Automating the Design Optimization of Vehicle Structures

William Bruce McCormack
Clemson University, wbmcc640@gmail.com

Automating the Design Optimization of Vehicle Structures

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Mechanical Engineering

by
William Bruce McCormack
May 2021

Abstract

In typical model-based design of structures, parametric design and finite element analysis are common tools for simulating structural behavior of systems under static and quasi-static loading. While these tools provide significant benefits over physical experimentation, cumbersome to set up and this time-consuming setup may need to be repeated many times while iterating on a design. The designers would therefore benefit from automating this design and analysis process so that they can explore the design space more efficiently or obtain higher performance design alternatives. To take full advantage of the benefits of this automation, it important to make the process as quick and easy as possible. Otherwise, the cost of setting up the automated analysis may exceed the benefits obtained during design exploration and iteration.

This research introduces a template-based approach to the automation of structural design and analysis that simplifies the setup process for certain classes of design problems. The platform for this automation is the process integration and design optimization tool, modeFRONTIER. Through several case studies in the area of vehicle structure analysis and design, it will be demonstrated how templates can significantly reduce the time and effort needed to frame complex structural design problems.

Acknowledgments

I would like to thank my friends and family who supported me through my time at Clemson. I would like to thank Nicholas Couch for motivating me and helping me to expand this research into composite modeling. I would also like to thank the faculty and staff at Clemson for their help and motivation during my time here. Lastly, I would like to thank Nader and Apurva Gokhale from Esteco, for helping me with my research and allowing me to intern over this past semester.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In typical model-based design of structures, parametric design and finite element analysis are common tools for simulating structural behavior of systems under static and quasi-static loading. These tools provide predictions for the behavior of structural systems when experiencing external loads which reduces the amount of physical validation designers must perform.

However, it is cumbersome to set up these tools, especially while iterating on the design. Manual setups often require attention to the defeaturing of the geometry, quality of the mesh, property selection, and set up of boundary conditions. Additionally, many finite element analysis (FEA) software tools require that steps be performed in specific sequences, meaning a changed geometry might require defeaturing, re-meshing, and reapplying the boundary conditions. These steps can quickly become tedious and can limit the exploration of the design space so that designers often choose to limit re-analysis to large changes in the design.

To overcome this, the design process can be automated to increase the scope of the design space exploration. However, setting up these tools can similarly become tedious and costly to perform. The use of automation must be efficient to set up and adaptable to desired changes by the designer. Similarly, it is desirable that the approach for automation applies across a broad range of design problems so that it does not artificially constrain the designers.

This research investigates methods for achieving broad applicability while simultaneously reducing the time and effort for setting process integration and design optimization (PIDO) problems.

A template-based approach to the automation of structural design and analysis is used to simplify the setup process for design problems. Through several case studies, it will be demonstrated that the use of template scripts, methods, and workflows can reduce the amount of work necessary to set up automated design exploration.

1.2 Scope and Focus of Design Methods

Typical model-based design relies on the manual integration of CAD and analytical models. This process can be cumbersome to perform by the designer and motivates the application of automation. Automation of this design process is well established in many forms and this study aims to explore the application for automotive structures. Automation presents the opportunity to reduce cost in searching the design space for desirable design alternatives [1] [2]. Additionally, automated tools present a basis for repeatable designs for designers of varying backgrounds.

Automation of design tools allows the application of optimization to design problems. This requires additional steps in the set up phase, but allows the designer to automate the generation of high performance designs. Setting up an optimization requires careful attention to the inputs and outputs, optimizer selection, objective of the optimization, constraints applied, and initialization of the design space. As optimization is built on a base of automated processes, repeatability can be a strength of many optimization setups and will be explored throughout this work.

This research focuses the application of optimization to the design process of 1D space-frame and 2D monocoque automotive structures. Performance targets for vehicle structures are typically focused on the mass, stiffness, and crashworthiness of the structure. In this research, crashworthiness is out of scope and will be left for future research. When considering stiffness, torsional stiffness of a vehicle is generally the target for design as it greatly influences the overall dynamics of the vehicle. A structure failing to meet its target stiffness may compromise the vehicle's dynamics and comfort. Torsional rigidity is modeled in this research using the displacement of the front wheels or desired frame nodes under a twisting load applied to the front wheels with the rear wheels fixed in all six degrees of freedom. Using this displacement, an angle of twist can be derived, and a final relative stiffness value is determined. Figure 1.1 shows an example setup in Altair's HyperMesh for deriving torsional stiffness. Bending stiffness is often used as a measure of a vehicle's structural stiffness. This is modeled as a downward load located at the passenger's seating, with the fixed constraints

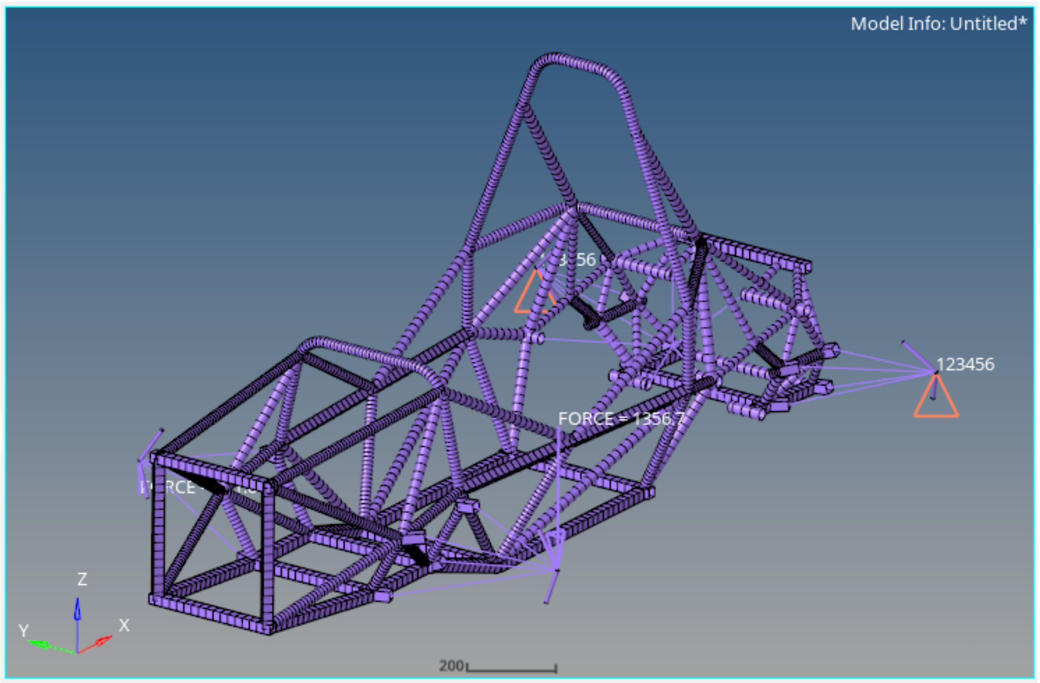


Figure 1.1: Analytical model of structure in Altair's HyperMesh loading in torsion.

at the suspension's mounting locations or by using a faux-suspension. Stiffness is then calculated using the displacement of the structure under the applied load.

Typical model based design for vehicle structures starts with 1D representation of beam sections and 2D representation of sheet material for initial ideation, with final design analysis performed on 3D models. This 1D and 2D modeling of space frame structures allows for faster setup and analysis of analytical models in FEA.

For this research, 1D rigid elements are often used to connect gaps in the structure or substitute desired subsystem components. For example, a double control arm setup might require 1D rigid elements to properly model loading of the frame into the suspension mounts. Figure 1.1 highlights this use in the modeling of an FSAE frame by implementing 1D rigid elements from the wheel centers to suspension mounting points.

1.2.1 Aim of the Research

This study aims to address weaknesses in typical model based design for space frame vehicle structures, by exploring the application of optimization through automation. Automation of the design process can explore the design space more effectively than a manual approach. However, a cumbersome set up process might limit the ability to apply automation to design problems with short design cycles. Additionally, simple design problems with intuitive design alternatives may not be suited for automation. It is therefore important for the designer to understand the strengths and weaknesses of automation.

The exploration of automation in this work presents an opportunity to document strengths and weaknesses that might impact the application of automation to similar design problems. Approaches can be identified to solve weaknesses as they are documented, culminating in a proposed design process for automated design.

1.3 Related Work

Methods and tools for process integration and design optimization have existed for at least thirty years [3]. The following related work has helped to motivate the application of automation in this research.

Automating the simulation of structures through modeling can present challenges to the designer. To simplify this process, Fard *et al.* applies size and shape optimization on a vehicle structures [4] by applying two approaches to simplifying the geometry into 1D elements and modeled joints [5] [6]. This approach to automation takes advantage of simplified geometric modeling to simplify the automation process.

Similarly, Korostelkin *et al.* apply thickness and material optimization to a vehicle structure to reduce mass while meeting stiffness and strength constraints [7]. This work helps to guide the expansion of optimization into more detailed finite element modeling, while implementing the optimization into the development cycle of a vehicle structure concept.

However, to expand the design process beyond the simulation, alterations to the geometry must be automated for use in FEA. A study conducted by Altair in conjunction with A/SP Lightweight SUV Frames Project team [8], explores the use of topology, size, and thickness optimization on different aspects of a structure over varying sections of the design. This project successfully applied optimization to the design process of the structure, however many of the optimizations were split into several sequential optimizations. This motivates the search for optimization approaches that encompass more of the design process including geometry and size and shape optimization, widening the scope of the design space.

To approach the automation of the design process from geometry to simulation, Duddeck and Zimmer [9] explore the implementation of parametrically defined geometry for use in optimization. Their work applies geometric optimization of vehicle structures by proposing robust remeshing techniques for 2D surfaces through a parametric approach. This work highlights difficulties in reliably automating geometric optimization for FEA, as mesh quality and connectivity can easily become inaccurate or decrease in quality. This also motivated the setting up of CAD models for use in automation of 2D surface structures to retain connectivity for later meshing in FEA.

Similarly, Bayatfar *et al.* explored the application of in-house and industry design tools to automate the optimization of a ship hull [10]. This presents a practical example of design automation of a vehicle structure highlights the potential complexity of implementing it and the possibilities automation presents for optimization.

However, as discussed the application of these optimization techniques can be cumbersome [11] with methodologies being proposed to lessen the difficulty in applying optimization to practical examples [12]. Gembarski *et al.* approached the automation of the design process through a knowledge-based design method (KBD), a subset of knowledge-based engineering (KBE), proposing a template-based approach to automation [13]. This approach aims to reduce time and effort in automating the design process given a number of proposed design templates and an example application. This motivated the template-based approach proposed in this research to similarly reduce time and effort in applying automation.

Tarkian [14] describes a proposed multidisciplinary design optimization (MDO) approach to design automation. This work implements design templates in the form of CAD and analysis models combined with a knowledge base and further meta modeling to automate the design process. This work similarly builds off KBE, using a knowledge-based systems approach, separating functions to apply templates from the templates. This motivates the approach later described in Chapter three to applying templates in conjunction with procedures to help guide their generation and implementation. While this approach applies multiple analysis types, this research only focuses on workflows using CAD and FEA modeling.

Dual *et al.* approaches automation through proposed parametric modeling methods. This work automates the design process for front collision beam structures implementing geometry and thickness optimization. This work motivates the use of parametric design methods when developing the proposed template-based approach in this research [15].

Lastly, previous work has additionally helped inform the expansion of the design space beyond homogeneous materials. Galgalikar, Rohan [16] applied optimization to a CFRP sandwich laminate for orientation and core cell density to minimize noise translation. This presents an example of optimization tying third party optimization software to FEA analysis for laminates, expanding beyond traditional material selections. Similarly, Monicke *et al.* [17] applied laminate stacking optimization to a composite cylinder to minimize mass. To achieve this, Altair's ESA Comp software was integrated into the optimization, helping to inform directions for approaching laminate stacking optimization.

To explore multi-objective optimization of FRP components, Awad *et al.* [18] applied thickness and orientation optimization to minimize mass. This work explored unique cost functions for optimization, while using third party software for the FEA and optimization workflow. Similarly, Gao *et al.* [19] optimized a CFRP vehicle component for a crashworthiness load case to demonstrate an approach to composite structure optimization. This work used thickness and orientation optimization to minimize mass, through a third party FEA software, using a novel genetic algorithm. Similarly, Kim *et al.* [20] tied geometry into the optimization for CFRP structures to maximize the buckling load while minimizing mass. This approach helped inform the automation of geometry and FEA for optimization of CFRP structures.

In conclusion, many researchers have used design optimization tools for structural optimization in general and for composite structures more specifically. Where a large portion of literature focuses on the benefits of applying automation to the design process [21], this research aims to address the application of automation. Especially for short design cycle or one-off designs, the time and effort associated with setting up the automation can be a significant portion of the overall solution time. This research aims to guide designers in the application of automation to vehicle structural design problems to reduce time and effort required.

Chapter 2

Design Optimization Method

To address weaknesses in the application of automation, in this research, we develop and characterize methods for the design process using the optimization software, modeFRONTIER. These methods are characterized as sets of procedures which direct the designer to create re-usable design templates for each of the steps of automation. These templates are therefore the final design files such as CAD models, FEA scripts, and modeFRONTIER workflows, created using the respective design procedures. The steps of automation required when automating the design process are the following:

1. Modeling the geometry in CAD to create a parametrically defined template model, specifically set up for use with an automated FEA script.
2. Setting up an FE analysis as a parametric script template, to allow automation of analysis for the previously defined CAD template.
3. Setting up a design optimization workflow template in modeFRONTIER to run the automated design process.

Figure 2.1 presents a general workflow presented in this research, to achieve automation of the CAD and FEA design steps.

By following the procedures proposed in this research, these parametric design templates can be created and edited for new design problems. The creation of these procedures and their resulting templates will be explored in more detail in chapter three, as the design process for a simplified structure is walked through.

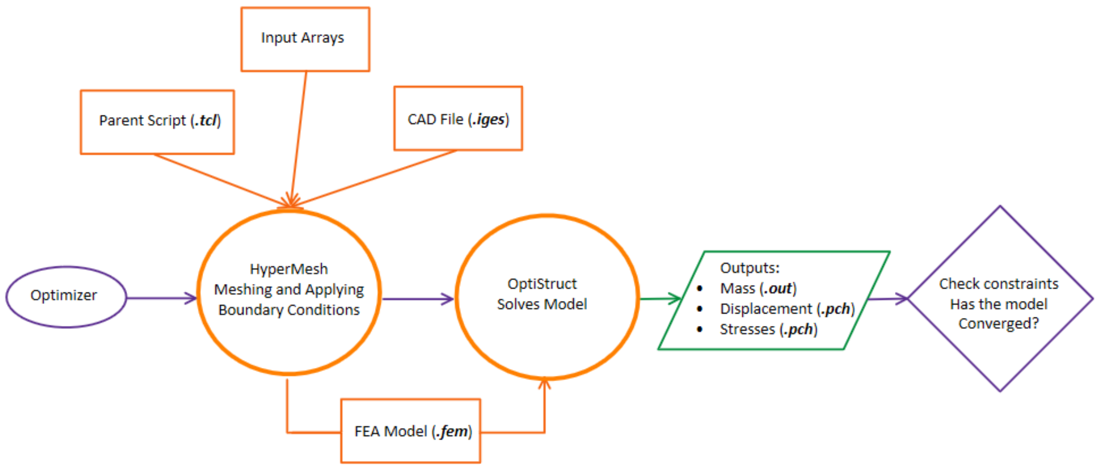


Figure 2.1: Design and analysis workflow.

