box, sill, B-pillars, and other reinforced members. Locally, these structural elements typically experience moment loads, axial compression, three-point bending, pole intrusion loading, and axial crash loading. Given that the Damida software already handles moment and axial loading analyses adequately, the scope of this thesis prioritizes developing simulation capabilities specifically focused on three-point bending, pole crash load cases, and axial crash scenarios.

A targeted survey conducted among engineers at Volvo, specifically design engineers, indicated an agreement to prioritize the development of the three-point bending load case. This decision underscores the significance of accurately capturing bending behavior within structural components, ensuring the design robustness and crashworthiness of Volvo's future BIW concepts in the early development phase in a cost efficient way.

2.3 Current tools study

2.3.1 Pre-study of Damida

Damida is an internal software developed by Volvo Cars designed specifically for quick structural analysis during early-stage vehicle design. According to previous

research documented by N. Bylund, Damida is characterized by its lightweight design, simplified workflow, and ease of integration into existing CAE environments at Volvo [2]. The software package is notably compact, occupying only approximately 150 MB on disk, enabling efficient deployment and fast operation within design teams.

Damida's primary functionality is its ability to generate simplified structural models from user-defined two-dimensional cross-sections. Specifically, the software can automatically extrude these 2D profiles into fully meshed 3D geometries, preparing them directly for analysis under various load conditions, including axial loads and bending moments.

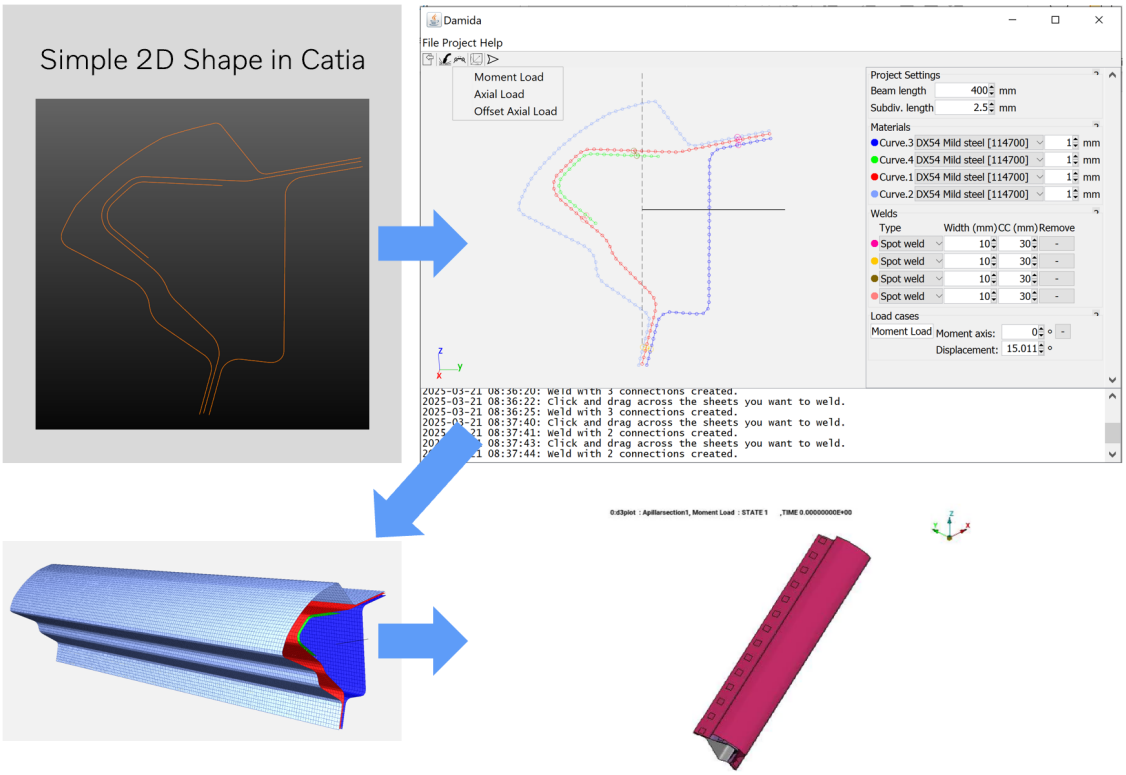

Figure 2.3: This figure illustrates the workflow of the Damida simulation process.

The computational analysis itself is performed by the LS-Dyna solver running on dedicated Volvo servers. Upon completion of the simulation, Damida automatically compiles and presents results in a user-friendly HTML interface accessible via a standard web browser. Typical outputs provided by Damida include force–displacement curves, moment diagrams, internal energy plots, and interactive animations clearly visualizing deformation patterns.

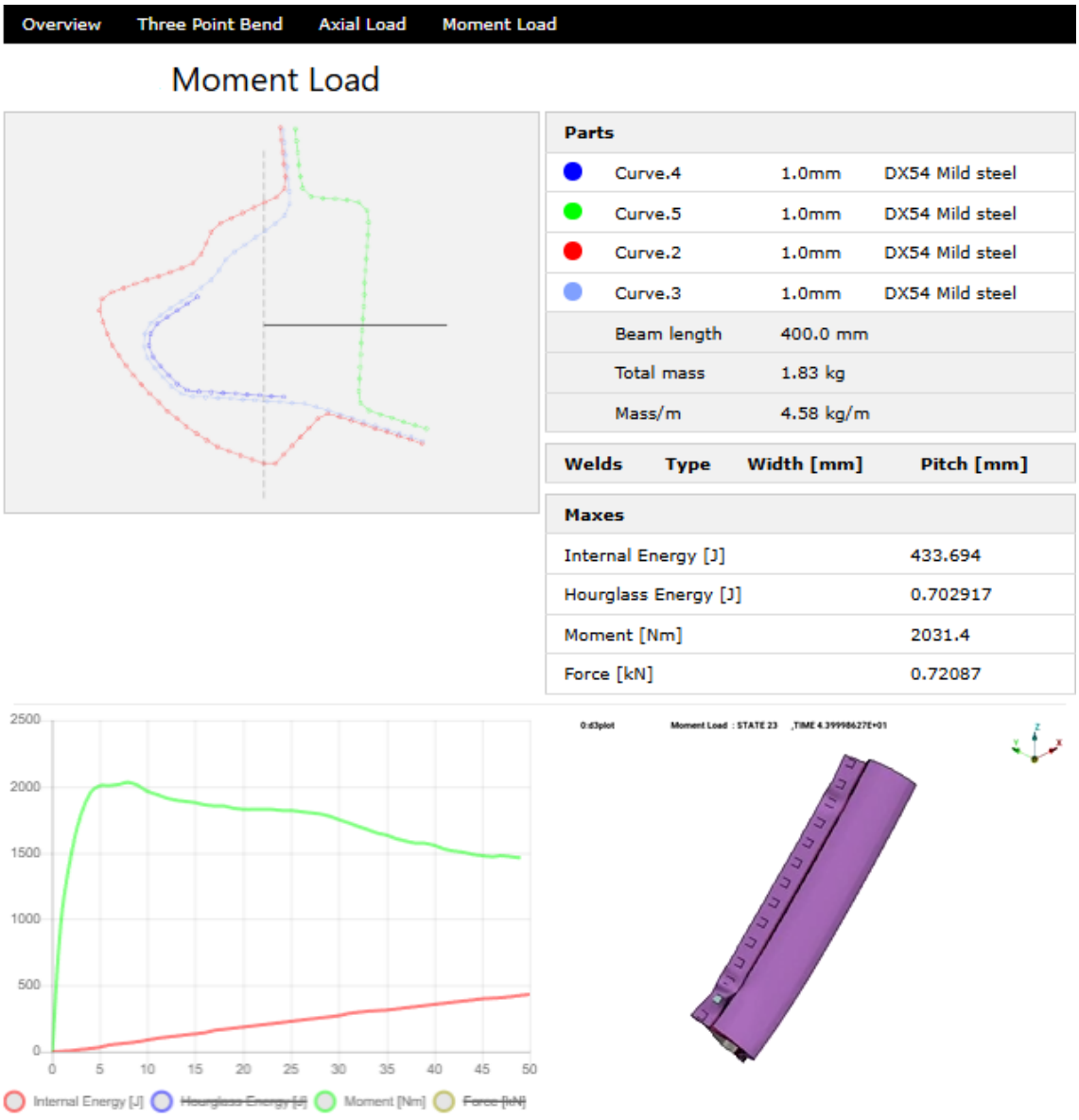

Figure 2.4: Example of the Damida output.

2.3.2 3D mesh in Damida

Damida utilizes a streamlined meshing algorithm optimized for rapid geometric transformation. The software accepts simple cross-sectional sketches input by the user and then employs automated extrusion techniques to generate computationally efficient shell meshes with a user defined element length suitable for early-stage structural analyses. By minimizing manual preprocessing steps, Damida reduces setup time and accelerates feedback loops for structural optimization.

3
Methods

This chapter describes the simulation automation workflow developed in this thesis, alongside its integration with Damida, Volvo Cars' proprietary concept modeling software. Emphasis is placed on implementing a standardized Three-Point Bending (TPB) load case which is a structural test performed to assess stiffness, energy absorption, and load path behavior in a Body-in-White (BiW) model during the early development stages. The code of two programs in this thesis is written in Python and primarily uses the `re` and `os` libraries for text parsing and file handling, respectively.

Rather than the outputs from the simulation, the chapter explains the engineering reasoning and automation processes involved in deriving TPB configurations from Damida beam definitions. The automation logic is implemented through two dedicated Python scripts which draw material and geometric details from existing models in Damida, then create corresponding LS-DYNA simulation files. This workflow transcends the boundaries of physical geometry and simulation-ready models, enabling physical validation at any stage of the design process.

3.1 Fix Cracks In The Model

While creating beams in Damida from curves, nodes are placed automatically along the defined path. However, for curves that are supposed to form closed loops—like in the case of crash rail or tube-like BiW structures—the first and last node coordinates are not exact, leading to a small gap at the closure. Figure 3.1, taken from ANSA, illustrates this. The crack marked in red will lead to erroneous outputs. This is simplified in LS-DYNA as a structural crack. While this appears trivial in a CAD environment, gaps like this will result in contact misalignment, local over-stressing, fracture during simulation, or loss of structural integrity.

This gap becomes an issue for small scale beams, where even a sub millimeter gap becomes magnified. Design engineers have, more often than not, been compelled to adjust these connections manually before simulation, which is tedious and highly susceptible to error.

The current work proposes an automated method to detect and repair unintended gaps in beam connections. The solution, implemented in Python, inspects the nodal coordinates of beam elements defined by curves. It evaluates the spatial distance between start and end nodes, and if an overlap without proper bonding is detected,

the nodes are merged to close the gap. This ensures continuity at beam closures while preserving structural integrity and the intended mechanical behavior of the design.

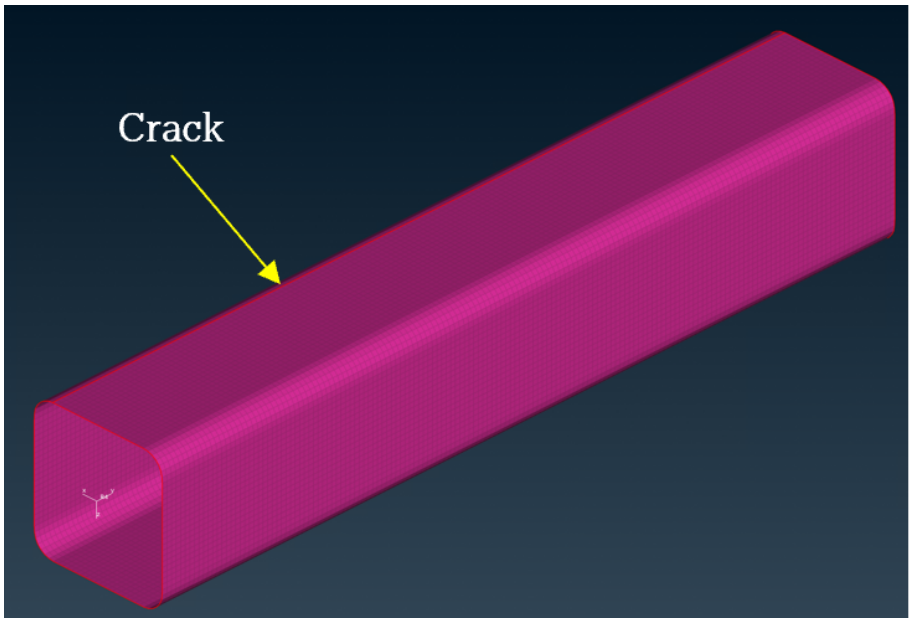

Figure 3.1: The crack in the beam model caused by imperfect geometry or unmerged nodes, which may lead to simulation instability if left unresolved.

3.1.1 Methodology of Fixing Cracks

1. The program will read and parse the **NODE* sections. In this step, all node IDs will be extracted as well as their coordinate values.
2. By analysing those coordinates, the program will compare the nodes based on their coordinate strings and retain the first occurrence.
3. Rebuilding the **NODE* section and removing the duplicate nodes.
4. Updating the element data. The removed node IDs in the **ELEMENT_SHELL* section will be replaced by using mapping.
5. A new file will be written and saved.

3.1.2 Implementation of Fixing Cracks

The code is structured as follows:
The first step is identifying the *NODE label and reading the data afterwards. Extract all node IDs and the corresponding coordinate values. These coordinates are stored as strings for comparison. Duplicate entries are tracked using a dictionary and marked for removal.

```
1     if line.strip().upper().startswith('*NODE'):
2             in_node_section = True
3             node_section_start_index = i
4             node_section_lines.append(line)
```

In the second step, the program will remove the duplicate nodes and record the removed node IDs at the same time. The coord_dict tracks the first occurrence of each coordinate value. The script builds a node mapping and uses it later for element updates.

```
    if coords in coord_dict:
            # Found a duplicate node
            mapping[node_num_str] = coord_dict[coords]
            remove_indices.add(idx)
     else:
            coord_dict[coords] = node_num_str  # Store the
                FIRST node
    new_node_data_lines = [line for idx, line in enumerate
       (node_data_lines) if idx not in remove_indices]
```

Third step, update the **ELEMENT_SHELL* section. Each line in the element shell section will be read and updated to reflect new node IDs using the mapping dictionary.

```
if in_elem_section:
            # Process the *ELEMENT_SHELL data
            eid, pid, *nodes = [line[i:i+8].strip() for i
               in range(0, len(line), 8)]
            nodes = [mapping.get(n, n) for n in nodes]
            updated_lines.append(f"{eid:>8}{pid:>8}{''.
               join(f'{n:>8}'_for_n_in_nodes)}\n")
```

3.2 Overview of Three Point Bending Workflow

The Three-Point Bending (TPB) test is a widely used mechanical loading method for evaluating the flexural behavior of structural components. In a typical TPB setup, beam type specimens are supported at two points symmetrically placed about its centerline, and a concentrated load is applied at the mid-section to create a defined bending. Key structural parameters like stiffness, bending strength, energy absorption, and failure modes can be evaluated through this setup.

Because of its geometric straightforwardness and reproducibility, TPB is frequently employed in material characterization and conceptual evaluation in structures and systems, particularly in the automotive sector, which demands rapid feedback on early Body-in-White (BiW) designs on performance measures. Here, the TPB load case is well suited to examine the impact of changes in cross-sectional geometry, material, and joining on bending response.

Simulation of TPB conditions in LS-DYNA permits focused studies of contact interactions, plastic changes, and localized instability, thus substantiating parametric models developed in earlier stages of design.

The flow developed in this thesis to realize the TPB load case on Damida is aimed at minimizing the likelihood of mistakes, improving reproducibility and consistency across different utilizations of the simulations, making development easier, as well as providing a controlled, repeatable, automated, and hierarchical simulation environment.

3.2.1 Rationale and Engineering Motivation

Design engineers can quickly compare different beam cross sections to find the most balanced design. In traditional setups, design engineers often need to ask CAE engineers to help them do a detailed simulation. It significantly impedes the ability to iterate through different scenarios swiftly during parametric studies or while experimenting with alternative designs.

This workflow transforms the simulation model generation process into an efficient and streamlined approach by adopting a script-based method. It ensures that each simulation model is created with consistent formatting and high accuracy and can easily adapt to boundary conditions or component geometry changes.

Additionally, strict formatting standards are usually followed by simulation templates. Through this precise approach, alignment of all results with these standards is ensured, while the customisation of cross-sections, boundary conditions, and load application elements is still allowed.

3.2.2 New Workflow High-level Diagram

To enable this load case within a scripted concept evaluation workflow, a standardized and reproducible simulation process was defined and integrated into the automation workflow. The proposed workflow consists of six steps, each of which contributes to the assembly and configuration of the LS-DYNA input deck. To summarise, these stages include:

1. Extraction of geometrical and configuration data from source .k files;
2. Computation of local coordinate system (LCS) directions and transformation vectors;
3. Analysis of beam geometry and calculation of offset values for sub-component placement;
4. Transformation of sub-component coordinates and generation of .k templates;
5. Insertion of transformed components and metadata into the simulation template;
6. Final update of control parameters and beam metadata to complete the simulation input.

Each phase of the implementation is fortified with a robust error-checking mechanism that swiftly identifies and addresses potential issues that may arise during the process. This workflow allows design engineers to easily set up the simulation framework for a wide array of tasks, including different beam geometries and impact

angles for TPB.

In Figure 3.2, the automation workflow is vividly represented with a comprehensive block diagram illustrating the program sequence. Each step encapsulates a critical operation within the script.

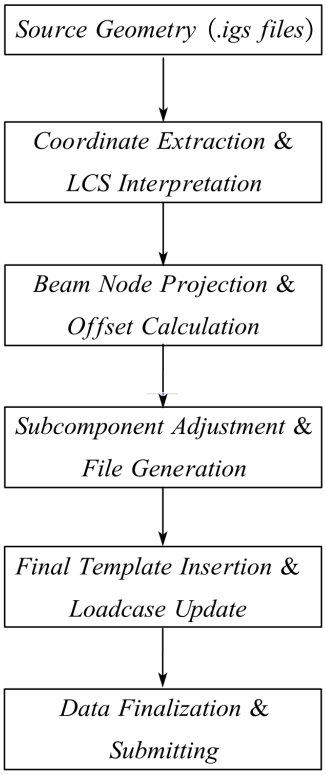

Figure 3.2: The Flowchat in a high-level diagram contains six key steps.

3.2.3 Error Handling and Script Robustness

The Python program includes multiple usability safeguards for error detection and user level exceptions. To enhance the resilience of the scripts, the following protective mechanisms have been employed:

- Functions do not fail catastrophically; instead, they omit lines with malformed or non-existent coordinates;
- Pre-file validation parses are performed using keyword validation (e.g., checking for **DEFINE_COORDINATE_SYSTEM*).
- Division-by-zero errors caused by attempting to execute protective measures against possible vector zeroing in normalization during try-except structures.
- During template processing, divergences from fixed line format are permitted, preserving multi-format template compatibility.

Moreover, other boundary conditions might be parameterized to extend the script by CAE engineers.

3.2.4 Portability and Reusability

No additional third-party libraries are needed for TPB calculations aside from Python 3 and Numpy. The script is designed to be reused not just for the three-point bending simulations but for other structural tests. The aspects of reusability will be covered later in this thesis.

Moreover, other boundary conditions might be parameterized to extend the script. This form of extensibility was crucial from a design perspective to provide sustainable value in the future, beyond the present use.

3.3 Include-files

This workflow is designed around a set of well-defined inputs and outputs. The key inputs are some templates, which will be discussed in the following sections:

3.3.1 Geometry File

This k-file contains the beam component which is also called the target file. The geometry is generated by using the Damida mesh generation function. After generating the geometry, the DAMIDA will name this file as *threepointbend.k*. The file should contain nodes, elements and some essential data, such as maximum bending displacement for the beam, shell thickness and material information.

3.3.2 Impactor and Supporter

The roller file has been separated into two parts. The support and the impactor both contain the geometry information and constraint information. The supporter model has been constrained in all 6 DOFs, and the impactor has been constrained in 5 DOFs. The direction of the X axis is not constrained as it is the direction of displacement. Because the impactor should move in that direction with a standard velocity of 1mm/ms as a quasi-static simulation.

3.3.3 Positioning of Impactor and Supporters

The alignment of the models is one issue that needs to be resolved. This thesis uses the **INCLUDE_TRANSFORM* and **DEFINE_TRANSFORMATION* features in LS-DYNA to apply rotation and positioning to sub-models without altering their internal node definitions. The process involves using a *DEFINE_TRANSFORMATION* card to set rotation angles along X, Y, and Z axes as well as translational offsets. The sub-model is referred by **INCLUDE_TRANSFORM* using the designated transformation ID. LS-DYNA will automatically execute the transformations during preprocessing.

Two transformation templates are created, one for the supporter and one for the impactor. Figure 3.3 below illustrates the transformation template under discussion.

```
*DEFINE_TRANSFORMATION_TITLE
push_device-Pos4
      602
$--------1---------2---------3---------4---------5---------6---------7---------8
POINT     1         0.0       0.0       0.0
POINT     2         0.0       0.0       0.0
POINT     3         0.0       0.0       0.0
POINT     4         0.0       0.0       0.0
POINT     5         0.0       0.0       0.0
POINT     6         0.0       0.0       0.0
POS6P     1         2         3         4         5         6
*INCLUDE_TRANSFORM
3PB_supporter.k
        0         0         0         0
        0
        1         1         1         1         1
      602
$--------1---------2---------3---------4---------5---------6---------7---------8
```

Figure 3.3: Example of the transformation template.

3.3.4 Final Calculation File

This is the main simulation control file in which users can find everything they need to perform the calculation named as *TPBmain_template.key*, and all the data needed will be implemented in this file. Then, the Damida will post it to the server, and the user will soon get their results in a couple of minutes, depending on their simulation time.

3.4 Node Reading and Projection

The spatial determination of the test beam's outline along the local X-axis represents the crucial phase in setting up the TPB simulation. This can be accomplished through importing node values from an LS-DYNA model file and transforming those values to the position template discussed in Section 3.3. The described script will analyze the beam model shell thickness and augment the roller parts by the calculated offsets which take into account shell thickness to ensure no intersection or inadmissible contact conditions occur during the simulation. This report will cover the methodology and significance of automation with nodal reading and projection.

3.4.1 Purpose of the Node Analysis in Preprocessing

Establishing an accurate and effective span along the local X-axis remains the immediate priority. That calculated length, in turn, dictates where the roller supports must rest so that applied loads move freely along the beam. In a standard symmetric three-point-bending configuration, the drop mass sits at mid-travel while the two reaction blocks support the beam's edges.

Real-world CAD assemblies, as often archived in Damida's global .k files, don't always coincide with the numerical origin, so some repositioning becomes unavoidable. The .k draft is extracted, then re-oriented to lie squarely with the principal X-axis. Positive and negative terminal distances off that new trace define the workable support range once the model is relinked to LS-DYNA.

3.4.2 Parsing Node Data in LS-DYNA

The function *read_nodes_from_k()* is responsible for reading all node values from the beam geometry. k file. To extract these values, the script will follow the logic below:

1. Open the beam geometry and start to read line by line;
2. Find the **NODE* block;
3. Read each subsequent line until meet *$* symbol which indicates the end of this node section and start to search another node section;
4. Extract the X, Y, Z values from fixed-field position:

```
x = float(line[8:24].strip())
y = float(line[24:40].strip())
z = float(line[40:56].strip())
points.append([x, y, z])
```

5. Convert the collected data into floating-point numbers;
6. Store the extracted values in a Numpy array for further process.

This problem-solving flow will allow the computer to correctly process node data even if space, tabs or formatting problems are present. The values stored in the Numpy array will also be easy to get and process in later steps.

3.4.3 Quality Check and Error Handling

Models are more often than not, imperfect in the real world, especially in the concept design phase of any given project. For example, there are inconsistencies where nodes may not be aligned due to the IGES (Initial Graphics Exchange Specification) file's lack of precision.

To enforce some structure to the script, four validation steps are included.

- If the node block is empty or does not close properly, a warning alert is issued, and the process will cease;
- If data in the node section is NaN or Inf, those values will be discarded;
- Files that contain fewer than three nodes will be queued as invalid for projection analysis;
- If the main program detects that the array has become empty, it will trigger a prompt showing a gentle error message and terminate gracefully.