2.1 Parametric CAD Modeling for Automation

Before describing the creation of a parametric CAD template through its respective procedures, we will review the general methodology used to parametrically prepare the geometry for FEA. For this research, the aim of parametrically defining the geometry is to both allow automation of the CAD's geometry alteration, and to define the geometry specifically for accommodating an automated FEA script.

2.1.1 Parameterically Defining Geometries

A parametrically defined CAD model allows the designer to alter the geometry's dimensions through the alteration of parametrically defined variables. This allows the designer to automate the process of editing a geometries dimensions. To achieve this using SolidWorks, dimensions are defined in the 2D and 3D sketches of the SolidWorks component or assembly using SolidWorks equations. This defines global variables within the SolidWorks model that can parametrically be altered through automated processes in modeFRONTIER.

2.1.2 Grouping Geometries for FEA

To prepare the geometry for an automated FEA script, the geometry is discretized into groups reflecting sections of the geometry that require unique property applications FEA. In this research, we use Altair's HyperMesh, which identifies unique components with sequential ID numbers. Each component can contain geometries and elements, and can have a property card applied to it. These property cards then have a material card attached, to model a specific material behavior (isotropic, anisotropic, etc.). The grouping of geometries allows the following actions from the FEA script:

- Geometries are imported into HyperMesh already discretized into groups as unique HyperMesh components.
- Grouped geometries can be meshed independent of one another, grouping the meshed elements within the same components.
- Property cards can be applied to grouped geometry's component IDs through scripting.
- Elements and nodes can be grabbed for applying boundary conditions, based on their known component ID.

For example, 2D and 3D elements require each new material modeled to have a component containing the elements desired with a unique property card applied, and the desired material card applied to that property card. Additionally for 2D elements in particular, the thickness of the plates are modeled in the property card, meaning each new thickness desired requires a unique component with a unique property card (however material cards can be shared). This is used by the following FEA script in this research's approach to apply properties and identify nodes, based on known component ID numbers.

For 1D elements, the properties are applied to the elements individually through the meshing process. This means multiple element properties can be grouped in a single component, possibly simplifying the grouping process. Each new cross-section requires a new property card and like 2D and 3D elements, each new material requires a new property card. For this research, we then use the grouping of geometries for automated meshing, and not for grouping elements. Elements are all meshed to a single component, while each unique group of beam-sections (with a unique combination of cross-section and material) are grouped by their geometries in unique components. This allows automated meshing of groups of geometries by their beam-section property sequentially using the "displayed" geometry feature.

In summary, the parametric defining of geometries allows the automation of geometry alteration and the preparation of the geometry for scripted FEA. These parametrically defined geometries can be altered and reused as templates, following a proposed set of procedures, with the aim of reducing time and effort to automate the design process of similar structural design problems. Chapter three will go into more detail on the procedures associated with the development of each template while walking through an example design problem.

2.2 Parametric FEA Scripting for Automation

With a parametric CAD model setup for automation, FEA scripts are used to automate the simulation of the structure. This research uses Altair's HyperWorks for FEA with Tcl/Tk scripts used to automate the setup process in HyperMesh. Tcl/Tk scripting can be recorded directly from the graphical user interface (GUI) or scripted directly. This research uses the Tcl/Tk scripting to create a template FEA script by following a set of procedures outlined in chapter three.

The proposed approach to the FEA scripts varies for 1D and 2D elements. For 1D structures, the beam-section's cross sections and properties are created sequentially from an input arrays at the start of the script. These input arrays define many parameters such as the shape, cross-section dimensions, material, orientation of cross-section, etc. Next, the grouped 1D geometries imported into HyperMesh are meshed sequentially with the elements being placed in a new single HyperMesh component. This is performed by initially hiding all geometries then sequentially displaying each group of geometries by their component ID, meshing them applying their respective property card, and hiding the geometry again. As discussed in the previous section, all meshed 1D elements retain their property card application on a element level, assigned in the meshing process. This allows the grouping of all 1D elements in one component to simplify the process of identifying element and node properties for applying boundary conditions based on the component ID that contains them.

To automate the meshing of 2D elements, property cards are similarly created sequentially base on inputs in the FEA script. The geometries are then meshed in one command with all components in HyperMesh displayed. The elements are meshed directly to their respective geometry grouping's component, retaining the grouping of geometries as established in CAD for the desired property application. Property cards are then sequentially applied to each component with the material card selection and desired thickness.

Scripting 2D composites presents a new challenge and is covered in more detail in chapter five. Composite modeling is explored in this research for a composite and core sandwich structure using Altair's classical laminate modeling using plies to model both the fiber reinforced plastic layers and the core layer. The approach to scripting the modeling is based on the approach used for 1D elements with input arrays defining the plies within the composite and their orientation, material, thickness, and geometry-grouping they are applied to. This is tied with a new function based on the 1D meshing function to allow parametric FEA scripting of composite modeling in the same fashion as 1D elements.

Following the meshing processes, the elements are cleaned to eliminate duplicate nodes and connect the elements across the body. This step can require debugging manually to ensure robust meshing of the structure. Steps can be taken in CAD to reduce cleaning difficulties such as ensuring geometries have identical dimensions along intersections and edges.

Next, 1D rigid elements can be created to model rigid connections or subsystem components outside of the scope of the analysis. These rigid elements transfer all movement and loading directly from one coordinate to another in the desired degrees of freedom.

Lastly, boundary conditions are be applied, using Tcl/Tk functions to identify nodes or elements for applying the boundary conditions and functions to then apply the conditions to those identified nodes or elements. Loadsteps are then created for the different modeled load cases and desired output measures. Finally, a solver file is generated as a .FEM script for solving by OptiStruct.

This approach to parametrically scripting FEA allows the development of FEA script templates for application to structural design problems. Chapter three will explore in detail the procedures required for developing these templates. Additionally, following chapters in this research apply these templates to the development of vehicle structures, to explore the strengths and weaknesses of the proposed approach.

2.3 Setting up the Workflow in ModeFRONTIER

Following the CAD model and FEA script, the workflow in modeFRONTIER can be set up. The finished workflow serves as the design template for designers to reuse, guided by procedures described in chapter three. The general workflow can be set up with integrated software nodes, batch scripting nodes, and general workflow nodes such as the optimizer selection node. Inputs are added as either variables or arrays and outputs are defined based on the FEA being run.

To edit the CAD, an integrated SolidWorks node in modeFRONTIER is connected to the parametrically defined equations in SolidWorks to edit the geometry. Options are then selected for outputting the geometry for FEA in the following process nodes.

To edit the FEA script, batch scripting nodes are used which parse the Tcl/Tk script to edit inputs directly before running though a batch script. The FEA process is split into two nodes, one to mesh the model and produce the Fem solver script, and the other to solve it in OptiStruct. To run the solver, a second batch scripting node is used to run the generated Fem file and parse the outputted files form OptiStruct for desired output parameters such as stress, strain, displacement, mass, modal frequencies responses, etc. A Pch file and Out file are generally used for OptiStruct models to measure output parameters. The Out file is automatically generated by Altair and the Pch file is selected for each loadstep defined in HyperMesh, as is allows parsing by text editors which is required for the batch scripting node.

With the workflow created, the last of the workflow nodes can be set up. This can include a design of experiments (DOE) node, an optimizer node, objective nodes for parameters, or constraint nodes for parameters. Here the designer can choose how modeFRONTIER explores the design space.

2.4 Template-Based Approach

With the methodology defined above, this research aims to simplify the application of automation to vehicle structural design. To improve the application of automation, a template-based approach is developed and refined through five case studies in the next chapters. These case studies will illustrate how this approach affects time and effort required to set up structural design problems for automation.

2.4.1 Application of the Proposed Template-Based Approach

To explore the application of automation, Clemson's Formula racing vehicle is chosen for case studies. The vehicle is designed and built for competition in the Formula Society of Automotive Engineering (FSAE) collegiate design competition. This structure presents a unique opportunity for application of automation as it requires a yearly design and build schedule and relies heavily on knowledge transfer and iterative design. This research will apply the proposed template-based optimization to the vehicle structure, working with the chassis division to study the effects on the team's design process.

For the case studies where that alter existing templates to reduce time in setting up the automation, each general alteration will be counted to give a rough estimate of effort required.

This approach to is mainly used to highlight where the majority of alterations are required, while providing a qualitative record of effort required by the designer.

2.5 Summary and Conclusions

Setting up automation of the design process can allow the designer to expand their exploration of the design space, at the cost of an often cumbersome set up. The proposed approach to automation can simplify the process of automating the design process for further exploration of the design space. This approach implements template FEA scripting and modeFRONTIER setups to reduce time needed to set up similar design problems. An accompanying methodology for parameterizing CAD then allows the robust automation of geometries for use in the template-based approach.

This research applies the approach to a FSAE vehicle structure to edit the geometry and FEA setup, before solving and optimizing for the desired objectives. By applying the approach to practical examples, strengths and weaknesses can be identified for improvement. This application is first applied with a limited scope through simple manual recorded scripting of the 2020 FSAE tube frame structure to develop a set of design templates for refined throughout the following case studies.

Chapter 3

Defining the Template-Based Approach

In this chapter, we discuss in more detail the design templates used in this research and the corresponding processes that are used to set up structural analysis and design problems.

In our template-based approach, we approach the introduction of templates similar to previous work by Gembarski et al. Gembarski *et al.* define template-based design as a subset of KBD [13], in which design tasks can be automated to reduce time and effort in the design process. KBD often reuses methods, algorithms, and results for automating the process when setting up a design solution space [22]. Template-based design from a KBD perspective, can be described as a method for creating a geometry. Gembarski *et al.* further break down design templates into three categories: geometric, structural, and functional. Geometric templates can be defined as rigid or variable, where rigid geometric templates are known designs that are well defined and easy to implement. Variable geometric design templates are more complex to implement as they are less defined, but still present a starting point with defined design rules to direct the design process. Structural design templates are described as a product structure, from which different solution spaces can be developed from. Lastly, a functional design template instructs the implementation of design tools and methods, alongside geometric development and descriptions.

Tarkian [14] similarly approaches design automation from a template-based approach using a proposed MDO process. This work approaches design automation using high level CAD and analysis templates, tied with meta-model generation. These high level templates are informed by a knowledge base and stored in databases for reuse and modification. The structure of the automation is informed using knowledge-based systems, closely related to KBD, but with a separate knowledge base and functions which allow use of the knowledge. This use of templates in conjunction with a knowledge base has motivated the development of the proposed template-based approach.

The description of a template-based approach for this research aligns with the above definitions of for design templates, where the instructed implementation of design tools allows the designer to automate the design process using a library of design templates. We therefor define the design templates used in the research as the CAD models, FEA scripts, and modeFRONTIER workflows developed or reused when automating the design process, tied to functional procedures for successful implementation of each. This template-based approach aims to reduce time and effort in setting up automation by presenting a base of CAD models, FEA scripts, and modeFRONTIER workflows to start from, with the accompanying procedures helping to guide implementation or creation of these templates. Iterative design problems benefit from the reuse of these generated design templates, requiring minimal alteration with each application.

3.1 Applying the Template-Based Approach

With the template-based approach described, we can present an example application to discuss the implementation in more detail. In this chapter we will illustrate the proposed template-based approach with a simplified cantilever beam example, automating the design process from an initial topology for beam shape, size, and geometry as well as plate size and thickness. An example geometry meshed in HyperMesh is shown in figure 3.1.

To automate the design process for this cantilever beam, we must follow a procedure for each step of the design process. This procedure allows the generation of each design template used in the automation of the design process. This chapter will define each design step's procedure and implement it on the presented example application.

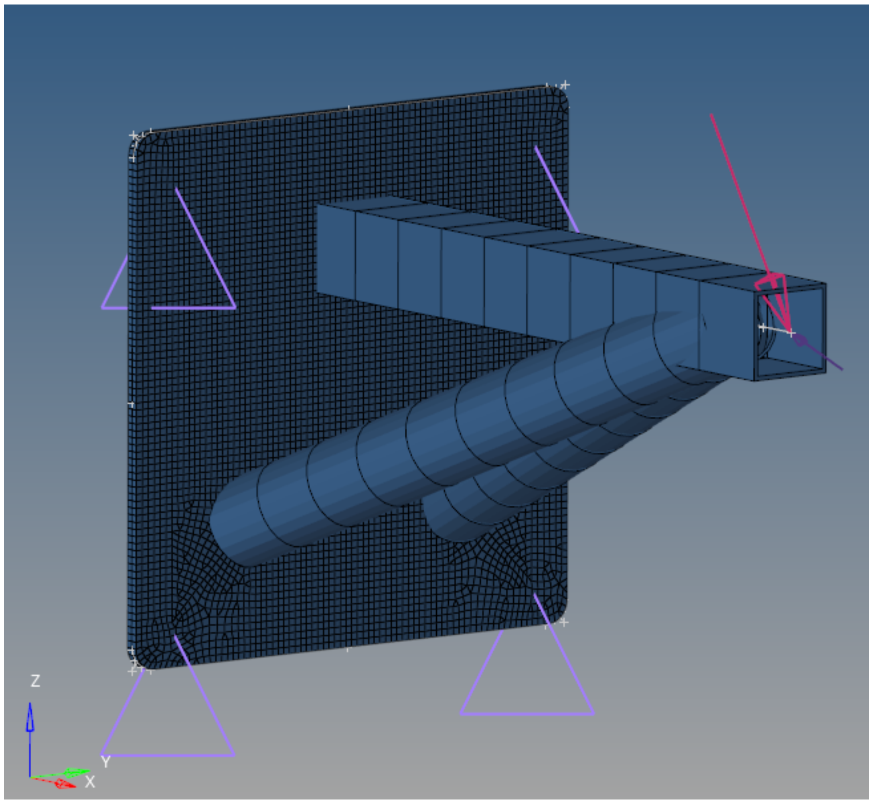


Figure 3.1: CAD of the geometry with the 3D sketches and 2D surfaces used to model the geometry desired for meshing in FEA.

3.2 Parametric CAD Template and Procedure

The first step in the automation of the design process is the creation of a parametric CAD. This model must be prepared to allow reliable alteration of the geometry and exporting to FEA. To achieve these goals, the CAD template must parametrically define the geometry's dimensions and discretize the geometry for use in FEA. To discretize the geometry for FEA, the geometry must be grouped such that geometries requiring unique properties in FEA are imported as separate components in HyperMesh. This allows the proposed FEA scrips in the following step to successfully mesh the geometry, apply properties, and apply boundary conditions.

The preparation of the CAD geometry for FEA can be approached using feature-based design. This approach to implementing CAD has been well established and used to guide the preparation of CAD models for simulation. Shah describes a feature as "a physical constituent of a part, be mappable to a generic shape, have engineering significance, and have predictable properties" [23] [24]. Spiess *et al.* use a feature-based approach to defeaturing components for FEA, applying it to aero engine components [25]. These feature-based approaches help to motivate the proposed approach to preparing a CAD geometry template for FEA. This includes the grouping of geometries, splitting geometries at desired nodes, and defeaturing of geometries. To successfully create this parametric CAD template or alter an existing one, the following procedure is described and applied to the example model.