The function integrated within system provides notification to users when the information is insufficient and missing during the execution of tasks.

3.4.4 Vector Projection onto Local Axes

After reading all the valid nodes from the beam geometry, these data will be saved into a 2D Numpy array of shape (n, 3), where n is the number of total nodes. In this step, each node will be projected to the beam's local X-axis (a_x, see section 3.4) and normalised to the unit length. The mathematical formula is shown below:

$$x'_i = (\vec{P}_i - \vec{O}) \cdot \vec{a}_x$$

In python language, this formula is programmed as :

```
translated_points = points - origin                    1
x_coords = np.dot(translated_points, ax)               2
```

Then, the results will be returned to a 1D array of projected distances along the beam axis.

3.5 Positioning Logic for Rollers

The sub-component offset logic acts as a crucial link between coordinate system extraction and final model assembly. By incorporating projection calculations, thickness variations, and clearance constraints, it enables accurate, geometry-aware placement of parts. This improves simulation accuracy, reduces runtime, and ensures smoother automation. This function can also be extended to newer load cases, as the logic suits plenty of load cases. This approach may be extended to support more advanced configurations, including multiple support structures, angled impactors, or beams that follow curved and non-linear paths defined by their coordinate systems.

3.5.1 Motivation and Challenges

This study addresses two common issues in setting up large-scale simulations. First, beam models often inherit arbitrary coordinate frames from their CAD origins, leading to misalignment when parts are reused or re-exported—manual correction is error-prone and reduces reproducibility. Second, small gaps or overlaps between contact surfaces in LS-DYNA can cause numerical instability and unrealistic results.

Therefore, one particular dynamic positioning system that calculates offsets based on beam geometry in LCS is necessary. In order to achieve this, the function developed in this thesis must comply with these three basic physical principles.

- The first principle states that the script should maintain a gap sufficient to avoid initial contact penetration between the rollers and the test beam;
- The second principle states that the script must ensure that the beam is installed correctly between the rollers and in accordance with the requirements of the physical test.
- The third principle states that the program must record the thickness and profile of the beam.

3.5.2 Input Parameters for Offset Calculation

To start the calculation, some inputs are required, which are obtained from the previous stages of the workflow (See sections 3.4 and 3.5). Several important input values are listed below:

- The *beam LCS origin* and *Xdirection vector a_x* will be used for orientation and projection;
- All *beam node coordinate values* will be analysed to find the maximum positive and maximum negative local Xaxis projections;
- The *maximum sheet thickness t_max* will be extracted out to add robustness to the function;
- The *safety distance* will be set as 2 mm by default in order to get the highest efficiency in the later calculation stage while maintaining the good robustness of the system.

Based on these input values, the Python program will compute two key values as the output data for this function: *position_for_supporter* and *position_for_impactor*. These two values will define how far along the x-axis of the LCS of the beam each roller should be translated in order to be moved to the correct position.

3.5.3 Establishing Clearance Rules

To avoid unwanted contacts while balancing the calculation efficiency, the distance between the rollers and the beam should be maintained at a safe margin. The rule for this script is:
Supporters are placed below the test beam:

$$d_{\text{supporter}} = \max(x) + c + t_{\max}$$

The impactor is placed above the test beam:

$$d_{\text{impactor}} = |\min(x) - c - t_{\max}|$$

In this equation, c stands for the clearance and is kept at 2 mm; *t_max* is the maximum thickness of the beam. After considering the beam's thickness, the script ensures that even if one beam has an extremely large shell thickness, the roller will be positioned correctly and avoid any unwanted contacts across the whole simulation.

3.5.4 Implementation in Code

In the Python main script, the logic is expressed as follows:

```
1 position_for_supporter = max_positive + 2 + max_thickness
    if max_positive is not None else 2 + max_thickness
2 position_for_impactor = abs(max_negative - 2 -
   max_thickness) if max_negative is not None else 2 -
   max_thickness
```

The function *adjust_coordinates* modifies the coordinate definition lines from the **DEFINE_COORDINATE_SYSTEM* block for every component.

For example, if the original impactor LCS origin is [0.0, 0.0, 0.0], and position_for_impactor = 70.0, then the modified origin becomes [70.0, 0.0, 0.0]. This modification is applied to all three defining points in the LCS definition (origin, X-point, and XY-plane point) by uniformly shifting the X-component. The rules for this three-point bending are: based on the beam's data, the impactor will use add operation and the supporter will use subtract operation, see the following codes:

```
def adjust_coordinates(values, adjustment, operation):
    if operation == 'add':
        values[0] += adjustment
        values[3] += adjustment
        values[6] += adjustment
    elif operation == 'subtract':
        values[0] -= adjustment
        values[3] -= adjustment
        values[6] -= adjustment
```

3.5.5 Directional Logic and Sign Conventions

A widespread difficulty in spatial transformation workflows is ensuring uniformity in signs and conventions, particularly in local-to-global frame transitions. To minimize confusion for the users and Damida programmers, the positive direction of a_x is defined as forward according to the beam's geometry and is further constrained to the movement of rollers in the longitudinal direction. For instance, the impactor shifts in the $+X$ direction while the supporters shift in the $-X$ direction. This process minimizes common errors of reversed vector signs or erroneous reference frames due to misinterpretation of coordinate systems.

3.5.6 Handling Missing or Invalid Projections

Though a configuration of a beam positioned entirely on one side of the origin is highly unlikely, it is noteworthy that the structure of Damida's mesh generation module is designed to always center the cross-section of the beam, thus positioning the origin strategically. This functionality, envisioned while developing this thesis, optimized it for potential future revisions. For edge cases in which *max_positive* or *max_negative* return None, the logic falls back to default behavior as outlined below:

```
if max_positive is None:
    position_for_supporter = 2 + max_thickness
if max_negative is None:
    position_for_impactor = 2 - max_thickness
```

3.6 Transformation and Template Writing

It is crucial for the data regarding the transformations to be put in a precise place after the sub-component structure placement logic has been set up. In this case, updating the keyword blocks within the transformation template, as well as the final key template of the calculation, needs to be done. The placement of components is seamlessly done during this transformation phase where spatial data from LCS (Local Coordinate System) is incorporated, thus providing proper representation of the component alignment.

Accurate simulation requires careful control of template formatting, part placement, and transformation settings. For a TPB setup, even small errors in these steps can break alignment with real-world test conditions.

The present subsection details the incorporation of transformed coordinates into LS-DYNA .key files. Emphasis is placed on syntax compliance and structural integrity, with the goal of sustaining an uninterrupted data flow throughout every stage of the simulation.

3.6.1 Motivation and Challenges

At this point, two fundamental steps are achieved. The first task involves overwriting the origin and direction vectors of each subcomponent's box template to accurately reflect the earlier computed spatial realignments. The second task ensures that coordinate stipulations in the .key file is set as a high priority and adheres to LS-DYNA rigid fixed width syntax rules. Having accomplished these 2 steps, the modified templates are incorporated into the main simulation using **INCLUDE_TRANSFORM*, which allows LS-DYNA to apply the necessary transformations automatically during the simulation run.

3.6.2 Structure of the Sub-component Templates

As previously discussed in section 3.3.3, to fully operationalize the TPB, both transformation templates for supporters and the impactor must be completed. These templates are labeled as *Supporter_template* and *Impactor_template*, which include *DEFINE_TRANSFORMATION_TITLE* and *INCLUDE_TRANSFORM* blocks. Furthermore, these transformations are important within the LS-DYNA keywords; for supporters, the transformation ID is 602, and for the impactor, it is 601. Figure 3.3 illustrates points 1-6, with each line containing an ID and three floating-point numbers representing the X, Y, and Z coordinates of that point concerning the LCS. These coordinates will be populated with previously computed results in alignment with LS-DYNA requirements, typically filling fields with 10 characters.

3.6.3 Applying the Transformation

Based on the previous script functions, all necessary data to calculate the new position for the rollers are accessible, and the updated values are merged with the beam LCS values. *POINT 1-3* has nine values, and they are the transformed subcomponent (in this thesis, it is the rollers), LCS, including the original point, ax point, as well as the bx point. *POINT 4-6* is the unmodified beam LCS used for reference.

By applying the function called *write_points_file()*, these 18 values will be handled and inserted into the template. To meet the LS-DYNA strict syntax requirements, several rules should be followed:

- The placeholder value is "0.0" and each POINT line is identified by matching this value;
- Fields in the natural columns 21-30, 31-40, and 41-50 are replaced by the result values from previous stages;
- Each replacement field is right-padded to 10 characters.

To realise this logic by using the Python code, there is a simplified version of the function script:

```
for i, line in enumerate(template_lines):
        if "0.0" in line:
            new_line = list(line)
            for pos in range(20, 50, 10):
                if replacement_index < len(replacements):
                    value = replacements[
                       replacement_index]
                    formatted_value = f"{value:<10}"[:10]
                    new_line[pos:pos+10] =
                       formatted_value
                    replacement_index += 1
            template_lines[i] = "".join(new_line)
```

The above code ensures that the formatting remains aligned with the LS-DYNA syntax and can be inserted into the final key template without causing any mistakes.

3.6.4 Output File Generation

To add robustness to the whole system and ensure correct transformation between the models, two output data files will be generated for developers and users checking the values to see whether they contain error values, and then they can adjust the model accordingly. The output files are *ImpactorTranslog.k* and *SupporterTranslog.k*. Another advantage of creating these two log files is that they will be easier to include in the final .key file. Moreover, the program will also print a statement to show the user that the output function is executed successfully, to assist debugging if necessary, see Figure 3.4.

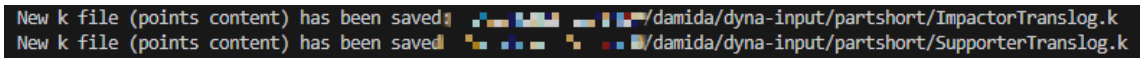

Figure 3.4: Success generation of the transformation files is posted.

3.6.5 Coordinate System Consistency

The most important LCS from the beam is always added to the transformation list, unchanged and in its original form, to maintain consistent transformations across simulations. In the three-point bending load case or other load cases which require model transformation, keeping the LCS of the target model as a stable reference point is the key to eliminating undesired errors. Whether it is for aligning the load and contact, calculating the distances between origins, or referencing in the post-processing scripts, this original LCS can act as a reliable anchor for the whole simulation system.

To add robustness, the a_x (the local x-direction vector) is stored separately in a plain text file named *lcs_values.txt*. This file will be used later in automated steps, for example, applying velocity to the impactor and generating the load curve as the reference vector to get the final contact force in the correct direction. Decoupling this vector from the template can introduce further flexibility into this simulation workflow or other load case workflows.

3.6.6 Formatting for *INCLUDE_TRANSFORM

The script will then try to find the **INCLUDE_TRANSFORM* block, and dynamically copy the source component file path of *3PB_supporter.k* as well as *3PB_impactor.k*. The dynamic file path location will then be pasted into two log files using the function *update_include_transform()*. This can ensure the transformation file is complete and contains the whole geometry definition. See the codes below:

```
1  def update_include_transform(log_file_path,
       dynamic_suffix, base_dir):
2
3      with open(log_file_path, 'r', encoding='utf-8') as
           file:
4          lines = file.readlines()
5
6      updated_lines = []
7      for line in lines:
8          if "*INCLUDE_TRANSFORM" in line.upper():
9              # Append the dynamic address
10             dynamic_address = os.path.join(base_dir,
                   dynamic_suffix)
11             updated_lines.append(line)
12             updated_lines.append(f"{dynamic_address}\n")
13         else:
14             updated_lines.append(line)
```

3.6.7 Validation and Quality Check

As mentioned in this thesis, LS-DYNA has strict syntax requirements. All parsers in this Python script are strict about spacing and alignment. To ensure zero mistakes, validation checks are built into this template writing function, through string slicing, line-length checks, and test runs of the generated .key files before handing them over to the supercomputer in the company to run the simulation. The code will first check incorrect field widths to see if any placeholder is below 10 characters or if it is missing decimal precision. Then, it will check for unexpected line breaks or corrupted content to remove problems from the final .key files.

To visualise these validation steps and give users more detail about their input data, the transformed origin coordinates, the transformed X and XY points, the relative offset from the beam, and the output file path will be printed. See Figure 3.5 and Figure 3.6.

```
Origin of LCS: [0.      0.89764 1.2494 ]
X-axis direction vector in LCS: [0. 1. 0.]
```

Figure 3.5: Example of the LCS information.