Step 1 - Determine the approach to grouping geometries for exporting in FEA. This can be through a part or assembly file.

Step 2 - Defeature the geometry if desired to reduce difficulty in setting up an FE model.

Step 3 - Discretize the geometry for reliable meshing through an automated FEA script.

Step 4 - Parameterize the discretized geometry in CAD using dimensions for reliable automation. For SolidWorks this requires using SolidWorks Equations.

Step 5 - Define bounds for the design space dimensions.

Step 6 - Debug the geometry for changes in the parameters within the bounds of the design space.

For the presented cantilever beam example, the dimensions are defined as the mounting hole width and height (with the holes defeatured for simplification), and the length of the cantilever beam in the x-axis. The CAD model created is shown in figure 3.2. The bounds for dimensions are set by the design constraints using reasonable selections for dimensions without a hard constraint.

This geometry is setup in a part file to simplify the setup in CAD, not requiring a top-down assembly. The tube geometries are then modeled as 1D lines in separate 3D sketches in SolidWorks for each unique beam section, discretized at the nodes where each beam section meets to ease merging nodes after meshing. The use of multiple sketches allows each geometry to be grouped as components when imported into HyperMesh. The plate is then modeled as 2D surfaces, similarly split into two surfaces where nodes for the beam sections meet and where the mounting holes are defined. By having the two surfaces meet at a corner where the mounting holes are desired, we can ensure a node will be present after meshing and used for applying boundary conditions. Otherwise, when cleaning the meshed elements, the nodes are likely to move to meet one another, breaking the modeled geometry. Lastly, it should be noted two sketches are used to define the geometry, that are not being meshed. These are considered reference geometries and will be skipped when meshing through the FEA script.

Parameterization is then completed by defining equation driven dimensions in SolidWorks

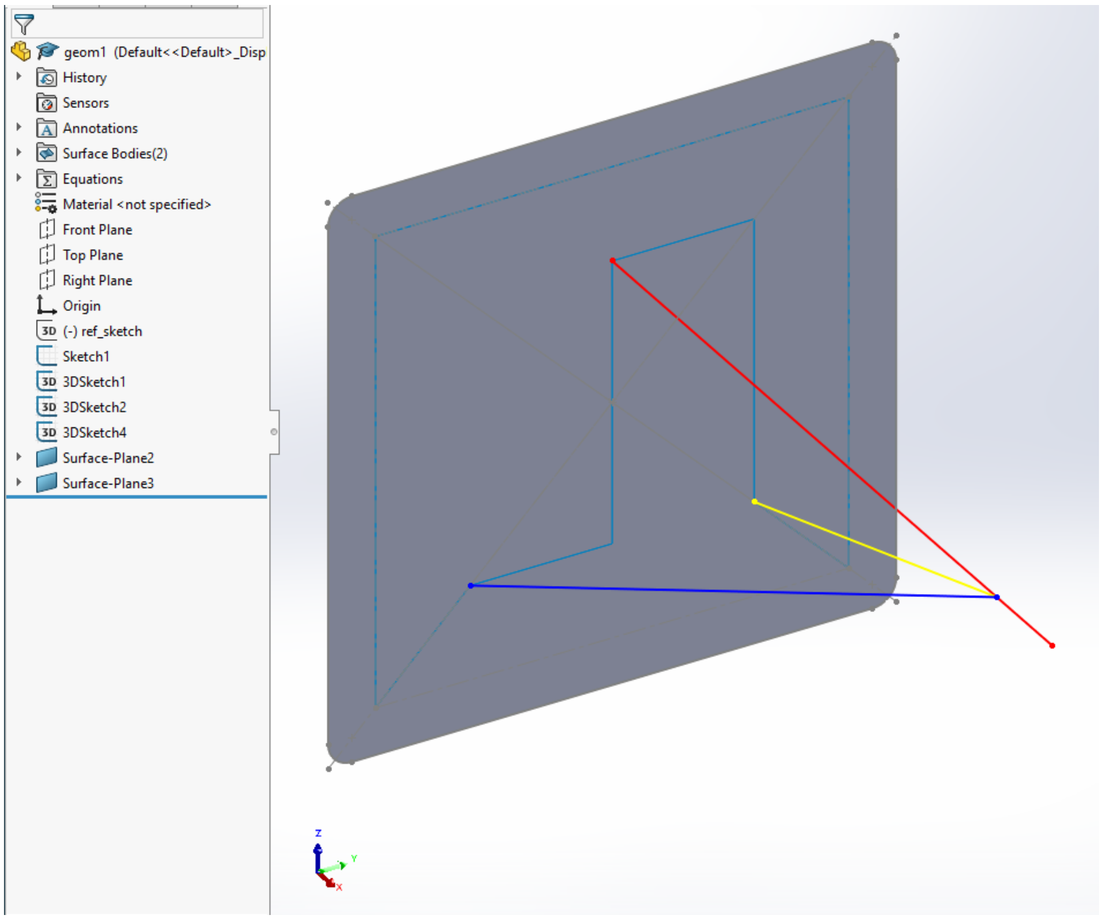


Figure 3.2: CAD of the geometry with the 3D sketches and 2D surfaces used to model the geometry desired for meshing in FEA.

for the desired design space dimensions. With these steps followed, a parameterized geometric CAD template can be used for automation. This will allow the generation of new geometries through modeFRONTIER for exporting into CAD.

3.3 FEA Scripting Template and Procedure

Like the parametric CAD template, a procedure is used to generate a parametric FEA template, using Tcl/Tk scripting. The procedure is based on an existing Tcl/Tk template using the following parts (Note: the specific order of these parts is not necessary to achieve the same results, but is recommended for consistent standardization of scripts):

- **Part 1**: User-Defined Inputs
- **Part 2**: Defining Functions for Repeated Scripting Patterns
- **Part 3**: Importing and Preparing Geometries
- **Part 4**: Automation of Meshing and Application of Element Properties
- **Part 5**: Application of Rigid Elements
- **Part 6**: Application of Boundary Conditions

3.3.1 Part 1 Procedure - User-Defined Inputs

The most important aspect of the proposed approach is the focus on parametrically defining each step. This allows the template to be applied to a variety of use cases where the design space and the size of geometries change. For example, a CAD model imported into FEA will likely have more than one geometry grouped into a component. This template will work regardless of the number of components, given the proper setup procedure is followed when parametrically setting up the CAD template. The goal is for the core procedure and layout of the FEA script template to otherwise remain the same as design problems change. To approach parametrization of the user input part of the template, the following procedure is proposed:

- **Part 1.1:** Importing Geometries
- **Part 1.2:** Defining Parametric Input Arrays for Meshing and Property Application
- **Part 1.3:** Defining Parametric Input Arrays for Geometric Variables
- **Part 1.4:** Defining Parametric Input Arrays for Applying Boundary Condition
- **Part 1.5:** Defining Material Properties

3.3.1.1 Part 1.1 Procedure - Importing Geometries

To start with Part 1.1, we will explore an example script for importing the cantilever beam geometry into an FEA model.

```
# Inputted name for CAD geometry and scaling factor
set CAD_geometry "geom1.igs"
set Scale_Factor 25.4
```

This script creates and assigns variables for the geometry file in a working directory ("./") as well as a variable for scaling the CAD geometry by 25.4 to convert it from inches to millimeters.

Next, a variable is introduced to account for geometries within the model that are not used for meshing. In this example, we have one sketch that is used for defining the geometry and will not be meshed. The variable `geom_skip` is created and assigned the value of 2 to account for this single sketch when automating the meshing process in Part 4.

```
# Variable for number of un-meshed geometries
set geom_skip 2
```

This directly sets up the CAD for automated execution of the script in Part 2.

3.3.1.2 Part 1.2 Procedure - Defining Parametric Input Arrays for Meshing and Property Application

Next, we can approach the scripting of Part 1.2, where we define the parametric input arrays for meshing and element property application. These arrays are parsed by modeFRONTIER for automating the inputs, and sequentially executed as part of the template script.

For this example problem, input arrays are setup for the beam sections automating the generation of each beam's shape, outer dimension (singular for round and square tubes), and thickness. For the plate thickness, a single input is defined. However, a similar approach to automating multiple plates can be used with a thickness input array. This requires three arrays in total. However, additional arrays can be added for material selection by material ID number, orientation of square tube cross sections, etc.

For this example, a further layer of complication is presented where the outer dimensions and thicknesses in the design space are not evenly discretized. To account for this, the input arrays for outer diameter and thickness are inputted as indices from 0 through the number of possible selections, n. The custom sizing is shown below:

- Outer diameters (inch): 0.5, 0.625, 0.75, 1.0, 1.125

- Thicknesses (inch): 0.035, 0.049, 0.058, 0.065, 0.095

These arrays present tube sizes directly purchasable from an example supplier with non-uniform step sizes. These sizes can then be indexed by the input array, allowing uniform discretization of the input array for use in modeFRONTIER. For example, if a tube is desired to be 1 inch wide with a thickness of 0.035 inches, the value of that beam's outer dimension variable (or respective index in its input array) must be equal to 3 to refer to the fourth index of the given outer diameters. Similarly, the thickness variable (or array index) must have a value of 0 to refer to the first index of the given thicknesses. This added layer of control allows early consideration of manufacturability, reducing post-processing effort.

The following script is then used to define the mesh sizes and input arrays for the tubes and mounting plate with example inputs that will be later parsed by modeFRONTIER. The plate thickness is updated directly as manufacturing thicknesses are evenly distributed for this example problem. Following input arrays for manufacturability are defined directly in the meshing function in Part 4.

```
# Mesh size for length of 1D elements (mm)
set mesh_size_1D 20
# Shape of beam to test (1 = round, 2 = square)
set beam_shape {2 1 1}
# Outer dimension (OD) for Beam Sections
set beam_OD {4 2 2}
# Thickness (t) beam dimensions
set beam_t {3 1 1}
# Mesh size for length of 2D elements (mm)
set mesh_size_2D 4
# Plate thickness (mm)
set plate_t 2
```

These arrays allow the automation of meshing and applying properties to both the tube sections and plate in the template FEA script.

3.3.1.3 Part 1.3 Procedure - Defining Parametric Input Arrays for Geometric Variables

Next, variables can be created for referring to desired geometries. This is often used for applying rigid elements or boundary conditions to the model. In this example, the nodes representing the hole locations on the mounting plate are required for applying a fixed constraint. To achieve this, an array for each hole's X,Y,Z coordinate is used to allow identifying the node number for each location. The following script was implemented in this example to identify all four hole locations in the FEA model:

```
# *** Creating Geometry Variables for Finding Desired Nodes ***
#Mounting hole dimensions for creating 6 DOF fixed SPC (inch)
set h_w 5 # hole width (inch) updated from modeFRONTIER
set h_h 5 # hole height (inch) updated from modeFRONTIER
# X,Y,Z coordinates, appended by hole # --> holes 1, 2, 3, 4
set mount_holes {}
lappend mount_holes 0.0 [expr {$h_w*25.4/2}] -[expr {$h_h*25.4/2}]
lappend mount_holes 0.0 -[expr {$h_w*25.4/2}] -[expr {$h_h*25.4/2}]
lappend mount_holes 0.0 [expr {$h_w*25.4/2}] [expr {$h_h*25.4/2}]
lappend mount_holes 0.0 -[expr {$h_w*25.4/2}] [expr {$h_h*25.4/2}]
```

This script allows modeFRONTIER to parse and edit the dimension of the hole's width and height using the variables `h_h` and `h_w` to represent the hole dimensions changed in CAD through modeFRONTIER. These are similarly parsed by modeFRONTIER using the same procedure for the single variables in case study one. Arrays are then created to store each hole's X,Y,Z coordinates in order for later automated identifying of the respective nodes at each location.

3.3.1.4 Part 1.4 Procedure - Defining Parametric Input Arrays for Applying Boundary Conditions

Next, the boundary condition inputs can be scripted as arrays for automation in Part 6. Part 6 requires editing by the designer for each unique design problem, limiting the ease of creating user inputs that can span a wide range of loading conditions.

For this example, an input is used to define the number of loadcases for later creation of loadsteps. Second, two input arrays are used to define the magnitude of the forces applied in the X,Y,Z coordinates, as seen below:

```
# *** Creating Loading Variables for Script Organization and Cleanup ***
# Number of loadsteps
set loadstep_no 2
# Creating loading magnitude arrays for X,Y,Z components (loads 1 & 2)
set load1 {200.0 300.0 -1000.0}
set load2 {0.0 -300.0 200.0}
```

Additional scripting can be added to label each load collector and loadstep used by HyperMesh. However, this was omitted for the example as it does not impact the performance of the script.

3.3.1.5 Part 1.5 Procedure - Defining Material Properties

Lastly, the material properties can be defined for parametrization or directly scripted, depending on the complexity of the model. In this example, only generic low carbon steel is used, so it is scripted directly to save space.

```
# Steel property card (E, G, nu, rho)
*createentity mats cardimage=MAT1 name=steel
*setvalue mats id=1 name=steel
*setvalue mats id=1 STATUS=1 1=210000 # GPa
*setvalue mats id=1 STATUS=1 2=80000 # GPa
```

```
*setvalue mats id=1 STATUS=1 3=0.3 # n/a
*setvalue mats id=1 STATUS=1 4=7.85e-09 # tonne/mm3
```

For large multi-material models, scripting can be used to create many material properties using user-defined functions and input arrays.

3.3.2 Part 2 Procedure - Defining Functions for Repeated Scripting Patterns

The following Part 2 is where the designers can implement user-defined functions to simplify specific tasks. For this example, four user-defined functions are implemented. Throughout this research, multiple functions are reused, simplifying the setup of new design problems.

- **Function 1**: `beam_make` which creates each tube-section's cross sections, properties, and meshes the respective geometry.
- **Function 2**: `find_node_number` finds the node number relating the the X,Y,Z coordinates given.
- **Function 3**: `create_SPC` creates "SPC" load collectors in HyperMesh for fixed boundary conditions. This function fixes inputted nodes from an array of node ID numbers for the desired degrees of freedom (entered as a string 123456, omitting numbers relating to degrees of freedom left open).
- **Function 4**: `create_force` creates "load" load collectors in HyperMesh for applied fixed boundary conditions. This function applies forces to inputted nodes from an array of node ID numbers using an inputted X,Y,Z component array to create the force magnitude and vector.

These user-defined functions are provided in full detail in Appendix A lines 85-187.

3.3.3 Part 3 Procedure - Importing and Prepping Geometries

This part of the template's script imports the geometry and prepares it for meshing, using variables names specified in Part 1 to automate the process. To prepare the geometries for meshing, this step hides all geometries so later meshing steps can show only the geometries being meshed at that particular instance. This method of meshing is covered in more detail in Part 4.

For 1D meshing, each 3D sketch in SolidWorks is imported as a unique component in the HyperMesh design tree. Each component of the desired beam sections for meshing is sequentially displayed, meshed with properties, and hidden. This is looped through a for-loop that starts on the first component ID of the first meshed 1D geometry. To account for geometries not being meshed as 1D elements, the counter defined in Part 1 is used to skip these component IDs at the top of the HyperMesh design tree.

for 2D meshing, all displayed 2D surfaces can be meshed directly into a HyperMesh component of choice, by setting it to current. This is scripted by showing all components and meshing all "displayed" surfaces, as no surfaces are present that are not available for meshing.

This part of the script also creates a new component in the HyperMesh design tree for collecting all meshed 1D elements. Meshed 1D elements have properties applied to the individual elements during the meshing process, where 2D and 3D elements rely on properties applied to the components containing the elements. The script for this part, used for the cantilever beam example can be seen in more detail in Appendix A lines 191-241.

3.3.4 Part 4 Procedure - Automation of Meshing

The meshing of the geometry requires minimal editing between different design problems, with the majority of attention required when different dimensional elements are used. For example, the cantilever bar requires meshing of 1D tubes and a 2D plate, requiring two meshing scripts.

Before meshing the 1D elements, cross sections for the tubes and properties for each tube are created sequentially based on the input arrays. These are then indexed when sequentially meshing the 1D elements of each tube-section to their respective geometry by component ID number. The 1D elements meshed as previously stated, by hiding all geometries first and showing only the geometry relating to the desired beam section from the indexed input arrays. This allows the meshing of all 1D curves within a component, regardless of continuity, allowing paring of tubes with the same input parameters.

Similarly, 2D elements have their properties created first. However, the properties are applied sequentially to the surfaces based on their component ID. To automate this, the surfaces follow the 1D element geometries allowing the counter for the 1D element loop to be continued into the sequential counting of the 2D surface geometry component IDs. All geometries can then be displayed and the 2D elements are meshed to all displayed surfaces into their respective component ID's.

This application meshes the 1D elements first, followed by property application for the 2D elements to allow the continuation of the 1D element meshing counter to 2D property application.

However, this can be changed if accounted for by the for 1D element meshing counter and hiding all geometries before looping.

Lastly, the meshed elements are cleaned by connecting all nodes within a desired radius of one another (in this example, we use 0.05 mm to ensure only nodes nearly on top of one another are connected). This ensure connectivity of the model's elements where they might initially be floating independent from one another and is done using the following script:

```
# Cleaning elements for connectivity with 'equivalence' function
*createmark components 1 "all" # marking all components as "1"
*equivalence components 1 0.05 1 0 0 # connecting all nodes within "1"
```

This method of automated meshing scales with input arrays for both 1D tubes and 2D surfaces, simplifying the expansion of the design space. The script for this part of the template used in the cantilever beam example, can be seen in full detail in Appendix A lines 245-287.

3.3.5 Part 5 Procedure - Application of Rigid Elements

This part of the template would normally be where 1D rigid elements are applied to the model. This cantilever example problem does not use 1D rigid elements, so an unrelated example script is shown below. HyperMesh allows the creation of 1D rigid elements (specifically RBE2 type) that translate all behavior from one or more nodes to another in a specified degrees of freedom. This can be done in the following methods:
- Connect node A to node B
- Connect node A to multiple nodes B
- Connects nodes B together and calculate a mid-point for node A between all of the B nodes.

When scripting this, we can use the user-defined geometric variables and the `find_node_number` function to make arrays of node ID numbers for desired nodes at specific coordinates. By doing this, we can create arrays of A and B nodes using a for-loop and connect them using a following for-loop using HyperMesh's `*createmarknodes` and `*rigidlinks` commands. This two step procedure allows the for loops to be copy and pasted and edited for new design problems.

The following example for scripting 1D rigid elements is used to demonstrate a parametric approach to the scripting. For example, the following script is used in the creation of rigid elements connecting each of the four mounting hole node locations in the FEA model (B nodes) to a new singular mounting node (A node) in six degrees of freedom (note: this particular example assumes node A is already created as a variable `$node_A` identifying the node ID number. We first create a for-loop for capturing the node IDs relating to the mounting points (B nodes), specified by the user input coordinate arrays in Part 1. These node IDs are stored in a new array `$mount_hole_IDs` and are used to create a rigid link from the B nodes to the desired A node. The following script is used to accomplish this:

```
# Creating array of node IDs for mounting holes (B nodes)
set mount_hole_IDs  # initializing array
# Looping for all four mounting holes to grad node IDs
for {set i 0} {$i < 4} {incr i} {
      # Counter for X,Y,Z coordinates of each hole
      set j [expr {$i*3}]
      # Identifying hole ID numbers
      lappend mount_hole_IDs [find_node_number $comp_no [lrange ...
            ...  $mount_holes $j [expr {$j + 2}]] 1 0 {1 1 1}]
}
# Creating 1D rigid element between 4 mounting nodes B and new node A
*createmark nodes 2 [lindex $mount_hole_IDs 0] ...
      ...  [lindex $mount_hole_IDs 3] # marking B nodes as "2"
```
NOTE: the line above is shortened, all 4 indexes (0-3) must be called
```
*rigidlink $node_A 2 123456 # Linking node A to the B nodes marked as "2"
```

This is an example of applying a rigid link from multiple B nodes in a mesh to a desired node A. When applying rigid elements in different circumstances, example codes can be created and reused to reduce setup time for new Tcl/Tk scripts.

3.3.6 Part 6 Procedure - Application of Boundary Conditions

This final part of the template applies the boundary conditions to the model, creates the load steps required to solve the load cases, and sets up an output solver file. The output files used in this research are Pch and Out files. Pch files are specified when setting up the loadsteps for output measures such as nodal displacement and peak element stress. The Out files are generated automatically by OptiStruct for every analysis and contain information such as system mass and frequency response for modal analysis. The Phc file format is chosen as it allows parsing by a text editor, as required by modeFRONTIER's batch scripting nodes for identifying outputs. This part of the script requires editing when new outputs are desired and boundary conditions change. However, to reduce the setup time required to approach this, parametric user-defined functions and template scripts are developed using the following procedure:

- **Part 6.1**: Identify node IDs required for applying boundary conditions and into arrays. Note, this is the same method as in the 1D rigid part of the template and can often be skipped if necessary node ID arrays were previously created.
- **Part 6.2**: For each load case, create the required load collectors and apply the desired boundary condition (fixed constraints, forces, etc.) using Part 2's user-defined functions. Inputs for these functions are often the node ID array and boundary condition variables (defined in Part 1 - user inputs).
- **Part 6.3**: Create load steps using a for loop for number of load cases defined in Part 1. Generate new loadstep scripts manually though the HyperMesh GUI as different outputs are required. (Note: to batch script the generated Tcl/Tk scripts for loadsteps, the lines containing "OS" must be deleted first.)
- **Part 6.4**: Edit node or element ID numbers to allow them to be placed at the top of outputted results (results outputted as Pch files order nodal displacement sequentially counting up, making parsing for a desired node otherwise cumbersome).
- **Part 6.5**: Script the solver file generation (.FEM format).

An example script for the following Part 6 can be seen in Appendix A lines 292-501.

3.3.7 Parametric FEA Scripting Templates Summary

The parametric FEA script templates are often the most complex to create and alter for new design problems. This requires the detailed breakdown into parts and an accompanying procedure for successful application. However, following the defined procedure, these templates can be created and altered for different structural design problems, with the aim of reducing the time required to set up the automation of the design process. This is explored further in the following chapters as the proposed template-based approach is applied to the design process of vehicles structures.

3.4 ModeFRONTIER Workflow Template and Procedure

Finally, the modeFRONTIER workflow can be set up as a template following the following procedure:

Step 1 - Defining the workflow nodes.

Step 2 - Creating or editing input and output nodes to match the design space.

Step 3 - Parametrically linking the CAD model.

Step 4 - Setting up Batch scripting nodes using parsing rules scripts.

Step 5 - Selecting desired one or more of the following: optimizer, design of experiments, constraints, and objectives.

Step 6 - Debugging the workflow before running.

These procedures allow the creation or alteration of existing modeFRONTIER templates to automate the design process. For the proposed cantilever beam example, the modeFRONTIER workflow developed can be seen in figure 3.3. This particular example is set up using modeFRONTIER's built in multi-objective genetic algorithm II (MOGA II) with automated initialization of 1,000 iterations.

Here we can see the general workflow for the design process with an optimizer node, followed by the parametric CAD node, a batch scripting node for creating the FEA model ('meshing node'), and finally a batch scripting node to solve the FEA model ('solver node'). Input variables along with an input file for the CAD model are set up leading into the CAD node. Similarly, input arrays are set up above the workflow leading into the meshing node, with outputs attached to the FEA solver node. An example parsing script is shown in figure 3.4. Theses scripts are automatically generated when manually defining a parsing rule and can be easily scripted by copy and pasting,

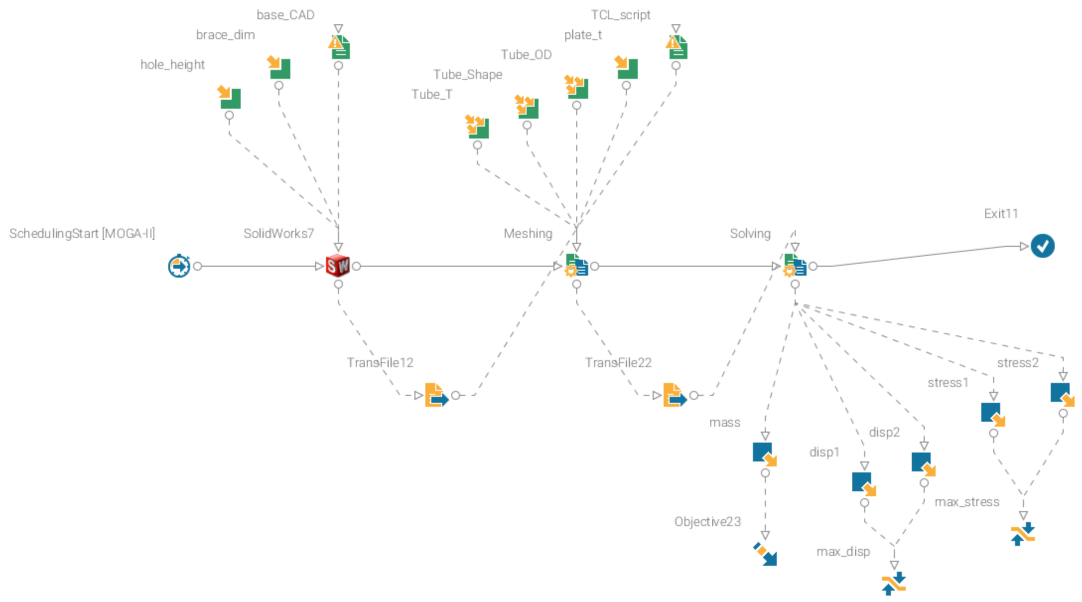
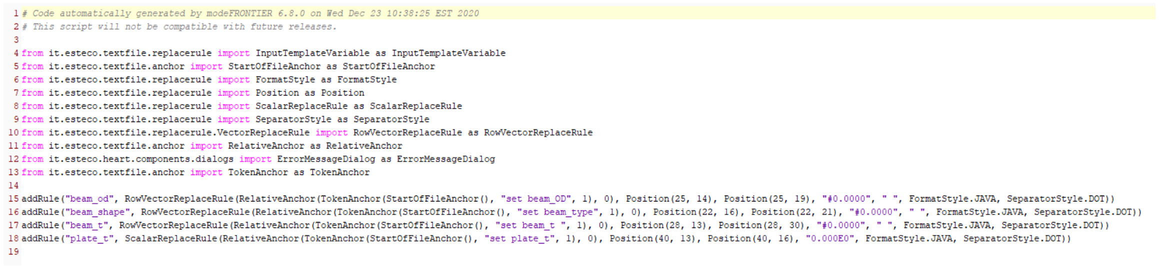


Figure 3.3: ModeFRONTIER workflow for the example cantilever beam example.

while updating the column and row index of the line to reflect the new variable/array being parsed. The solver node is similarly parsed for the outputs using the generated Fem and Out files from one analysis of an example FEA model.



```
# Code automatically generated by modeFRONTIER 6.8.0 on Wed Dec 23 10:38:25 EST 2020
# This script will not be compatible with future releases.

from it.esteco.textfile.replacerule import InputTemplateVariable as InputTemplateVariable
from it.esteco.textfile.anchor import StartOfFileAnchor as StartOfFileAnchor
from it.esteco.textfile.replacerule import FormatStyle as FormatStyle
from it.esteco.textfile.replacerule import Position as Position
from it.esteco.textfile.replacerule import ScalarReplaceRule as ScalarReplaceRule
from it.esteco.textfile.replacerule import SeparatorStyle as SeparatorStyle
from it.esteco.textfile.replacerule.VectorReplaceRule import RowVectorReplaceRule as RowVectorReplaceRule
from it.esteco.textfile.anchor import RelativeAnchor as RelativeAnchor
from it.esteco.heart.components.dialogs import ErrorMessageDialog as ErrorMessageDialog
from it.esteco.textfile.anchor import TokenAnchor as TokenAnchor

addRule("beam_od", RowVectorReplaceRule(RelativeAnchor(TokenAnchor(StartOfFileAnchor(), "set beam_OD", 1), 0), Position(25, 14), Position(25, 19), "#0.0000", " ", FormatStyle.JAVA, SeparatorStyle.DOT))
addRule("beam_shape", RowVectorReplaceRule(RelativeAnchor(TokenAnchor(StartOfFileAnchor(), "set beam_type", 1), 0), Position(22, 16), Position(22, 21), "#0.0000", " ", FormatStyle.JAVA, SeparatorStyle.DOT))
addRule("beam_t", RowVectorReplaceRule(RelativeAnchor(TokenAnchor(StartOfFileAnchor(), "set beam_t ", 1), 0), Position(28, 13), Position(28, 30), "#0.0000", " ", FormatStyle.JAVA, SeparatorStyle.DOT))
addRule("plate_t", ScalarReplaceRule(RelativeAnchor(TokenAnchor(StartOfFileAnchor(), "set plate_t", 1), 0), Position(40, 13), Position(40, 16), "0.000E0", FormatStyle.JAVA, SeparatorStyle.DOT))

```

Figure 3.4: ModeFRONTIER parsing scripts used for the input arrays and the FEA Tcl/Tk script.

Next, the desired constraints and objectives are added and the workflow is debugged by running the system for a single iteration. This workflow is setup to minimize mass while constraining stress to 200 MPa and displacement at the end of the beam to 2 mm. By setting up the node ID number at the end of the beam to 1 in Part 6.4 of the FEA script template, we can easily parse for that node's displacement in the generated Pch output file using a parsing script in modeFRONTIER. With no bugs present, the design automation is set up and ready to run.

3.5 Summary

In this chapter we defined the template-based approach and the procedures used in creating or applying the proposed design templates. This approach was applied to a simplified cantilever beam example using 1D and 2D elements to shown in detail the application and creation of each template and its respective procedures. The following chapters will further apply the template-bases approach to the design cycles of an FSAE vehicle structure to explore the strengths and weaknesses of this approach.