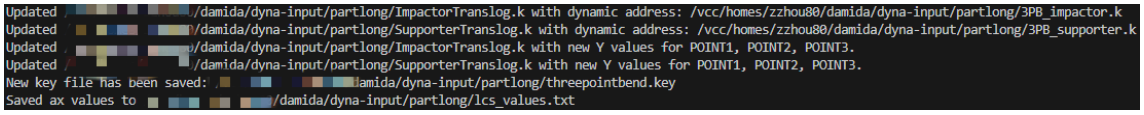

Figure 3.6: Saved files' information is posted.

3.7 Loadcase and Control Data Integration

Recent TPB simulations conducted in LS-DYNA reveal that mere manipulation of geometry or coordinate frames falls short; a parallel effort to automate loading conditions, boundary definitions, and various control settings proves indispensable.

The following discussion details a procedure in which load cases and control files are fused into a single master template. This process incorporates automated block extraction, template insertion, and adaptive boundary adjustments driven by the model's geometry.

3.7.1 Template Placeholder for Loadcase

The main simulation template file has created a dedicated placeholder as *$ TP-BLOADCASE*. This marker indicates where the simulation-specific loading conditions should be inserted. The logic for this insertion is:

- Search for the placeholder in the template file;
- Store its line index;

- Inject the extracted loadcase block after the index;
- Write the results into the final main .key file - *threepointbend.key*.

This guarantees that every simulation generated from the program has an accurate and contextually relevant loading definition, maintaining consistency across runs. See the codes below:

```
new_template_lines = (
    new_template_lines[:tpbloadcase_index] +
    loadcase_block +
    new_template_lines[tpbloadcase_index:]
)
```

3.7.2 Control Card Modification

In dynamic and quasi-static simulations, the termination time defines how long the simulation should run.

Rather than hardcoding the termination time every time, the Python script will read the input data from the beam geometry-k file. Once the target value is found, the script updates the lines immediately by finding the **CONTROL_TERMINATION* block in the template file. Only characters in positions 1-10 are overwritten to follow the strict LS-DYNA syntax. See codes below:

```
match = re.search(r'"displacement":\s*([\d.]+)', line)
formatted_value = f"{displacement:>10.1f}"
next_line[:10] = formatted_value[:10]
```

In this logic, the script can ensure the accurate simulation length and prevent premature termination or wasted computation.

3.7.3 Integration with Beam Length

An essential component of metadata integration is confirming that the LCS reference points and specifically, the Y coordinates are properly calibrated, see Figure 3.7. These points are called POINT1, POINT2, and POINT3 and are used for positioning sensors or control devices along the beam. Instead of assigning fixed values, their Y coordinates are adjusted proportionately to the actual length of the beam, preserving proper alignment throughout mechanical alterations.

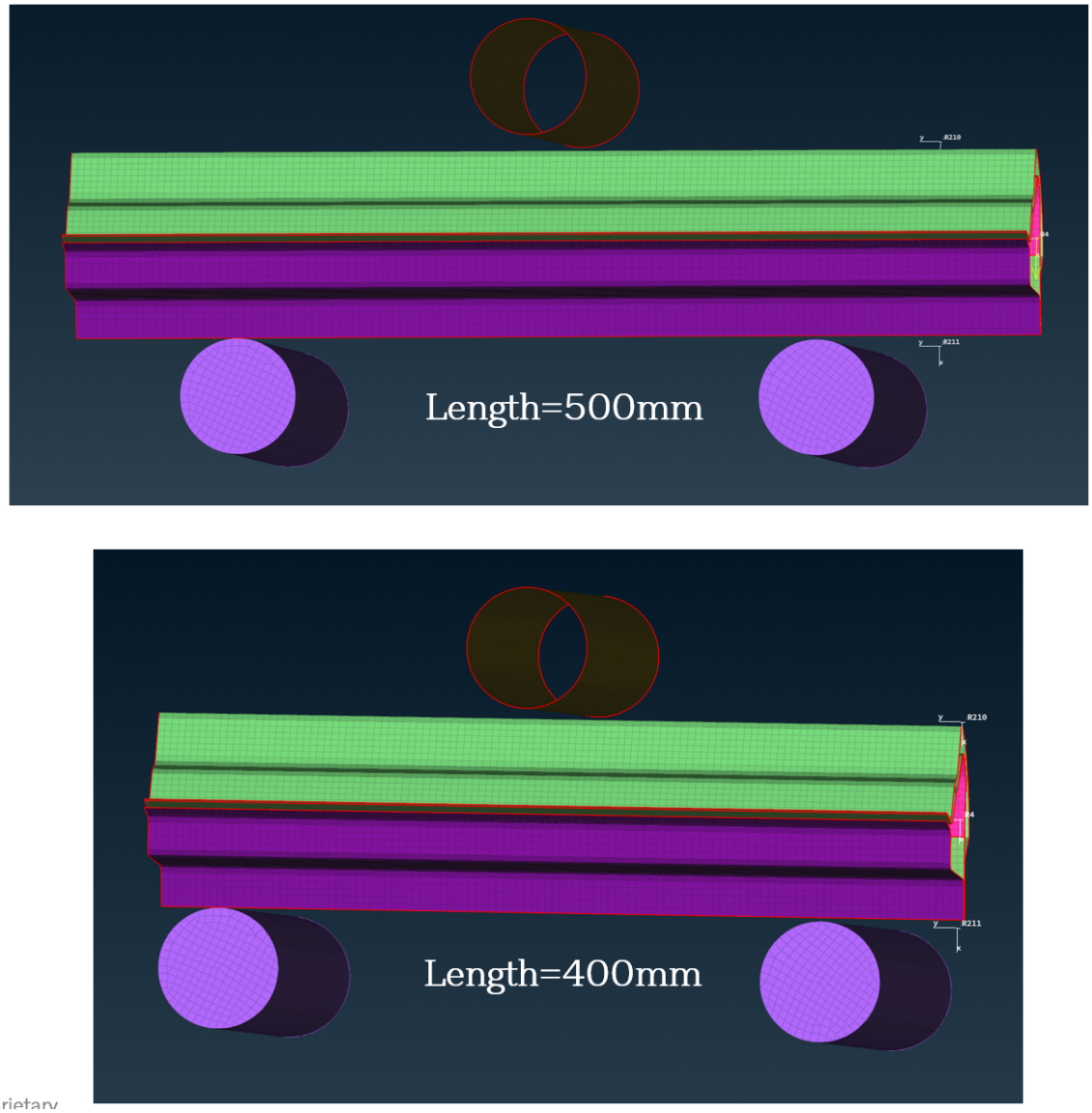

Figure 3.7: Two beams with lengths of 500 mm and 400 mm, demonstrating a comparison of different beam lengths.

Why is this important? If the beam is shortened or lengthened and coordinates maintain static, critical parts may be positioned incorrectly.

The script will locate the keyword *beamlength* within the beam file and subsequently retrieve the corresponding value, which is the length provided by the user in the Damida UI. Moreover, the following equation will be applied to calculate the Y-axis displacement required to recenter the control points :

$$\text{offset} = \frac{L_{\text{beam}} - 400}{2}$$

The upcoming step is that the script applies this latest data directly to the corresponding lines in *ImpactorTranslog.k* and *SupporterTranslog.k*. See the codes below:

```
formatted_y = f"{new_y:>10.6f}"
updated_line = line[:30] + formatted_y + line[40:]
```

This logic ensures geometric consistency when beam lengths vary and proper positioning of displacement control points.

3.7.4 Consolidation into Final Simulation File

Currently, there are two transformed sub-component templates available, one primary simulation file with placeholder tags, metadata containing necessary loadcase and termination parameters, and the beam length aligned key-point coordinates.To put everything together, the merge function *insert_into_template()* is devised.

This function reads the primary template file and looks for special markers, starting with "$ Include transform." At this location, the function attaches *INCLUDE* directives that point to our transformed subcomponent files. Subsequently, the extracted loadcase information from metadata is inserted after the *$ TPBLOADCASE* marker. Other control commands (e.g. termination conditions (CONTROL_TERMINATION)) are also updated and incorporated. Lastly, a new *threepointbend.key* .key file is generated which logs all actions for auditing purposes.

Upon completion, the only thing left to do is submit the file to LS-DYNA. Again, no changes are needed and this thoroughly eliminates the chance for errors in setting up the simulation.

3.7.5 Logging and Output Summary

In addition to the core simulation file, numerous supplementary files will also be created. For instance, a small text document labeled *lcs_values.txt* stores the X-axis direction vector. Such support files, alongside the other auxiliary files, enhance clarity in the overarching logic of the entire program. The provided axis information permit the analysis and visualization tools to bypass parsing intricate simulation input files.

In addition to primary functions, the running script records an activity log of its operations. The actions of applying transforms, selecting offsets, and issuing file outputs are logged. Each operates on a log that may be as simple as a display or more complex and stored in a different file. Users who sense something amiss have the option of retracing every step using these logs until identifying the exact point where the process went wrong.

3.7.6 Extensibility and Customization

Flexibility is integrated into the script from the first day. Users only need to place tags like *$LOADCASE1* and *$LOADCASE2* in the template to signal where they need changes to be made. Need more load cases? Just add the necessary tags, and the script will use them without any further alterations required. Changing a full

suite of complex scenarios from a detailed sanity check to a comprehensive suite becomes not a long, tedious undertaking, but rather a five-second edit.

In addition, the Damida's integration with an auto submission function serves to streamline the scheduling, execution, and supervision of several simulation jobs on VCC company servers. This is collecting control cards and load cases into input decks post-processing which constitutes the final step before jobs are dispatched to LS-DYNA. The relevant coordinates are aligned, values are pulled from the meta-data, and data are formatted as .key files. Thus, every run is a replica of the tests the users intend to execute.

3.8 Post-Processing: Transformation of Reaction Forces

Apart from preparing the model and performing the simulation, now we can extract the final results from the new load case - *Three Point Bending*. Here the most important output signal from the LS-DYNA simulation is the reaction force. This is the force graphed from the beam where contact with the impactor occurs. Such data is written by LS-DYNA into a file called *rcforce* which contains the contact forces resolved globally.

To appraise the beam's mechanical response as well as that of the entire simulation system, it is necessary to translate especially when the components are not aligned with the GCS. In this case, the global reaction forces need to be transformed into the beam's LCS. In this section, one such post-processing technique involving parsing *rcforce* using LCS projection to extract time and force data for plotting will be described.

3.8.1 Purpose of Force Projection

In the default configurations of LS-DYNA simulations, reaction forces are usually extracted at the model's GCS origin. Nonetheless, the vehicle coordinate system determines where the beam is located and how it is rotated. That implies distinct beams have distinct locations and angles of rotation with respect to the GCS. Because the beam's LCS is set as the anchor of the entire system, it is justified to use this LCS as a reference for convenience or due to experimental setup even in this phase.

Another issue to be discussed is the interpretation of the beam's orientation which, if neglected, will result in distorted and worthless results. There are two primary effects resulting from not taking the orientation into consideration. The first effect will be that the reaction force is registered as a blend of global X and Y components rather than the intended local X component. In addition, the changing values of the component's orientation parameters for the successive designs will result in inconsistent global directions of forces.

In order to meet the objectives of generating force-displacement curves, determining peak flexural load angles for bending, and evaluating simulation outcomes across varying design versions, there arises a clear approach. Damida can get the correct axial reaction force along the beam by projecting the global force vector onto the direction of the LCS unit vector, *a_x*.

3.8.2 Overview of Rcforce File

The rcforce file is a standard LS-DYNA ouput file. It contains the following numerical data:

- Timestep or physical time;
- Contact force in all three global X, Y, Z directions;
- Mass;
- Moments in all three global mX, mY,mZ directions.

See Figure 3.8.

```
interface number,         1,is single surface-resultants are undefined
interface number,         1,is single surface-resultants are undefined
interface number,         2,is single surface-resultants are undefined
interface number,         2,is single surface-resultants are undefined
 slave          99 time 0.00000E+00  x  0.00000E+00  y  0.00000E+00  z  0.00000E+00 mass  0.00000E+00  mx  0.00000E+00  my  0.00000E+00  mz  0.00000E+00
 master         99 time 0.00000E+00  x -0.00000E+00  y -0.00000E+00  z -0.00000E+00 mass  0.00000E+00  mx -0.00000E+00  my -0.00000E+00  mz -0.00000E+00
```

Figure 3.8: Rcforce file contains above information.