Chapter 4

Template Based Approach for Structural Design Optimization

In this research, we first explore the application of automation to a case study for the design process for an FSAE tube frame structure. This first case study aims to help understand the steps required when setting up automation for the structural design process, while helping to generate base templates for following case studies. The FSAE frames used by the Clemson's Formula SAE team provide a unique application for this research as it has been well established and refined for competition use. The topology of the frame is therefore well established with the competition rules providing constraints for the tube selection and geometry's dimensions. The geometry of the frame is parametrically modeled with only two dimensions being open for modification by the design space. Similarly, 47 tubes on the rear section of the frame are open to optimization for thickness. This provides a good design process for applying automation with the chassis designers able to help guide the bounding of inputs and constraints.

Traditionally, the FSAE's chassis division begins the design process following suspension hard points with two months of design development starting in August. This gives the designer minimal time to iterate on the design with the previous frame having around 30 discrete iterations with less than half being simulated using FEA. This case study instead started in early July to allow ample time in setting up the automation the first time. Figure 4.1 below shows the final vehicle concept applying the results of this studies exploration.

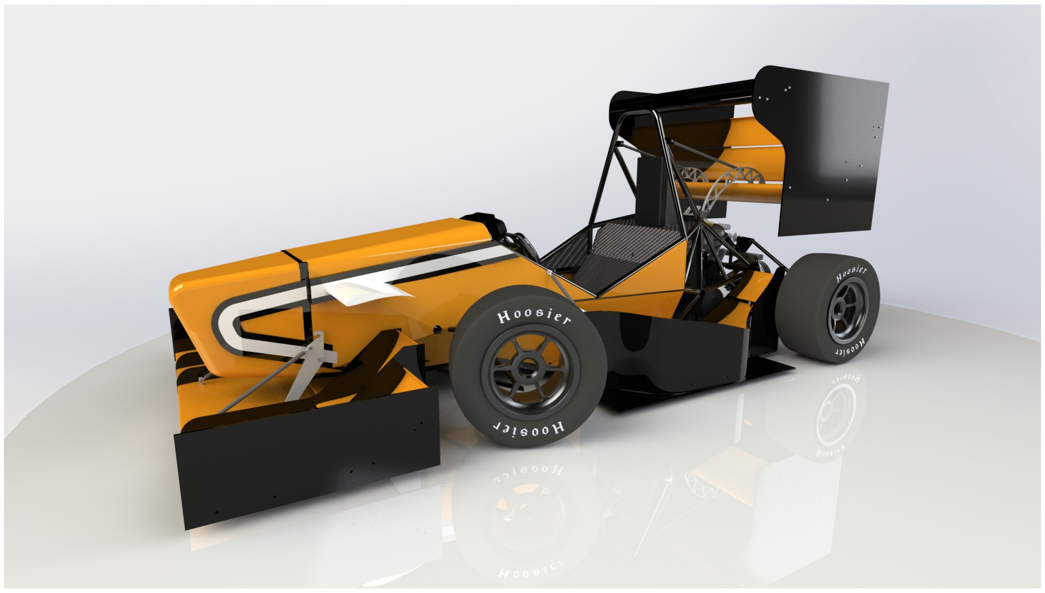


Figure 4.1: CAD Model of finished vehicle concept.

4.1 Parametric Approach to Design Automation

This study approaches automation of the design towards the end of the design process starting in July and ending in November. As the topology is refined, automation tools are established. The automation process was refined throughout this time and implemented in October to generate a set of low mass, feasible design alternatives. This case study explores the application of automation before the development of the template-based approach, and helps to motivate the development of the design templates and their procedures.

To start, the geometry is prepared for FEA by grouping the beam sections into sketches in a SolidWorks component by their cross section. This allows a logical process of meshing the beams in HyperMesh manually and motivates the CAD templates and meshing procedure proposed in this research. The final geometry similarly has an initial sketch at the top of the SolidWorks design tree defining major geometries parametrically while not being meshed in FEA. This additionally motivates the grouping of non-meshed 1D geometries above meshed geometries in the proposed CAD and FEA scripting templates.

With the CAD model parametrically defined, a Tcl/Tk script is manually generated by setting up the FEA once using the HyperMesh GUI and commented for anchoring through modeFRONTIER's batch scripting node. This makes the application of modeFRONTIER's batch scripting node tedious as each input variable requires a discrete input variable node in the modeFRONTIER workflow. To edit the script through modeFRONTIER, an EasyDriver batch scripting node is used to edit all the desired input variables within the script. The recorded Tcl/Tk script requires scrolling through the code to find each instance of an input variable with comments required in some instances to ensure reliable editing by modeFRONTIER's parsing rules. These parsing rules can be saved as a script for re-use within modeFRONTIER, though manually recorded scripts are likely to require unique parsing rule scripts compared to a standardized FEA script.

Results from the FEA are parsed from generated Out and Pch files in the FEA solver batch scripting node in modeFRONTIER. This requires careful anchoring of the parsing rules in modeFRONTIER to allow reliable reading of outputs as the design problem changes. The Out and Pch files output the mass, von Mises stresses, and displacements of the nodes for this study. The modeFRONTIER setup used in this optimization is pictured in figure 4.2 below.

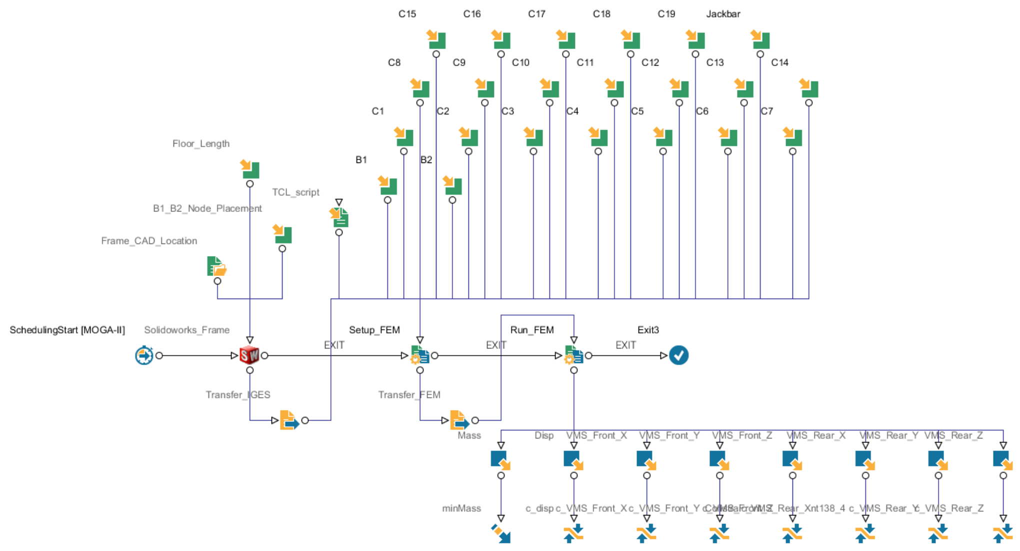


Figure 4.2: Setup of the FSAE structure design workflow in modeFRONTIER.

Design variables are selected for beam sections open to thickness changes and a set of cockpit dimensions shown below in figure 4.3. The design variables are discretized for beam thicknesses from 0.001 inches (0.025 mm) to 0.100 inches (2.540 mm) in 0.010 inch increments. This will require a final rounding of the optimized beam section thicknesses in post processing. The topology of the frame is not symmetric requiring the full vehicle modeled for this analysis, however the beam sections

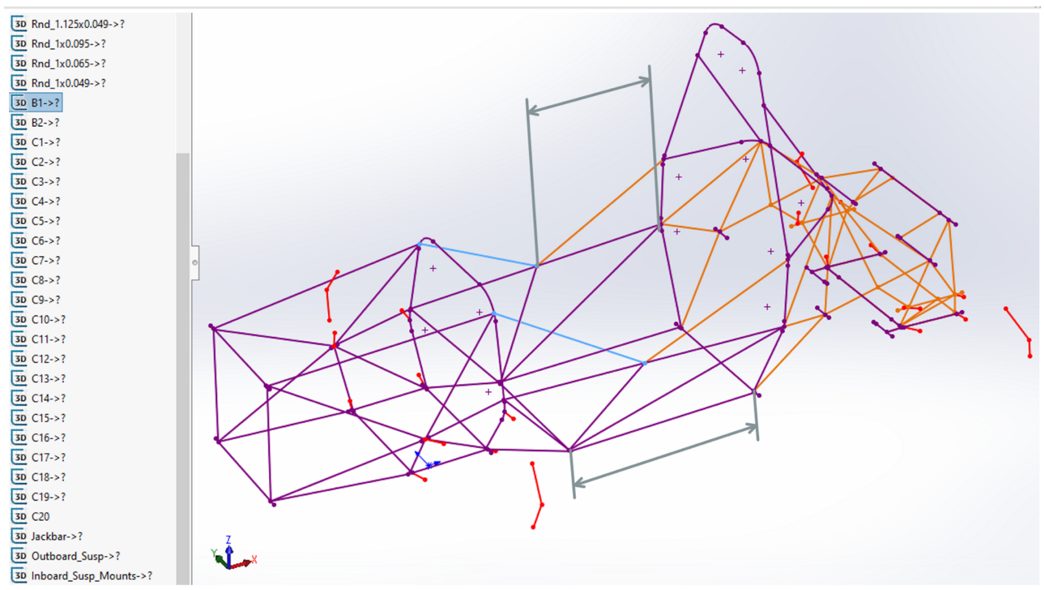


Figure 4.3: CAD model in SolidWorks highlighting the input variable topology dimensions. The orange and two blue beams represent the design space used in the optimization.

are meshed in pairs symmetric to the center plane. Figure 4.3 below highlights this with the ‘B1’ beam sections highlighted in blue in the CAD model.

This optimization aims to minimize mass while constraining torsional rigidity and peak von Mises stress. Torsional rigidity is constrained through outputted Z-axis displacement of the front wheel center. As previously noted, this study implements integrated CAD nodes in modeFRONTIER to optimize the geometry’s dimensions followed by the batch scripting nodes to mesh and solve the FEA simulations. The CAD node updates dimensions defining the topology, outputting updated Iges files for meshing and saving a copy of the CAD geometry with each iteration. This method allows direct updating of the CAD geometry as the optimizer runs, without requiring the designer to manually update the model based on optimizer results. This method also retains important dimensional relations defining the topology, only available through the CAD geometry.

The optimizer selected for this study is modeFRONTIER’s MOGA II genetic algorithm with an automated initial random distribution generated by modeFRONTIER.

4.1.1 Results

An issue was caught during the first six hours of the first optimization, where the absolute value of the displacement was not taken resulting in negative displacements beyond the constraint. This bug is easily accounted for with the final optimization taking approximately two and a half days to complete 10,000 iterations. Results were generated before the final design deadlines used by the FSAE team, with the entire process taking less than five months from July to mid November.

Figure 4.4 below shows the convergence of the mass along with the distribution of feasible and infeasible designs. The minimum mass of the optimization results is 28.65 kg found in many of the final design alternatives. This is likely a result of the small rear tubing presenting negligible changes in mass with minute changes in thickness. The 10,000th iteration is used for further exploration of the designs moving forward in the study.

These resulting tube thicknesses require rounding to the next nearest procurable size, resulting in a mass increase of the 10,000th iteration to 29.0 kg. This is achieved by taking the Tcl/Tk script from the desired iteration and rounding the thickness manually before re-running the script and analysis. Final post-processing through the coping of tube nodes results in an estimated final mass of 26.58 kg in CAD. Out of the 10,000 iterations, 80.65% of the overall design alternatives are feasible. The infeasible designs are evenly distributed throughout the optimization, evident in figure 4.4 below. The majority of the minimum mass design alternatives show a large selection of tube thicknesses at their lowest bounds. This suggests more mass can be lost through expanding the design space or altering the topology.

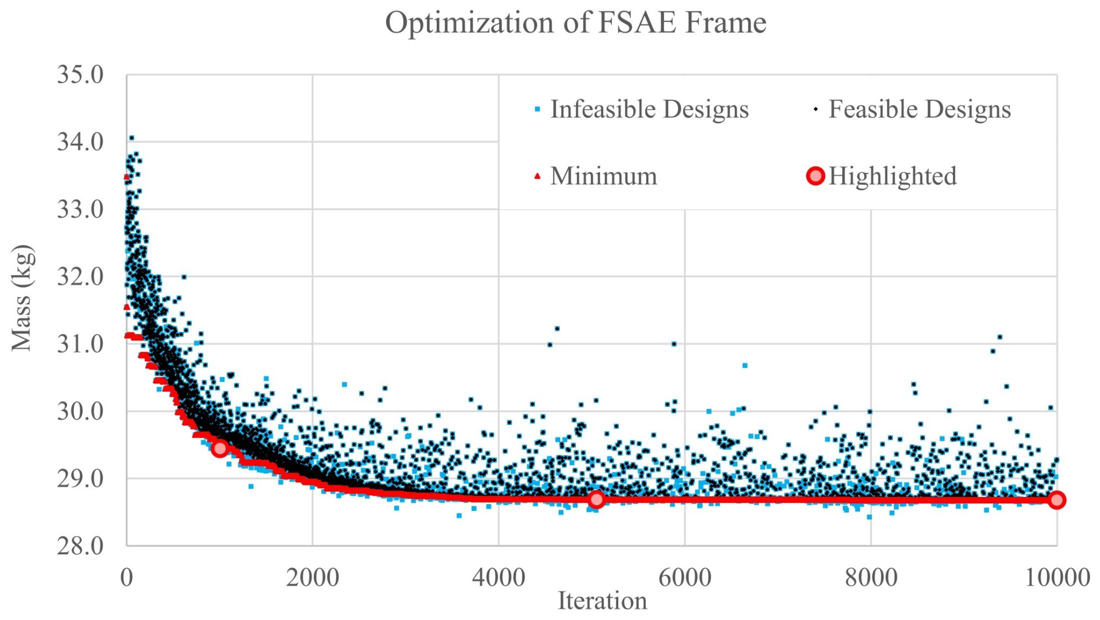


Figure 4.4: In the plot above, infeasible designs are highlighted in blue, feasible are highlighted in black, and the minimum designs by the respective iteration are marked in red. Additionally, three points are highlighted for further discussion below.