3.8.3 Parsing Strategy

To successfully implement this function in a Bash shell script. The logic shows as follows:

1. Load the local coordinate direction vector a_x from lcs_values.txt, which was saved in previous pre-processing stage;
2. Initial output array: this function is similar to the original damida function output secforce;
3. Loop through the *rcforce* file line by line to extract time step and global forces from X, Y, Z directions;
4. Project the global force vector onto the local X direction;
5. Store all results in the format x: timestep, y:Flocalx into *rcforcedata*.

3.8.4 Vector Transformationn Method

Below, the key equation used in this section is shown—it projects the global reaction-force vector onto the beam's local axis. Beforehand, the preprocessing step guarantees that the local X-axis has unit length, allowing the calculation to home in solely on the force component acting in the beam's deformation direction.

$$F_{\text{local},x} = \vec{F}_g \cdot \vec{a}_x = F_x \cdot a_x + F_y \cdot a_y + F_z \cdot a_z$$

In the bash shell script, it is coded as:

```
1 Flocalx=$(echo "$xforce␣*␣$ax_x␣+␣$yforce␣*␣$ax_y␣+␣
    $zforce␣*␣$ax_z" | bc -l)
```

3.8.5 Precision Handling and Output Formatting

LS-DYNA usually prints its values in scientific notation, so the script rewrites each value on the fly with printf "%.10f". That tweak does three things at once: the output becomes easier to read, the plotting software stops tripping over tiny rounding quirks, and every line follows the same clean format. The data will then be stored in a comma-separated string.

This design supports real-time plotting in web-based GUIs, which Damida has used for the past two decades in automated simulation report generation. The codes are shown below.

```
1 xforce=$(printf "%.10f" "$xforce")
2
3 echo "Parsed␣RCForc␣data␣in␣local␣coordinate␣system:␣
    $rcforcedata"
```

3.8.6 Use in Engineering Evaluation

One of the most useful spin-off datasets is the reaction force projected onto the beam's axis. It drives three core tasks:

- *Plotting force-displacement curves*: Match F_local x with nodal displacement, and users will have an immediate force-deflection (or stress-strain) trace.
- *Spotting fractures on the fly*: A sudden, sharp drop in that trace often flags yield or crack initiation long before the post-processing stage.
- *Checking simulations against the lab*: Because the force is resolved along the same axis as the physical load cell, the resulting numbers line up cleanly with experimental readings, making benchmarks far easier.

3.9 Summary of Methodology

This chapter encompasses the entire strategy starting from the earliest lines of preprocessing code to the very first slice of post-processed data for an automated three-point-bending (TPB) workflow in LS-DYNA. Accuracy and repeatability were equally important aspects to consider. Regardless, the recipe still maintains enough flexibility to adapt to beams of different dimensions or simulations with peculiar requirements.

Initially, there comes the beam's local coordinate system. Instead of keeping it trapped within the geometry file, the script extracts, reconstructs the direction vectors, and builds an orthonormal basis. With a local frame established, node positions now become scalar projections. Those projections determine where it is safe and useful to drop the impactor and the supports.

Geometry redefinition follows as the next step. The updated coordinates are inserted into template input cards that adhere strictly to LS-DYNA's fixed-width formatting requirements, ensuring compatibility during model import. Once each part is

correctly positioned in its local coordinate system, the **INCLUDE_TRANSFORM* keyword is used to assemble the components into a unified master input deck. Other data also drawn from the geometry file such as beam length, load magnitudes, and displacement.

Automation workflow achieves all in one go and is modular, parametric, and structured for batch executions. It efficiently automates every critical step like geometry handling, coordinate transformation, simulation assembly, result synthesis, and provides unlimited possibility for future add-ons such as optimization loops, test-to-simulation correlation, or even a comprehensive database to catalog every run.

4
Results

This chapter outlines the outcomes of a particular set of test cases that assess the automated simulation program developed in this thesis. Here, interpretation of system responses to a wide range of adjustable parameters, including geometric, material, and boundary conditions is the focal point, as well as whether the logic for transformation, assembly, and simulation setup works in various configuration scenarios.

To organize the validation effort in a meaningful way, several tests have been performed. Out of which, five representative cases were selected. These cases were designed to address the following issues within the given automation framework:

- The precision of local coordinate transformation under rotation;
- Capability and ease to work with various section geometries;
- Degrees of beam length and mass distribution sensitivity;

These cases aim to validate the accuracy of the generated .key files and measure the extent to which the automation workflow fits the requirements of a structural assessment in the concept design phase.

The LS-DYNA explicit solver was used with geometric and material nonlinearities enabled for all simulations. A shell script transformed the raw reaction-force records by remapping each vector into the local coordinate frame of its parent beam. The procedure preserved a uniform physical interpretation, even when the geometry had been rotated for different analysis configurations.

Such systematic data handling, carried out before any prototype testing, confirms that the workflow can quickly scale and adjust without major reconfiguration. Designers evaluating preliminary concepts thus receive a robust simulation environment.

4.1 Case Study 1 – Simple Beam with 45° Rotation

4.1.1 Goal

The particular objective of this test case was to confirm that the developed automation workflow manages geometrical changes in the beams, appropriately, and rotates the beams and responds accordingly. A simple square-section beam was chosen as a prototype geometry for the reason of being the simplest possible shape. To

verify that the transformation logic, contact pose, and load vectors maintain their functionalities in non-orthogonal plane scenarios, the local coordinate system of the beam was rotated by 45 degrees. See Figure 4.1:

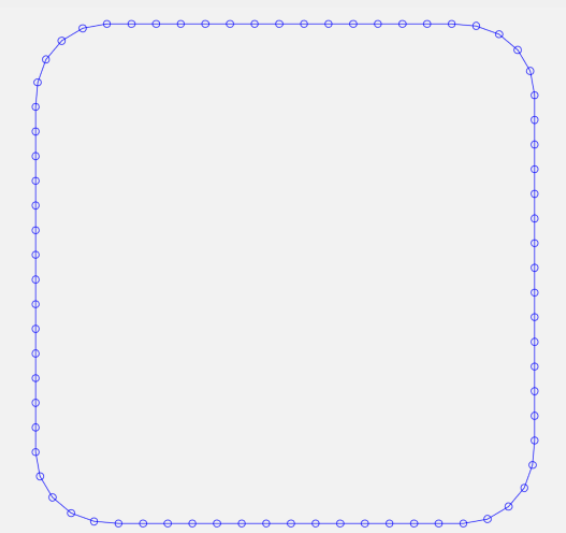

Figure 4.1: A simple beam cross section generated from CATIA.

The goal of this particular test was not to evaluate the response of the structure under loading. Rather, test the **INCLUDE_TRANSFORM* functionality—this feature of the automation would perform the coordinated assembly of the structures in the workflow with transformations from other segments of the working model, to evaluate how far the transformation logic and the assembly framework withstand external perturbations.

4.1.2 Input Overview

The cross-section of the beam is geometrically square, and it was placed at the center of the three-point bending test assembly. As shown in the Figure 4.2, the rotated beam's position was diagonal to the supports and impactor, and this positioning made use of the local coordinate system (LCS) and projecting vectors for accuracy.

- Beam Length: 400 mm
- Shell thickness: 1.6 mm
- Material: DX54 Mild Steel
- Rotation: 45 degrees

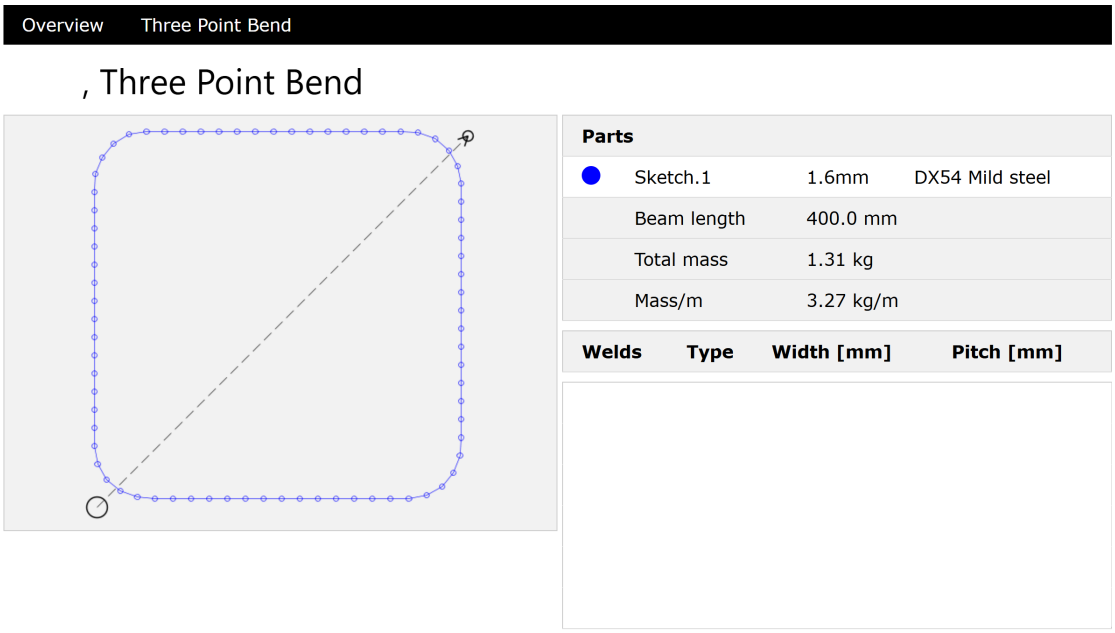

Figure 4.2: The input setting of the simple beam with 45° rotation.

4.1.3 Results

In Figure 4.3, based on the simulation report and the force-time, the simulation appeared to run well without convergence problems or unrealistic contact behaviours. Furthermore, the reaction force curve (RCForce) shows a rise during the initial loading portion of the curve, followed by a brief plateau phase and decay indicating that there is a contact cycle between the impactor and the rotated beam.

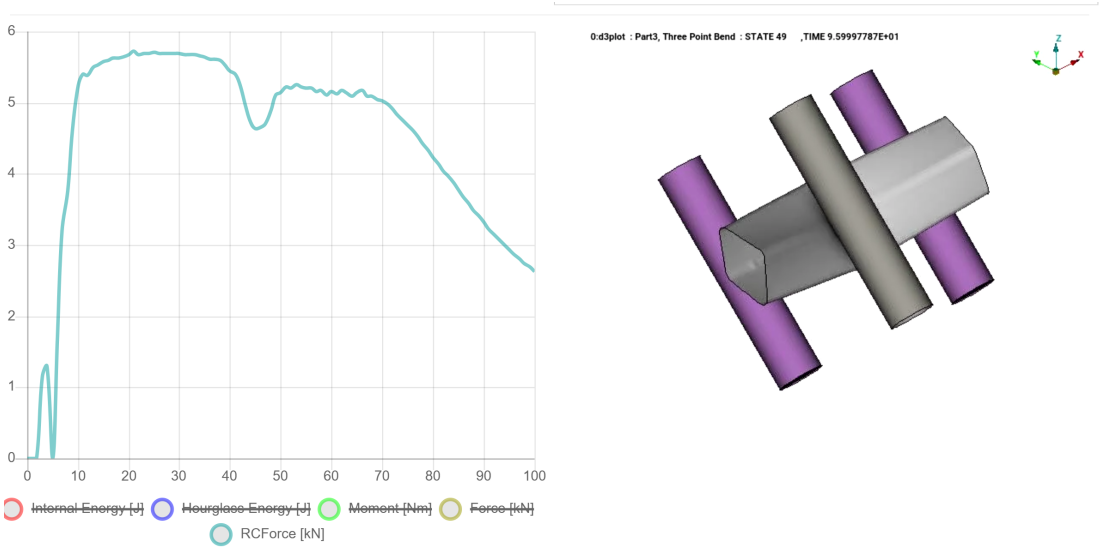

Figure 4.3: The results are extracted from the new Damida of the simple beam.

4.1.4 Robustness Check and Evaluation

This scenario validates the automation workflow's ability to manage arbitrary orientation shifts via a series of local-to-global coordinate transformations.

From a software robustness perspective, this scenario validates the automation script's ability to correctly interpret LCS within the context of three-dimensional solid geometry. It underscores the significance of alignment to orthogonal axes. This consideration becomes critical when modeling real-world structures as described earlier, where alignment with coordinate system planes is not guaranteed.

4.2 Case Study 2 – Beam with Joints and Varying Thickness

4.2.1 Goal

The automation process is evaluated for its ability to process the layer combinations of materials, welded sections, and shells of varying thicknesses in real-world structures. In contrast to the simpler geometries explored in the first case, which utilized a homogeneous beam, this configuration more closely resembles a crash box or a segmented Body-in-White assembly, as it incorporates multiple parts with heterogeneous materials and cross-sectional properties.

In this instance, the heterogeneity of the model includes multiple sections, and the evaluation is centered on whether the automation script completes the tasks of interpreting and transforming the model without losing alignment, contact integrity, or simulation stability across the interfaces of the joints. See Figure 4.4,

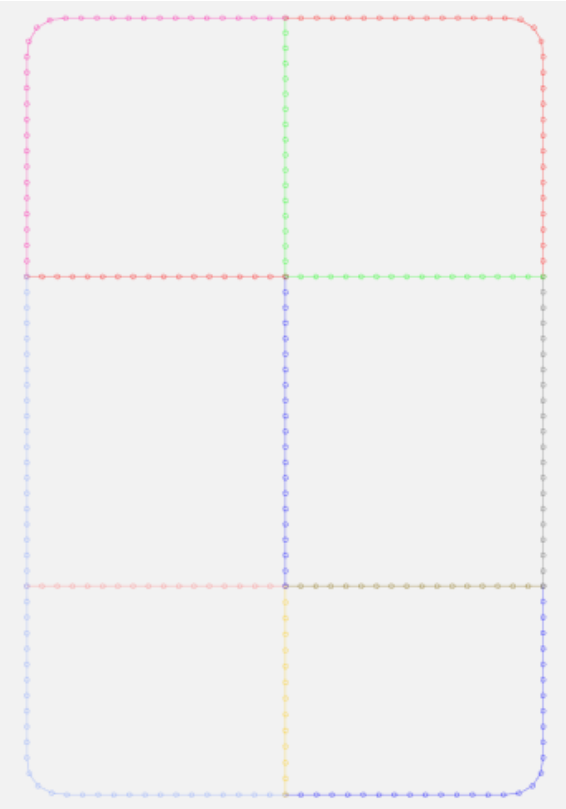

Figure 4.4: Beam with Joints cross section generated from CATIA.

4.2.2 Input Overview

This case was modeled as a 500 mm beam which was split into different sections which will be assigned material and shell thickness individually. The configuration comprises:

- Beam Length: 500 mm
- Shell thickness: 0.5-2 mm
- Material: DX54 Mild Steel and Generic Boron steel
- Rotation: 0

Damida's graphical interface verifies that there are colored categorized subsections as previously stated, see Figure 4.5. This configuration was forwarded using the same program Case 1 was run through, without any manual intervention. This serves as an evaluation of how well the system manages complexity.

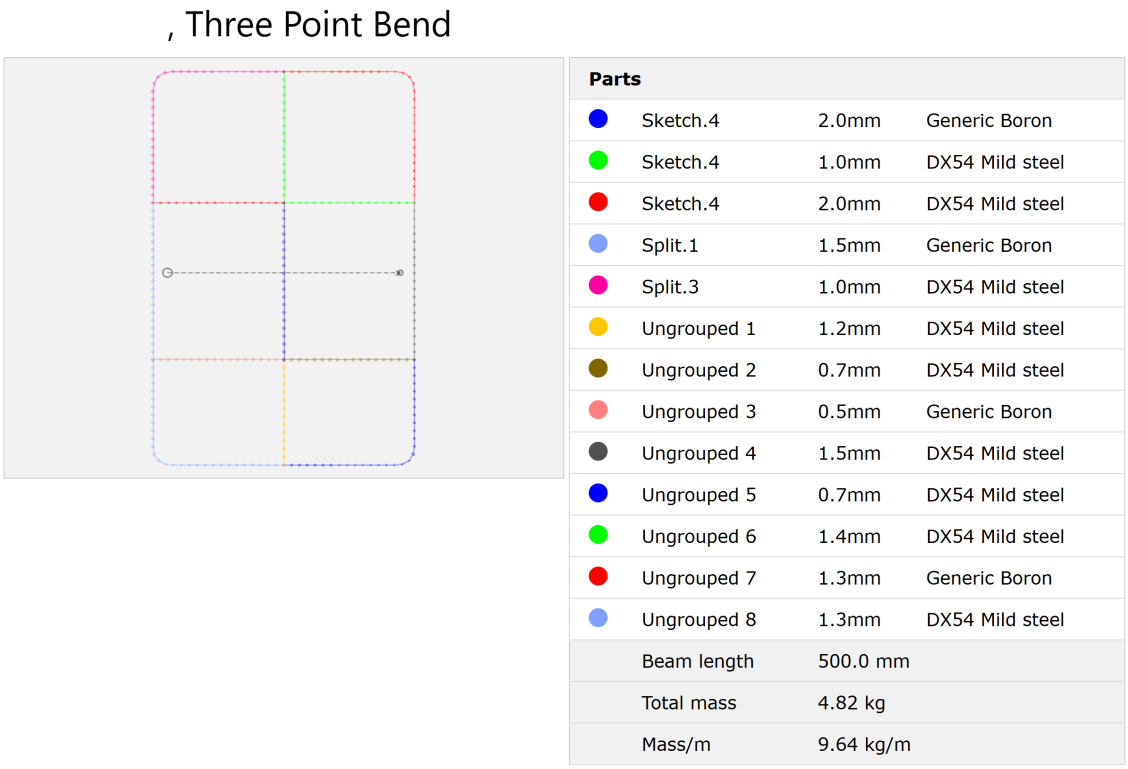

Figure 4.5: The input setting of the beam with joints.

4.2.3 Results

The simulation results indicate full completion, and the Force vs. Displacement curve exhibits a pronounced increase in the initial stage, which stabilises before sharply declining.

From a visual perspective, the model simulation confirms the proper location of the impactor and support placement even with the internal segmentation of the beam. The interfaces were stable and loading appeared to be evenly distributed across the entire assembled structure. There were no observable collapses of elements or non-physical motion near structural boundaries. See Figure 4.6.

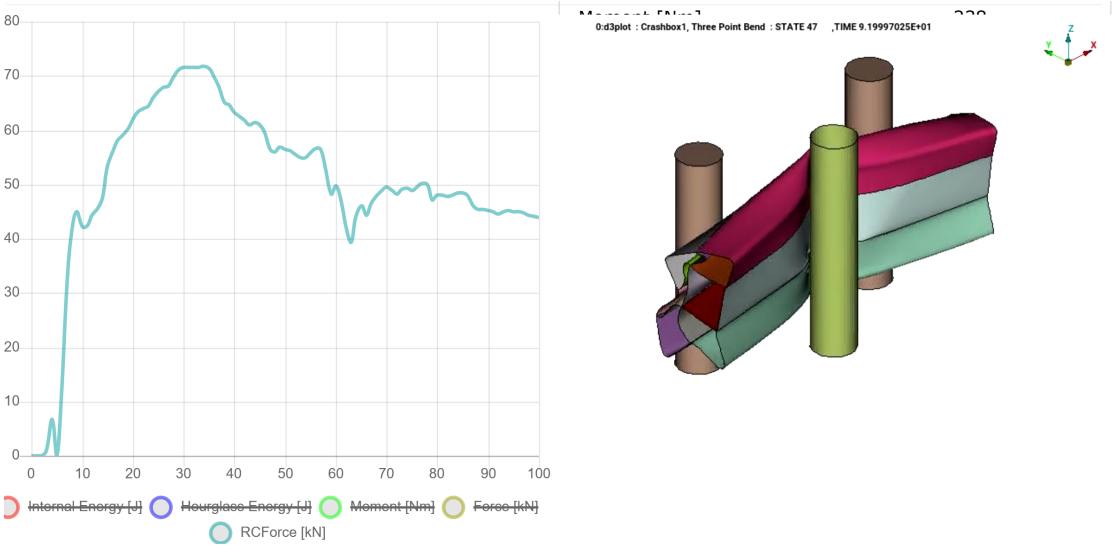

Figure 4.6: The results are extracted from the new Damida of the beam with joints.

4.2.4 Robustness Check and Evaluation

This test case has successfully validated the resilience of the automation framework with respect to the presence of heterogeneous jointed geometries, multiple materials, and varying shell thicknesses. It is further confirmation that the workflow developed does not work only on simplistic or shallow models but rather on complex structures of beams such as are commonplace in industry and which feature frequent segmentation and localized strengthening.

4.3 Case Study 3 – Beam with multiple sheets and no welding

4.3.1 Goal

The scope of the third case was to determine the effectiveness of the program in dealing with non-welded, multi-segmented curved beams. These beams are representative of an early stage conceptual geometry which are often located in the underbody or crash rail regions. Unlike the previous case which assumed multiple parts to mean physical connectivity or welds, this configuration is made of adjacent but not welded beam segments which would pose new challenges for the contact definition and deformation continuity, see Figure 4.7.

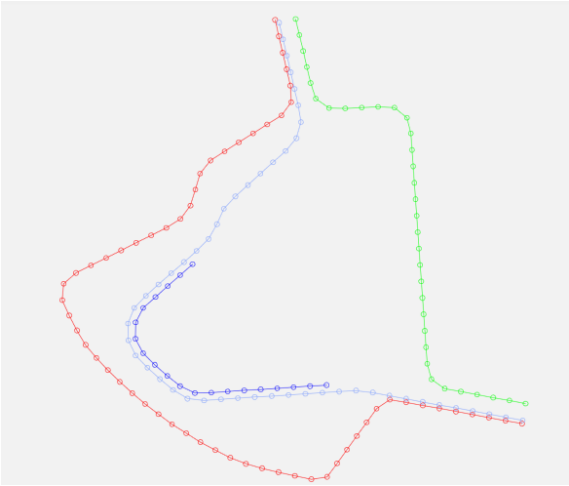

Figure 4.7: Beam with sheets cross section generated from CATIA.

4.3.2 Input Overview

In this test case, the sample beam was made up of four curved segments which were separately defined. Each segment had custom assigned shell thickness and material properties, see Figure 4.8. The configuration comprises:

- Beam Length: 500 mm
- Shell thickness: 1.2 mm and 1.6 mm
- Material: DX54 Mild Steel and Generic Boron steel
- Rotation: 0
- No weld spot

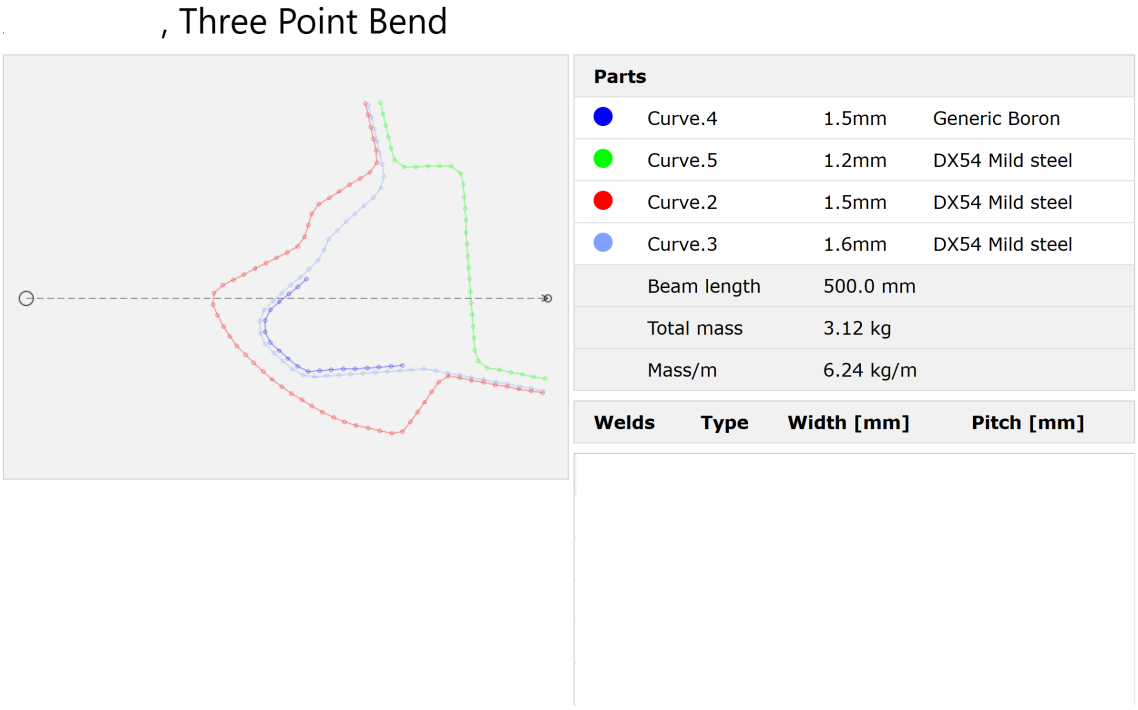

Figure 4.8: The input setting of the beam with sheets.