Table 4.1: Iteration 10,000 design alternative and nearest rounded values

beam-section	thickness (mm)	thickness (inch)	rouned thickness (mm)	rounded thickness (inch)
B1	**0.940**	0.037	**1.245**	0.049
B2	**1.143**	0.045	**1.245**	0.049
C1	**1.245**	0.049	**1.245**	0.049
C2	**1.600**	0.063	**1.651**	0.065
C3	**1.245**	0.049	**1.245**	0.049
C4	**1.245**	0.049	**1.245**	0.049
C5	**0.889**	0.035	**0.889**	0.035
C6	**0.889**	0.035	**0.889**	0.035
C7	**1.499**	0.059	**1.651**	0.065
C8	**1.803**	0.071	**2.108**	0.083
C9	**0.889**	0.035	**0.889**	0.035
C10	**0.889**	0.035	**0.889**	0.035
C11	**0.889**	0.035	**0.889**	0.035
C12	**0.889**	0.035	**0.889**	0.035
C13	**0.889**	0.035	**0.889**	0.035
C14	**0.889**	0.035	**0.889**	0.035
C15	**1.778**	0.070	**2.108**	0.083
C16	**0.889**	0.035	**0.889**	0.035
C17	**0.889**	0.035	**0.889**	0.035
C18	**0.889**	0.035	**0.889**	0.035
C19	**0.889**	0.035	**0.889**	0.035
Jackbar	**0.889**	0.035	**0.889**	0.035

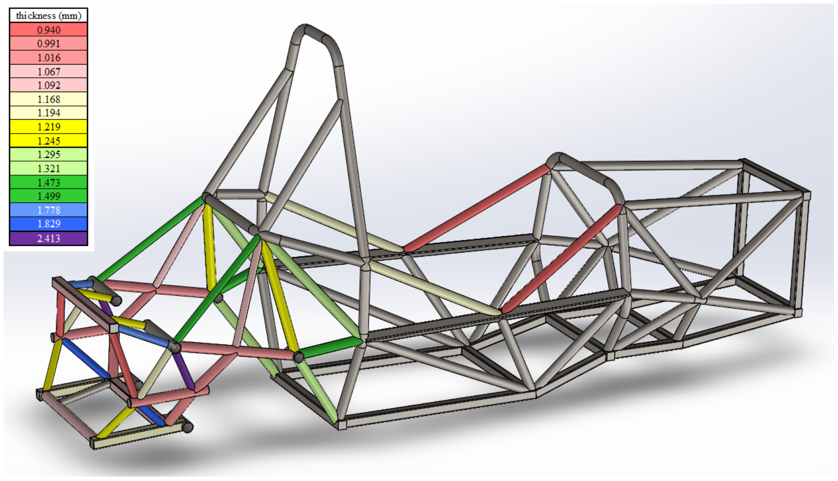
The 10,000th design alternative along with its rounded values is presented in table 4.1. The results show a minimization of most beam sections to the lowest bound of 0.889 mm. This is likely a result of the tight packaging of the rear of the frame, not necessitating larger thicknesses.

To analyze the feasible designs, the design variables from three unique designs are chosen from the optimization's results. figure 4.5 below highlights these designs by selecting the 1,008th, 5,054th, and 10,000th iterations for analysis. These designs represent the minimum mass design of their iteration. Notably, the 5,054th iteration and 10,000th iteration are similar in their overall spread of beam section thicknesses and share a rounded final mass of 28.68kg. This suggests the general convergence observed by iteration number 5,000 might represent a relatively high-performance design alternative. Similarly, only three pairs of beam sections between the 5,008th and 10,000th iterations differ in their thicknesses after rounding to the nearest manufacturable size. This is highlighted below in table 4.1.

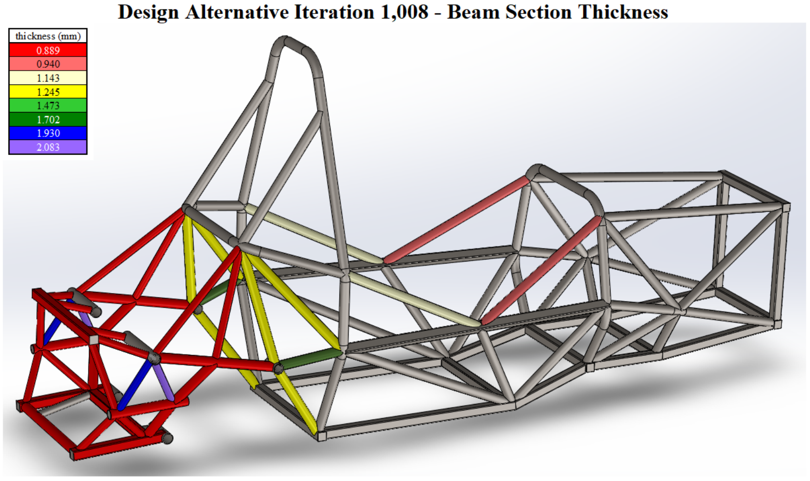
4.1.2 Discussion

This initial case study successfully highlighted areas of improvement when automating the design process, while motivating and directing the development of a template-based approach for following case studies. This study also highlights limitations in the initial approach to automation. The setup process can be daunting for inexperienced designers without familiarity of design the tools. The development of an FEA script and following parsing through modeFRONTIER present the most daunting steps noted by the chassis designers in this case study. This motivates the development of procedures describing the creation of design templates used to automate the design process. In particular, the standardization and parametrization of the Tcl/Tk scripting and modeFRONTIER workflows will be approach to address this concern in following case studies. A standardized FEA script may reduce the difficulty in parsing for inputs while providing parametric scripts that can be reused for multiple design problems.

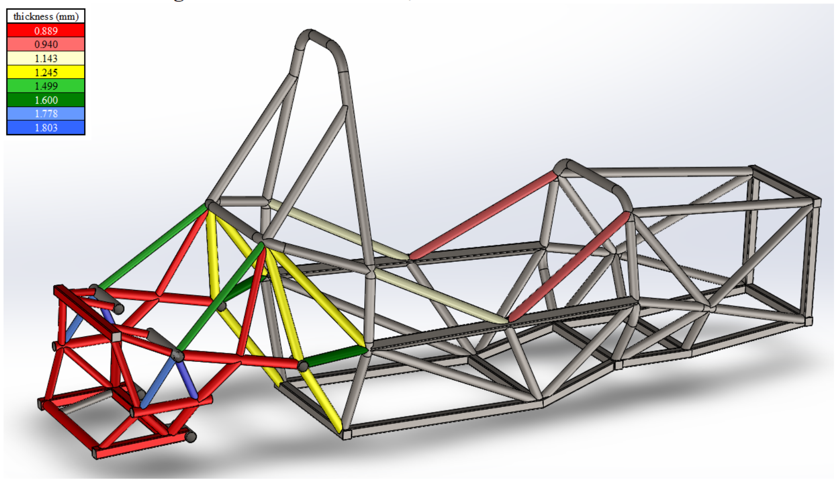
The method of parameterizing the CAD model, implemented in this design problem is robust in this application as there was no unreliability in the CAD geometry alteration. To achieve this, some precautionary steps are taken to increase the reliability. The 3D curves used in each 3D sketch are discretized finely at tubing intersection for reliable meshing. This allows the cleanup scripting to properly join 1D elements to each other's nodes. Similarly, dimensions are defined off the origin and major planes wherever possible to ensure all dimensions are independent of one another. This



Design Alternative Iteration 1,008 - Beam Section Thickness



Design Alternative Iteration 5,054 - Beam Section Thickness



Design Alternative Iteration 10,000 - Beam Section Thickness

Figure 4.5: CAD model in SolidWorks updated to represent the design alternatives for iterations 1,008, 5,054, and 10,000. The grey tubes represent the non design space, while all tubes with color are the design space. These design alternatives correlate to the highlighted designs in figure 4.4 above.

methodology helps to define the procedure for developing the parametric CAD templates proposed in this research.

However, the optimization of the CAD model highlights compatibility issues that may arise specifically when requiring outputted parameters from the model. SolidWorks is chosen for the chassis designer's familiarity and integration node within modeFRONTIER. However, alternative CAD platforms are required if measurements from the model are needed as outputs. Additionally, CAD software without an integrated CAD node will require setting up using the batch scripting nodes, likely increasing the difficulty. Additionally, the CAD platforms tested ran only on Windows Operating Systems. This increases the difficulty of running optimizations including the CAD node on other operating systems.

Overall, the application of optimization in this case study is successful in reducing the mass of the initial design alternative, while informing the dimensions of the geometry. This study highlights the ability to retain important geometric relations by altering the CAD model directly, updating both the CAD geometry and analytical model in each iteration. This opens opportunities to integrate further subsystem optimization into the optimization. For example, control arm geometry for an FSAE vehicle if bounded correctly may be added to the optimizer, allowing further changes to the topology. The following case studies explore the application of a template-based approach while expanding on the design space.

4.2 Template-Based Approach to Design Automation

The initial case study motivated and directed the development of the template-based approach proposed. The following design season for the FSAE team presented an opportunity to then apply the template-based approach to the following 2021 FSAE frame development. This following study explores the application of the design templates and procedures, while expanding the scope of the optimization to include full suspension and frame beam section optimization on top of the previous geometric optimization from the last study.

4.2.1 Application of a Template-Based Approach

To begin automating the design process for the second case study, the parametrically defined CAD model from case study one presents a starting point. The CAD model's topology is updated to the new design season dimensions guided by the chassis designers. The geometry's beam sections are grouped for FEA scripting using the procedures proposed in this research. The suspension links are modeled using the same approach as the space frame with 3D sketches in SolidWorks grouping the two links of each control arm and all remaining links for the spring load path and steering.

Next, the FEA script template is created using the proposed procedures. The script starts with Part 1's user-defined inputs to allow standardization of parsing rules in modeFRONTIER. Input arrays are used for beam section shapes, Outer dimensions (for X and Y local dimensions in the case of non-round shapes), and thickness. This script includes input arrays for material selection and cross section units, however these are not required for the automation of the design process. Inputs for importing the geometry, creating the material card (4130 steel), and geometric variables are created using the proposed procedures in Part 1. Lastly, the boundary conditions of the FEA are scripted as directed by the procedures for Part 6.

Part 2's user-defined functions are then added, using all the functions defined in the example cantilever beam design problem.

Part 3 and 4's script are then created following the proposed guidelines to prepare and mesh 1D beam section geometries. This is identical to the example shown in chapter 3, without the 2D element meshing and property application scripts.

Part 5 then scripts the application of 1D rigid elements. This is the part of the FE script's template requires the most attention as the suspension's joints all require scripting for the desired degrees of freedom. The FSAE vehicles use all six degrees of freedom 1D rigid elements in to model the outboard suspension uprights, engine, and driver mass loading. To model spherical and cylindrical joints in the suspension, 1D rigid elements are used in three and five degrees of freedom respectively. These elements are created using the coordinates specified in Part 1.3's input arrays for geometric variables. These arrays take advantage of symmetry along the centerline plane (XZ-plane) using an option to mirror the coordinates entered across the desired global axis. This is using Part 2's function `find_node_number`. To model these joints, a distance of 0.1 mm is used between the two nodes being connected. This provides room for the elements to more, while adding negligible compliance to the system. However, this does require some outboard 1D rigid suspension links (such as the uprights and wheel assemblies) to use temporary nodes when being created. To accomplish this, Part 1.3's geometric variables are used to create a temporary node at the X,Y,Z coordinates and identified using `find_node_number` to create the 1D rigid element.

Lastly, with 1D rigid elements created, Part 6's boundary conditions are applied. This part requires four sets of "SPC" and "load" type load collectors to be made, following the proposed procedure. The loadsteps are then created following the proposed procedures by manually creating one loadstep, replacing the output options used in the cantilever example, and debugging for batch scripting.

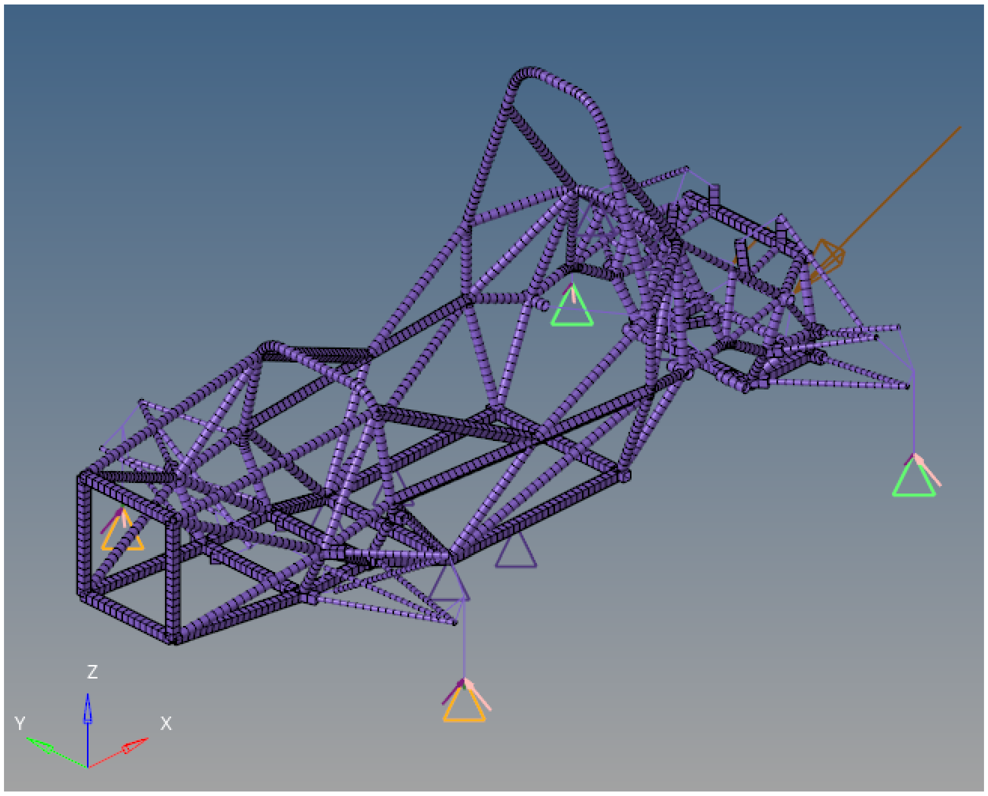


Figure 4.6: HyperMesh model of the 2021 FSAE frame, created by the Tcl/Tk template developed in case study one and refined for the expanded design space.

The load cases scripted for this design's analysis cover each of the major loading conditions of the frame (braking, step steer cornering, and differential loading) and one torsional stiffness test. The bending stiffness load case is removed as the smallest possible beam section selections result in a bending stiffness within the target range. The stiffness load case only outputs displacement used to calculate stiffness, while the remaining load case only outputs peak von Mises stress experienced in the model. The final FEA model can be seen in figure 4.6.

Finally, the modeFRONTIER workflow template is created using input arrays in favor of case study one's individual input variables. This reduces the time required to parse the inputs in the batch scripting node used for FEA model creation. Additionally, these parsing rules are recorded and saved as text files for both the inputs and outputs to allow for quick application of parsing rules for new modeFRONTIER workflows. The parsing rules use comments in the Tcl/Tk script before the input variable arrays to allow automation of the parsing rules. These parsing scripts for inputs and outputs can then be edited to match new Tcl/Tk scripts. This can reduce the time required to set up parsing inputs and outputs for large problems with many of each.

One further addition to the optimization workflow is a rules compliance check using a Matlab script and modeFRONTIER node. This halts the iteration if the inputs are not rules compliant, moving on to the next iteration. This information is taken into consideration by modeFRONTIER's optimizer but does not result in outputs for the iteration. This results in an output that is recorded as neither feasible or infeasible when plotting. This approach allows the optimizer to select designs that are only rules compliant, reducing post-processing steps. Figure 4.7 shows the modeFRONTIER workflow used to run this optimization.

4.2.2 Results

The optimization ran for 13,000 iterations with 228 minimum mass design alternatives generated all within a gram of 69.79 kg. Figure 4.8 plots the results from the optimization, showing general convergence around iteration 10,000. The rules compliance check resulted in 25.4% of iterations stopping, resulting in "NaN" values being reported in the outputs. This resulted in 35.4% of the design alternatives being feasible and 39.2% being infeasible.