4.3.3 Results

Even though there were no physical connections or spot welds to hold the segments together the simulation was successful. The contact interfaces to the beam and impactor/supporter maintained motion and stability over time, and the reaction force curve exhibited progressive, multi-phase deformation behavior consistent with structural stiffness variability of the components, see figure 4.9.

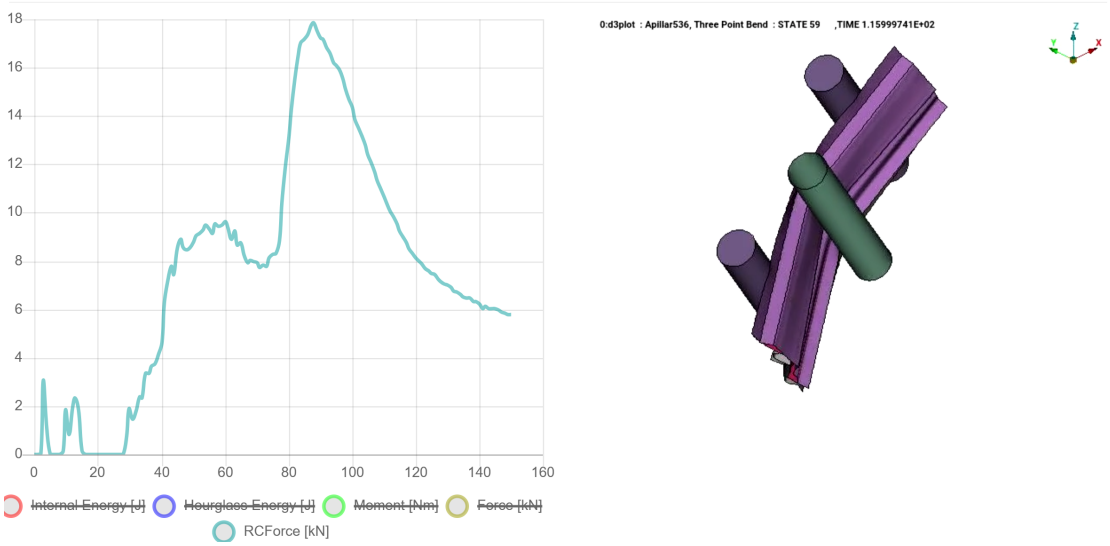

Figure 4.9: The results are extracted from the new Damida of the beam with joints.

The right part of the image on deformation shows the rightmost position where the curved beam segments underwent individual deformation yet acted as a whole to the global deformation response. In absence of constraints and without welding, some inter-segment spacing was noticed due to the unbounded nature of the structure, which is anticipated.

4.3.4 Robustness Check and Evaluation

This outlines the capability of the automation framework to manage non-welded multi-segment beam structures with changes in thickness and material composition while maintaining precision in contact interaction simulation and response to applied load in dynamic situations. The tool is appropriate for very preliminary design phases when components are considered independent volumes awaiting merging.

4.4 Case Study 4 – Robustness Check with Short and Long Beams

4.4.1 Goal

This dual-case test was set up to evaluate the automation simulation workflow's robustness and generalization by applying it to two opposing geometric extremes: a very short beam and a very long beam. While these cases do not typify design specimens, they help investigate the edge workings in the logic for transformation, placement of components, and the solver's stability under unusual boundary conditions. This case study is meant to test the upper and lower scaling limits without resulting in script failure and analyze whether nontrivial physical results (results like penetration, global bending) arise from these inputs.

4.4.2 Input Overview

There are two test models in this case study. For their cross-section, they use the same one as Figure 4.1. For short beam input information, see below and Figure 4.10:

- Beam Length: 200 mm
- Shell thickness: 1 mm
- Material: DX54 Mild Steel
- Rotation: 0
- Total mass: 0.41 kg

Because of the very small length when compared to the fixed support spacing, the beam is effectively shorter than the span between the supporters. This was done on purpose in order to induce a penetration or numerical inconsistency, thus testing the model's behavior with infeasible configurations.

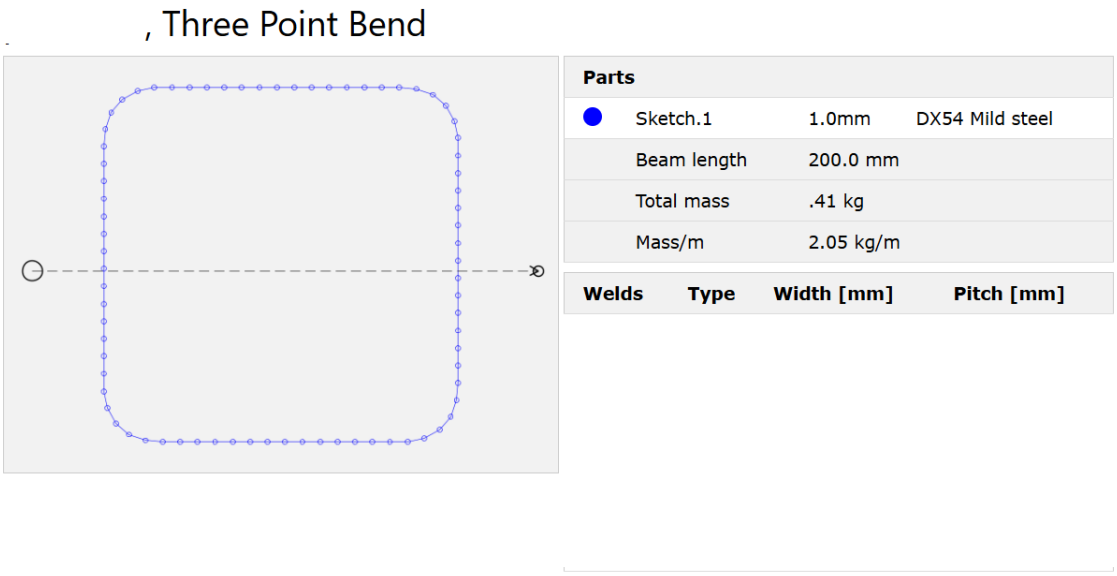

Figure 4.10: The input setting of the short beam.

For the long beam, see the information and Figure 4.11 below:

- Beam Length: 5000 mm
- Shell thickness: 1 mm
- Material: DX54 Mild Steel
- Rotation: 0
- Total mass: 10.22 kg

On the opposite end, the beam was varied to a length 10 times greater than the 500 mm standard case. This is to check if the component's placement, interaction criteria, and .key file logic scales properly for high-aspect-ratio scenarios.

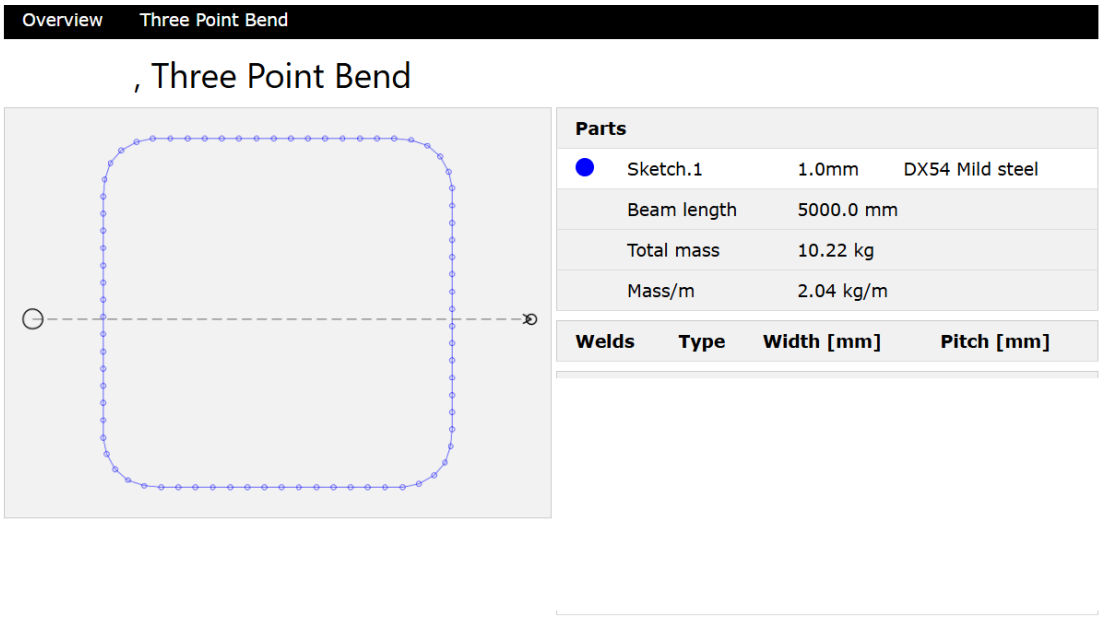

Figure 4.11: The input setting of the long beam.

4.4.3 Results

Figure 4.12 shows the result from a short beam. As shown in the graphics, the force curve (shown in the bottom right corner of the figure) is flat and near zero. This further confirms that no significant resistance was developed during the simulation. The beam is shown to fully cut through the supports without any form of contact which both physically and numerically makes sense. Thus confirming that the simulation automatically accounts for the lack of contact during the geometric alignment fail and the automation program runs safely without crashing, even when the conditions provided yield trivial results.

This behaviour illustrates a robust failure tolerance: stable error reporting where the system fails, but the failure is devoid of physical meaning. In applied contexts, this allows large batch processes to continue uninterrupted even when invalid setups are present.

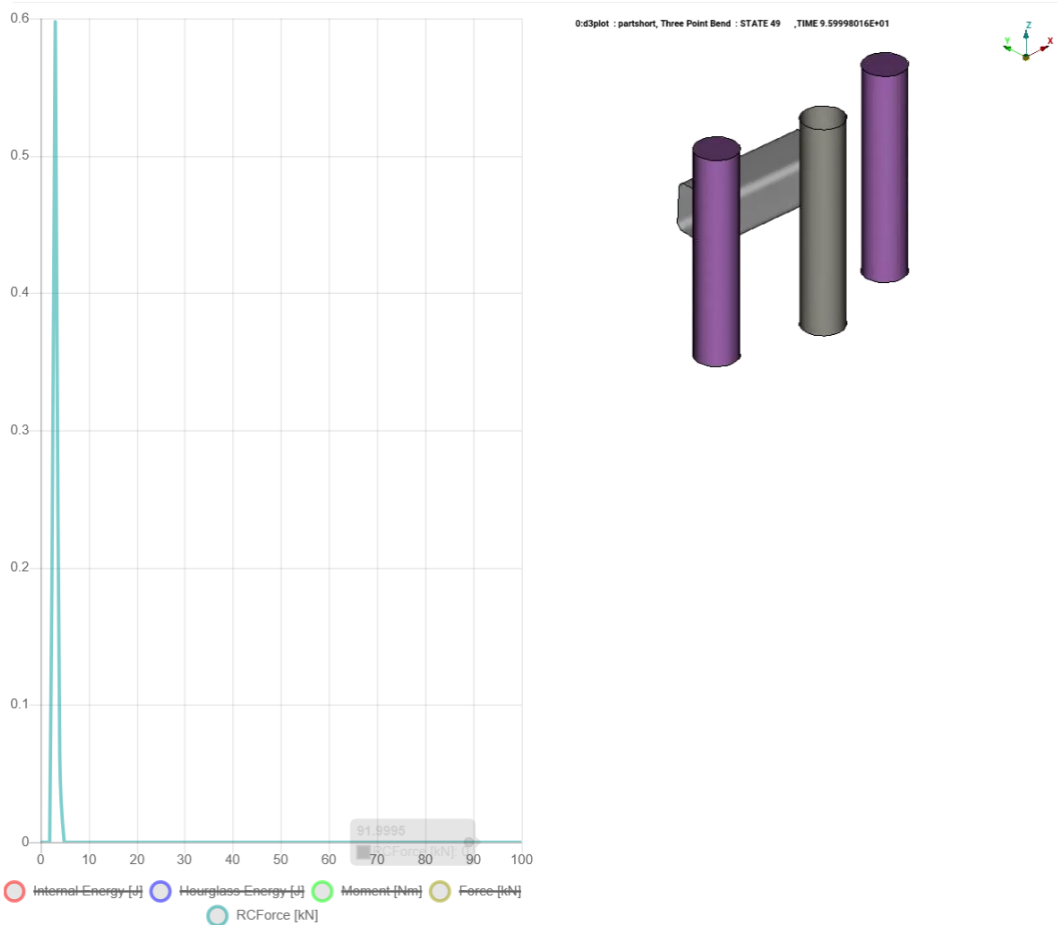

Figure 4.12: The results are extracted from the new Damida of the short beam.