The sets of high performance feasible design alternatives allowed discussion between the suspension and chassis designers on the final design of the chassis as a whole. Design alternatives were selected similarly to case study one and discussed for their performance considering further

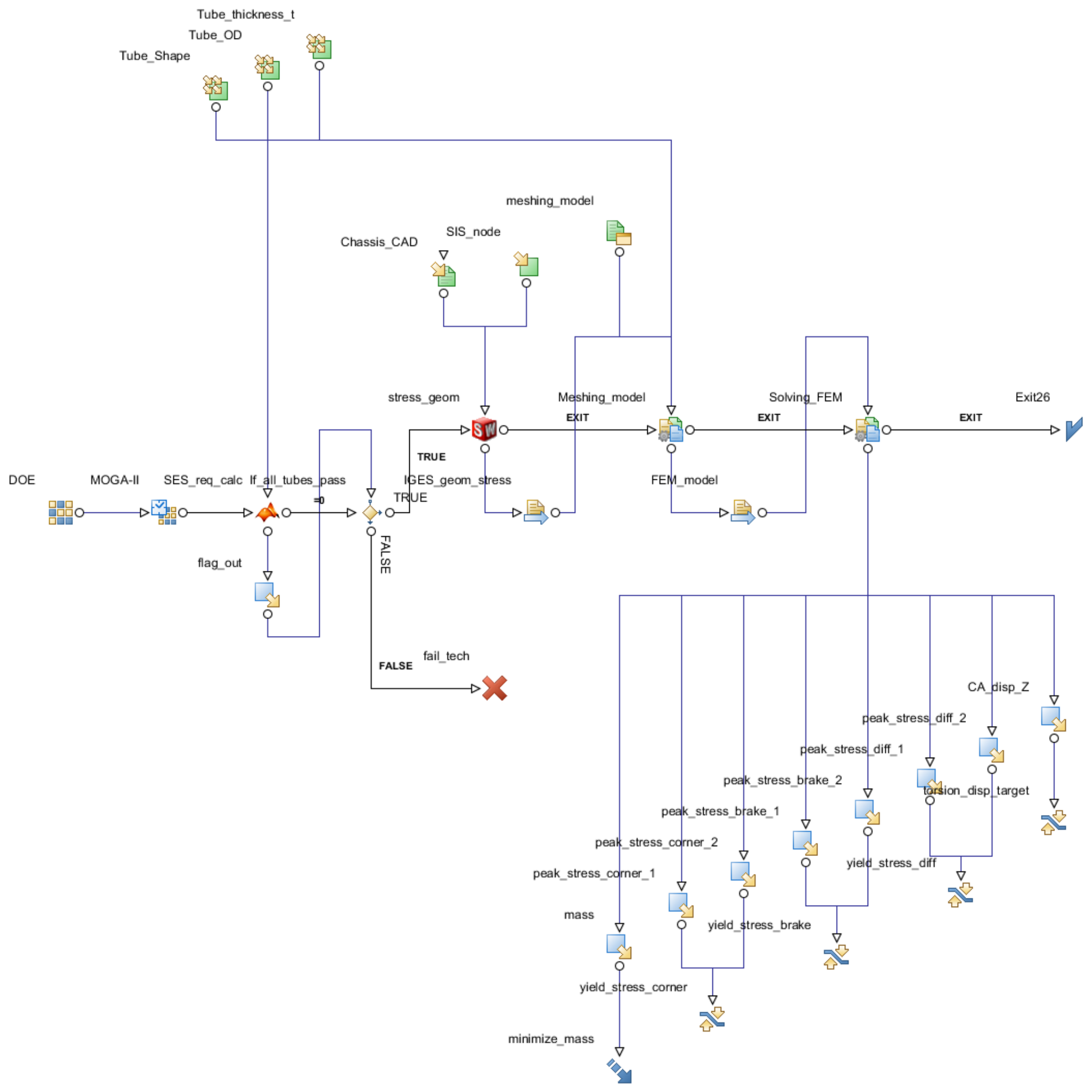


Figure 4.7: ModeFRONTIER workflow for the 2021 FSAE frame and suspension optimization.

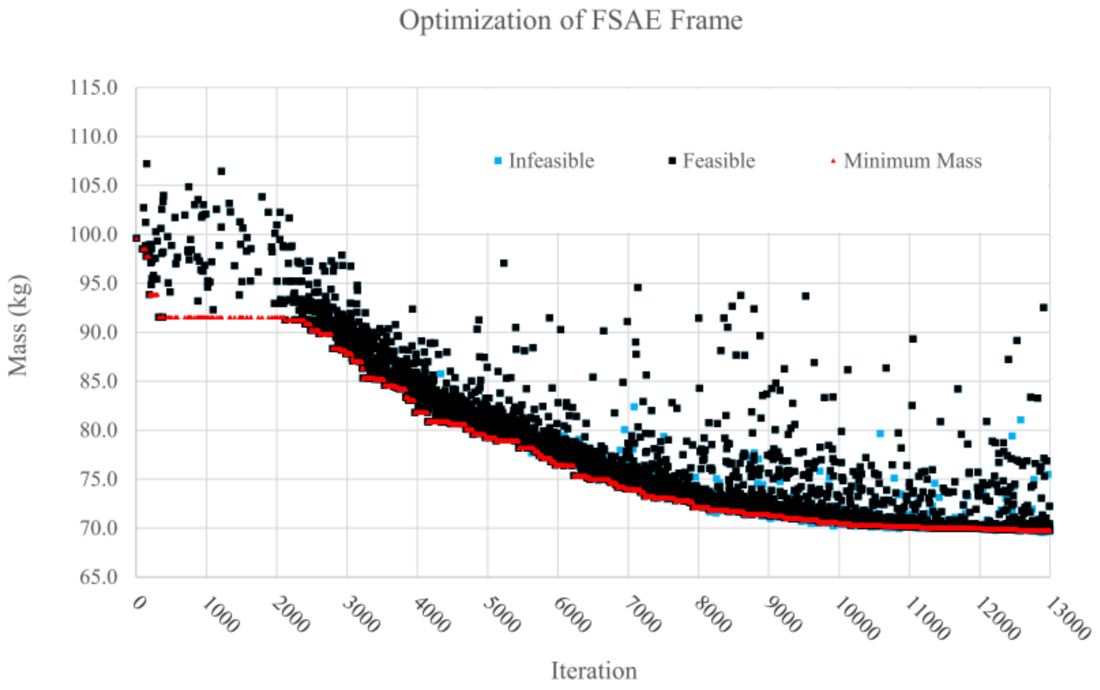


Figure 4.8: In the plot above, infeasible designs are highlighted in blue, feasible are highlighted in black, and the minimum designs by the respective iteration are marked in red.

manufacturability in welding and coping. The development of this frame started in early July similarly to case study one and finished in September. Setting up and running the optimization itself took less than a month.

4.2.3 Discussion

This case study successfully applies the template-based approach to the automation of the design process. The additional time allotted by this approach helped to motivate an expanded exploration of the design space, while generating 228 high performance design alternatives for the designs to choose from in within the standard design cycle. This case study suggests the implementation of a template-based approach may allow designers of following FSAE vehicles to further reduce the time required in automating the design process.

The expansion of the design space to include the suspension links allows the designers of the frame and suspension to develop their respective systems as a total vehicle package, rather than separate entities. This is a further shift from the initial split development of the frame and suspension and leads to increased communication and refinement of models. The adoption and expansion of these tools suggests this approach suits the iterative nature of these design problems and presents further opportunities to automate the design process of similarly iterative design problems. Iterative problems allow minimal alteration of templates for application to new design problems.

Beyond the inclusion of vehicle systems such as suspension, the structural designers identified manufacturability as a following area of interest. The implementation of template-based automation aided the expansion of the design space and will be further implemented to further explore manufacturability in the following section of the paper.

4.3 FSAE Chassis Tolerance Sensitivity Analysis

The following case study three aims to apply the proposed template-based approach to automating the design process for use cases beyond optimization to minimize mass. The motivation for this case study is to approach bottlenecks in the FSAE structure design and manufacturing processes identified in the second case study that may present opportunities for automation. As discussed in the earlier case study, design for manufacturability is identified as an area of interest. The added detail in the addition of suspension link and joint modeling highlights regions of high stress in the front upper control arm (FUCA). This is coupled with identified bottlenecks in the manufacturing process of the tube frame structure when working to reach tight tolerances down to plus or minus 0.002 inches for some manufactured jigs.

This study aims to both expand the application of the proposed template-based approach while helping FSAE chassis designers quantify affects of tolerance selection on the chassis. These bending displacements can lead to higher normal stresses within the links (designed mainly as two force members) as well as impacting the kinematics which can have large impacts on lateral grip from the tires. Figure 4.9 highlights the bending loads experienced by the FUCA for a quarter car model while the rest of the suspension is in pure tension and compression.

4.3.1 FSAE Tube Frame Manufacturing Background

The manufacturing cycle of the FSAE structure for the Clemson FSAE team, generally matches or exceeds the time required for the design cycle, reducing testing time of the entire vehicle package. Previous vehicle structures have taken as long as six months to build depending on the

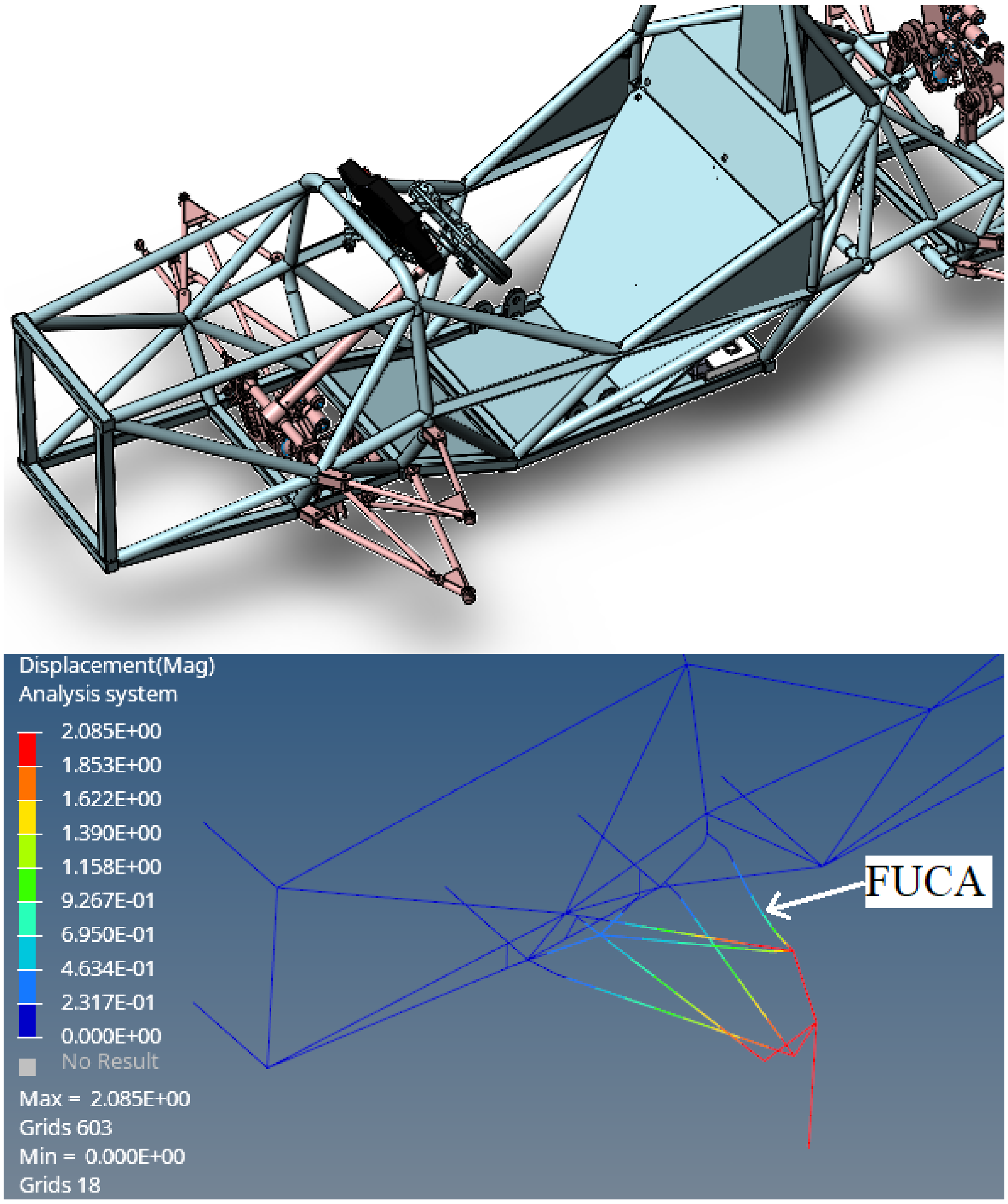


Figure 4.9: CAD of 2020 FSAE vehicle (Top). Displacement results from a single linear static analysis of the Tcl/Tk script for the 2021 FSAE frame and suspension, highlighting the front upper control arm (FUCA) in bending (Bottom).

experience of those manufacturing it. The uncertainty in manufacturing time is identified as a weak point in the development of the FSAE vehicle by the structural designers with a few parameters independent from production experience.

Manufacturability in the form of jigging tolerances in particular, is a controllable metric that contributes some of the largest delays to the build cycle. All tubes in the chassis require careful jigging, quality checking, and re-manufacturing if tubes are not within their specified tolerance, delaying the vehicle's time for testing. Smaller tolerances greatly increase difficulty in jigging the structure. These tolerances are traditionally based on arbitrary baseline tolerances used by the design team, rather than performance requirements. A tolerance study is therefor identified as a good candidate for further application of the template-based approach to quantify the affects of tolerance selection on vehicle performance.

4.3.2 Approach

The approach taken to quantify the affects of tolerance is edited directly from the previous case study two's generated templates. To start, the CAD template is reduced to only the front left corner of the vehicle with dimensions added to desired nodes of the geometry. These added dimensions are at inboard and outboard suspension nodes as well as control arm mounting nodes to allow adjustment in the X, Y, and Z axes. These are changed by the optimizer up to plus or minus 0.100 inches (2.54mm) in the global axes as shown in figure 4.10. This level of error in tolerance is identified by the chassis designer as the largest allowable tolerance errors for the chassis before packaging constraints are broken. Overall this alteration requires three major changes. These are splitting the geometry into the front left corner, deleting the geometries not being used, and redefining the parametric definitions to reflect error in the X,Y,Z axes at the nodes.

Next, the FEA script template can be altered, with four main alterations to the user-defined inputs (Part 1), the 1D rigid application (Part 5) and boundary condition application (Part 6). The user-defined inputs are updated to reflect new geometric variables in Part 1.3. These are updated to reflect new boundary conditions for case study three. Further geometric variables are then added to help account for the nodes being moved by the modeFRONTIER inputs for applying rigid elements and boundary conditions. These are the only parts of Tcl/Tk script parsed by the modeFRONTIER workflow.

Next the 1D rigid scripting in Part 5 are trimmed to reflect the quarter car model. Part 6,

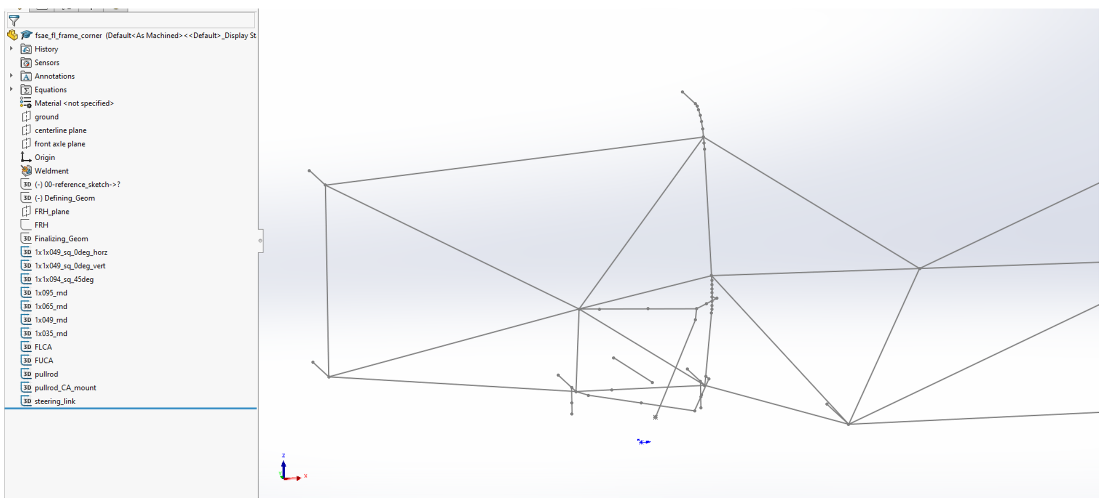


Figure 4.10: SolidWorks CAD model of the front left corner of the chassis using 3D sketches for each beam section.

the boundary condition part of the template is similarly trimmed to reflect a lower number of load cases. The torsional stiffness and powertrain load cases are removed from the Tcl/Tk script. The geometric input variables in Part 1 of the script are then altered to reflect the corner car model's symmetry by applying fixed boundary conditions along the centerline (XZ) plane as well as deleting geometric variables relating to forces applied to the removed three-quarters of the chassis. These load cases are removed as the model is only for the front corner which does not reflect the global stiffness of the chassis or the powertrain loads in the rear of the frame.

Lastly, the modeFRONTIER template is altered, requiring the largest change from case study 2. First, the rule's compliance check is removed as all previous inputs for the beam sections are not being altered in this workflow. Instead, the only alterations to the FEA scripting is the parsing of geometric input variables to account for the changing geometry. Input arrays are added to edit each of the nodes in the X, Y, and Z axes using modeFRONTIER's integrated SolidWorks node. Similarly, output arrays are created for the peak von Mises stress and displacement in bending along three points of each of the control arms for each load case. The displacement is calculated using the X, Y, and Z coordinates of the FUCA's inboard, outboard, and middle nodes for both the forward link and rearward link. These displacements are then sent to a new Matlab node which calculates the absolute magnitude of displacement of the control arm links in bending. This measure for both FUCA links is then set as additional objectives for maximizing. The optimizer is

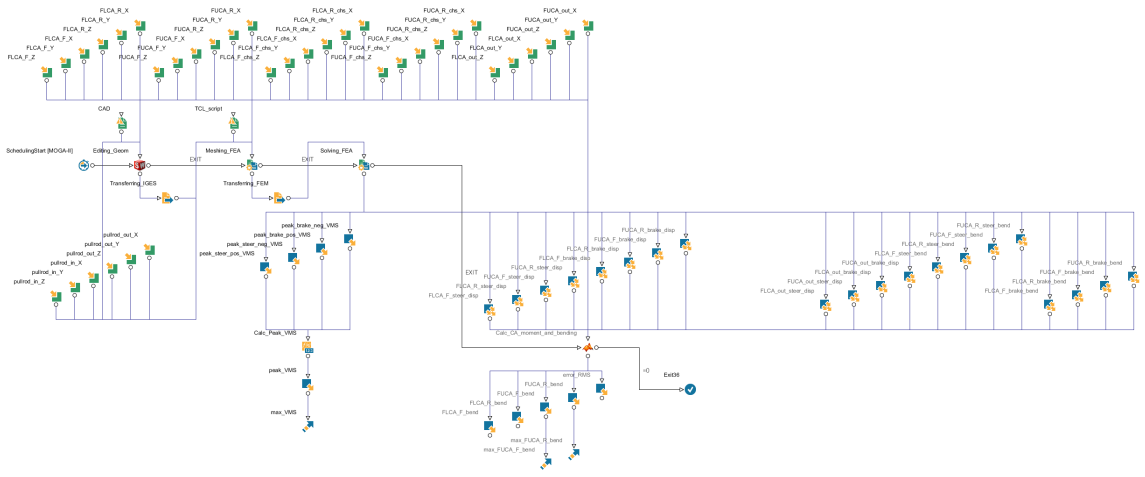


Figure 4.11: ModeFRONTIER workflow for the FSAE quarter car tolerance sensitivity analysis.

modeFRONTIER's MOGA-II with 5,000 self-initialized iterations. This set up requires alterations to every step of the modeFRONTIER workflow template, with the main alterations being the added input arrays, output arrays, and Matlab node for calculating bending displacement. The addition of parsing scripts reduces the time required to set up the workflow, however the creation and connecting of nodes in modeFRONTIER takes a relatively large amount of time and effort compared to the rest of the alterations. The final modeFRONTIER workflow is shown in figure 4.11.

The optimizer can therefore explore the design space for extremes in the solution space by modifying the nodes to maximize the peak stress and bending displacement in the FUCA. The FUCA is only selected as it has the pullrod mounted from it and experiences bending loads from the spring load path, where the remaining suspension links are purely in tension and compression.

4.3.3 Results

The edits to case study two's templates required roughly one months of attention to get running. The largest amount of time was invested into setting up the new input and output arrays, the connecting of the nodes in the workflow, and debugging the workflow for reliable measuring of bending displacement. The Matlab code required debugging for varying X, Y, and Z coordinates to ensure no dependence on global coordinates.

The optimization to maximize stress and displacement resulted in a maximum stress for any iteration recorded of 311.2 MPa, 123.0 MPa larger than the minimum peak stress for any iteration of

188.2 MPa. Figure 4.12 plots the peak stress recorded in the FEA model over the design alternative iterations. Here, we can see the optimizer begins converging to a design alternative that maximizes the stress delta around iteration 2,000.

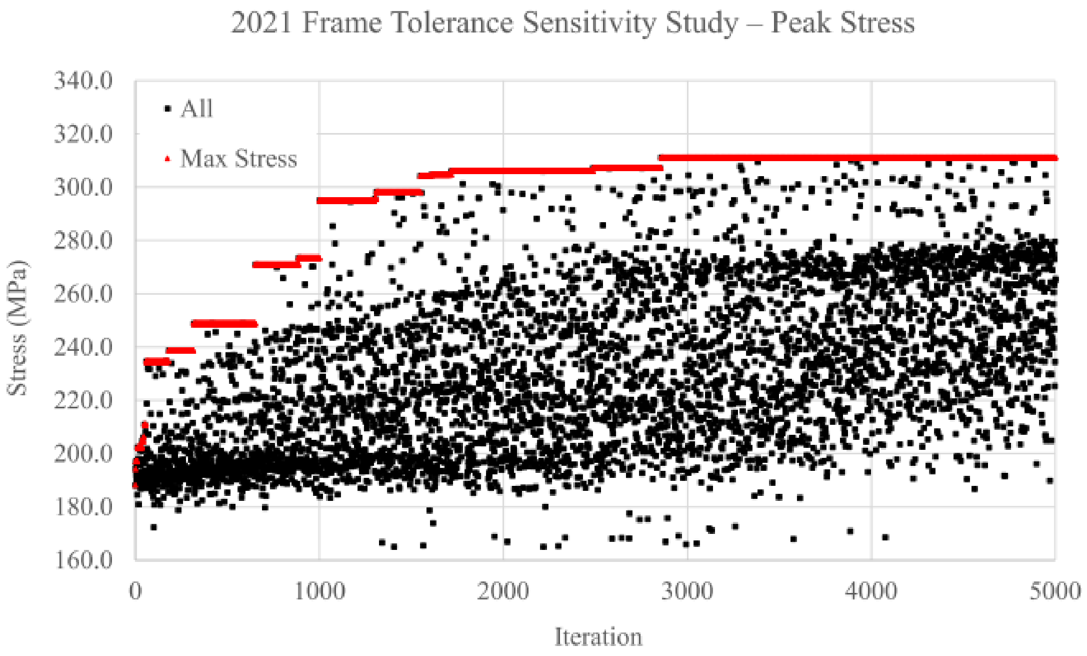


Figure 4.12: Plotting peak stress of each design alternative over the iteration number.

A maximum bending displacement is recorded at 30.2 mm, where the minimum recorded is 1.6 mm. These results show significant stresses and bending behavior under worst case error in tolerances. Figure 4.13 plots the peak bending displacements of the FUCA's forward and rearward links for each design alternative. Similarly to the stress, we can see convergence starting around iteration 1,000 where the optimizer has found a design alternative that induces nearly the maximum possible bending into the FUCA.

Further analysis of the highest stress and displacement design alternatives show the largest stresses in the FUCA links and spring load path mounting. This suggests the frame itself is relatively robust to tolerance errors, experiencing far less stress than the suspension links and mounting after manually reviewing some of the highest stress design alternatives. This can be improved in later studies by implementing checks for which components the peak stress is recorded. However, these results are expected as the frame is made from much larger cross-sectional beam sections with many beams bracing each node.

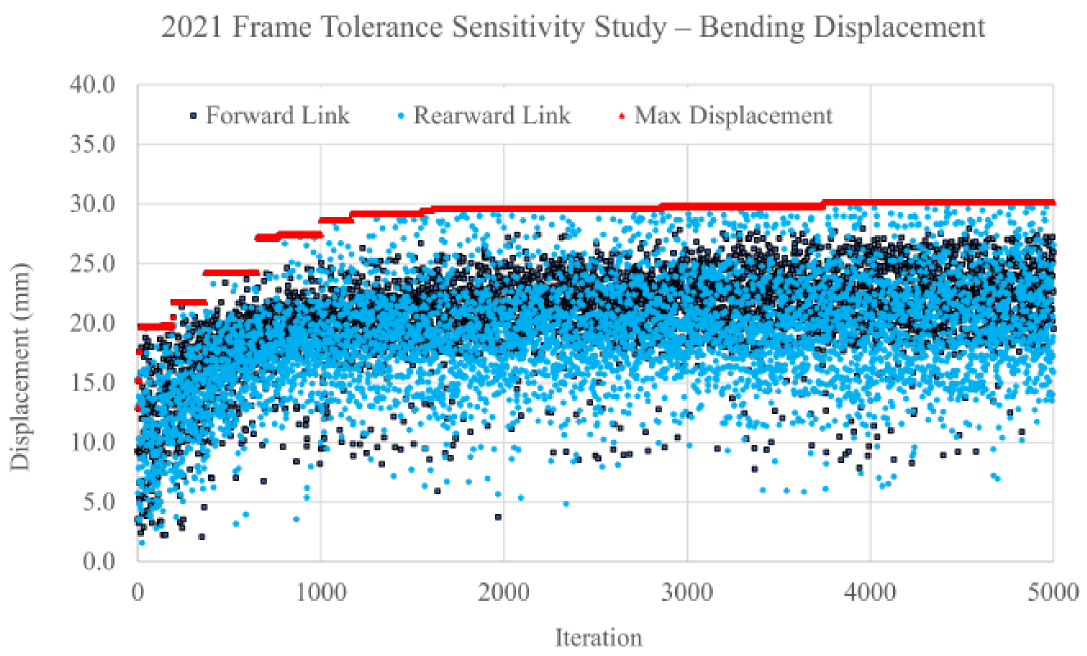


Figure 4.13: Plotting peak bending displacement in the forward and rearward FUCA links over the iteration number.

4.3.4 Discussion

This study highlights the ability to adapt existing templates for new design problems. The reuse of the frame's CAD and FEA script templates require minimal alterations, while the expanded modeFRONTIER workflow requires a relatively high amount of effort. The expansion of the modeFRONTIER workflow however would require similar, if not more effort to create if not reusing the base template from case study two. This approach allows the chassis and suspension designers to set informed tolerances for both the suspension and frame within the usual design schedule.

4.4 Summary and Conclusions

The use of a template-based approach shows promise for reducing the time to apply automation to the design process. The presented case studies show the benefits of a refined set of templates driven by a set of procedures. The results from the second and third case studies suggest this proposed approach can be reduced further to expanded design spaces and alternative design problems. Bottlenecks are identified in particular at the scripting of 1D rigid elements, the expansion of the modeFRONTIER workflow, and the implementation of further steps to the design process workflow.

Lastly, these advancements help to improve the FSAE chassis designers' ability to explore the design space for feasible high-performance results. However, with optimization being established as a standard design tool for the development of FSAE structures, opportunities for mass reduction require departures from the conventional space frame structure. This helps to motivate the following design exploration of composite structures for stiff, lightweight designs automating the process through the proposed template-based approach. The following case studies in chapter five apply the proposed template-based approach to the exploration of initial composite monocoque structures, expanding the design space from 1D tube models to 2D surface models.

Chapter 5

Optimization of Composite Geometry and Laminate Stacking

The following case studies further expand the application of a template-based approach to the development of carbon fiber reinforced plastic (CFRP) structures with a honey comb core sandwich construction. The template-based approach is chosen for application to the ideation design stage of an FSAE composite monocoque structure as it shows promise for reducing the time and effort in setting up the design space. The the aim of this case study four is then to identify weaknesses in a template-based approach when applying previously developed templates to a relatively unique design problem. The exploration is applied to the ideation stages of the design process, to help the chassis designers explore the design space for feasible, high-performance structures. This exploration serves as the largest departure from the previous structural design problems, with changes mostly being in the automation of the CAD template and FEA script template.

5.0.1 Background

CFRPs have allowed designers to design stiff, light-weight structures when compared to traditional materials, but at the cost of increased complexity in the design process [26]. A challenge the FSAE designers face is the lack of first hand familiarity with similar CFRP structures. To gain an understanding of the design space, the design process for stiff, lightweight structures is automated using the proposed template-based approach. Specifically, the design process for CFRP sandwich structures is investigated for a baseline composite monocoque geometry.

The previous templates for steel tube structures are modified following the proposed procedures for composite sandwich structures, optimizing the CAD geometry and discretization of plies across the structure before exporting the model for FEA. The CAD geometry is set up using similar parametrization to previous approaches with alterations to allow reliable modeling of 2D shell geometries. The workflow and Tcl/Tk script templates are edited to optimize the laminate and its stack up for 2D shell elements, allowing the optimizer to apply each laminate's number of plies, layup order, core sizing, material selection, and orientation by modifying the corresponding Tcl/Tk script. Like previous workflows, all changes to the geometry and laminate are