In Figure 4.13, this picture shows the long beam result. The force curve's monotonically increasing nature indicates that engagement is stable and no contact slip occurs. The deformation plot shows a long, monotonically increasing curve without

any unrealistic numerical spurious artefacts.

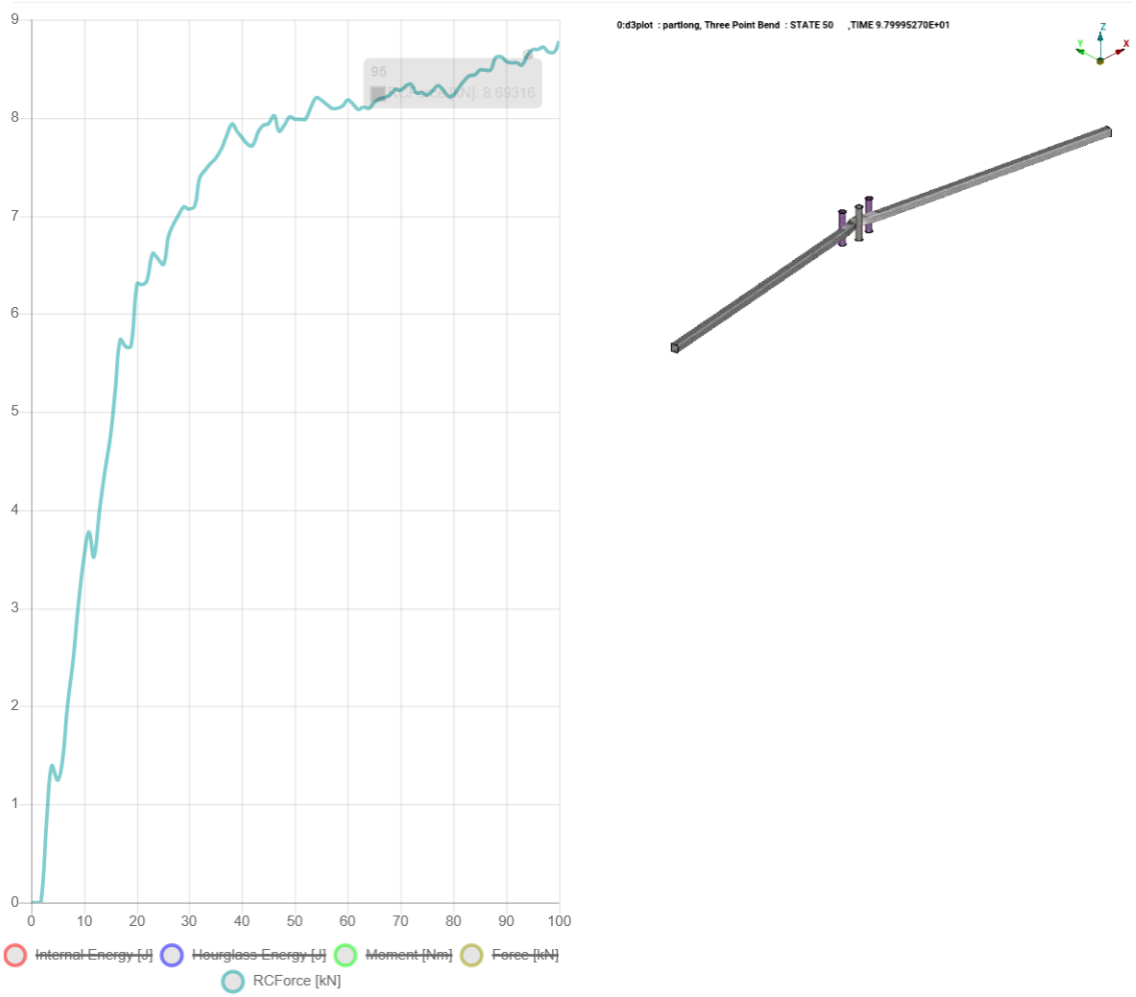

Figure 4.13: The results are extracted from the new Damida of the long beam.

What is noticeable is the fully automated centering of the impactor and support systems along the beam's length, despite the beam's great length. The system accurately computed projections of the beams, calculated the required offsets, and placed all the parts along the long span which illustrates good scaling.

5
Conclusion

As vehicle development cycles continue to shorten, the automotive industry faces increasing pressure to deliver designs that are lighter, safer, and more cost-effective. This makes simulation-based design invaluable. Between geometric design and structural validation, there is a gap where traditional full-body finite element models are not available or are impractical during the concept development phase. This thesis fills that gap by creating and validating an automated simulation framework designed for early-stage Body-in-White (BiW) concept evaluation.

The primary objective of this work is to develop an automated workflow that enables the rapid construction and simulation of beam concepts within a controlled environment, specifically under the conditions defined by the Three-Point Bending (TPB) load case. The workflow emphasizes a modular design approach: it modifies the geometry at the component level using local coordinate systems derived from Damida's mesh generation function, adjusts the relative position of rollers with respect to the beam model, and automatically generates LS-DYNA input files. This process eliminates the need for additional manual steps beyond setting the core simulation parameters.

Unlike traditional CAE workflows that rely heavily on manual meshing, alignment, and keyword setup, this solution automates the entire process through the use of standardized templates. It uses LS-DYNA's **INCLUDE_TRANSFORM* functionality to retain modularity of the models while allowing for some customized spatial relationships between parts. Additionally, it enforces realism in the simulation by means of force projection in post-process visuals, where global contact forces are then projected into the local coordinates of the beam, making interpretation smoother across different geometric relations.

The robustness and performance characteristics of the Python-based simulation process were exhaustively tested and validated with four different test cases, each presenting a unique challenge to the automation logic:

- The baseline square section beam with a 45-degree rotation for testing coordinate transformation;
- A jointed beam increasing in thickness and simulation material to approximate real BIW design features;
- A multi-segment curved beam, testing the spatial continuity;
- Very short and very long beams, testing geometric validity and scale represent edge cases.

Another key advantage of the proposed workflow lies in the modular structure of its input and output files. By adhering to LS-DYNA's rigorous keyword syntax and separating beam, impactor, and support elements into distinct components, the system can accommodate a variety of test scenarios and enables easy substitution of different test setups. This allows the tool to shift from being exclusively a concept exploration aid to an early validation asset in design refinement.

That said, limitations are present. While TPB contact conditions have been managed successfully, extension to more advanced interactions (multi-point contacts, rolling supports, or multi-impactor scenarios) would require further refinement.

Every project confronts certain boundaries, yet this thesis still fashions a meaningful advance in the automated evaluation of structural concepts. Rather than locking the insights into a single program, the work sketches an extensible end-to-end workflow that future developers can weave into larger vehicle-development routines.

Within the proposed framework, engineers can score Body-in-White component designs at an early stage at the click of a button. Because the automation program does most tasks behind the scenes, simulation begins steering design choices while they are still conceptual. Given more tuning and polish, the methodology is well-positioned to anchor the next generation of fast digital-vehicle design.

Bibliography

[1] Holweg, M. (2008). *The evolution of competition in the automotive industry*. Build to order: The road to the 5-day car (pp. 13–34). Springer. https://doi.org/10.1007/978-1-84800-225-8_2

[2] Bylund, N., & Eriksson, M. (2001). Simulation driven car body development using property based models. SAE Technical Paper Series, 2001-01-3046. Society of Automotive Engineers, Inc.

[3] Belytschko, T., Liu, W. K., & Moran, B. (2000). *Nonlinear finite elements for continua and structures*. Wiley.

[4] ANSYS, Inc. (2023). *Implicit*. LS-DYNA Knowledge Base. Retrieved June 25, 2025, from `https://lsdyna.ansys.com/knowledge-base/implicit/`

[5] Meschke, G., & Wagenknecht, S. (2015). *Automotive lightweight design: CAE methods in early-phase body development*. Vehicle System Dynamics, 53(1), 74–92.

[6] Hughes, T. J. R. (2000). *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Publications.

[7] Cook, R. D., Malkus, D. S., Plesha, M. E., & Witt, R. J. (2002). *Concepts and Applications of Finite Element Analysis (4th ed.)*. Wiley.

[8] Predictive Engineering. (2017). *LS-DYNA analysis for structural mechanics* [Training presentation pp. 18]. Predictive Engineering, Inc.

[9] Hallquist, J. O. (2007). *LS-DYNA Keyword User's Manual (Version 971)*. Livermore Software Technology Corporation.

[10] Zienkiewicz, O. C., Taylor, R. L., & Zhu, J. Z. (2013). *The Finite Element Method: Its Basis and Fundamentals (7th ed.)*. Elsevier.

[11] Volvo Cars. (2020). *Volvo Safety Vision*. Retrieved from `https://www.volvocars.com/intl/safety/culture-vision`

[12] Gao, X., & Li, C. (2014). *Development of a unified CAE concept model for vehicle crash analysis*. International Journal of Automotive Technology, 15(2), 233–239.

[13] Tran, H., Furrer, D., & Malas, J. C. (2021). *Enabling efficient engineering through CAE process automation*. Computers in Industry, 127, 103396.

[14] Gilson, R. D. (2019). *Automating finite element simulations with Python: A practical approach*. Engineering with Computers, 35(3), 835–849.

[15] Wang, J., Li, Y., & Fang, J. (2020). *LS-DYNA based modeling and validation of aluminum extrusion crash components*. Thin-Walled Structures, 149, 106633.

[16] Chen, Y., & Huh, H. (2018). *Automatic modeling of LS-DYNA simulation model for impact analysis*. International Journal of Automotive Technology, 19(4), 647–654.

[17] Tseranidis, S., & Papazoglou, A. (2013). *A parameterized LS-DYNA simulation environment for automotive safety applications*. Simulation Modelling Practice and Theory, 36, 121–137.

[18] Anderl, R., & Mendgen, R. (2000). *Modelling with modules and interfaces in early design stages*. Computer-Aided Design, 32(5–6), 323–331.

[19] Kang, S., & Lee, K. H. (2015). *Template-based FE modeling for modular crash simulation*. International Journal of Automotive Technology, 16(4), 683–690.

[20] Ghadimi, P., & Torkamani, M. (2020). *Failure prediction in 3PB test for thin-walled structures using explicit FEM*. Thin-Walled Structures, 150, 106688.

[21] Lan, F., & Wang, H. (2012). *Numerical analysis of fracture behavior of metallic sheets in three-point bending*. Engineering Fracture Mechanics, 91, 78–89.

[22] Shen, W., & Huh, H. (2017). *Development of a LS-DYNA interface for automated crash simulation*. International Journal of Automotive Technology, 18(6), 1001–1009.

[23] El-Masri, M., & Louhichi, B. (2019). *Towards a flexible framework for CAD/-CAE integration using Python scripting*. Computer-Aided Design and Applications, 16(4), 647–657.

[24] Feher, L., Varga, B., & Kovacs, A. (2015). *Efficient FE modeling method development for vehicle structure crash analysis*. Periodica Polytechnica Transportation Engineering, 43(2), 73–78.

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY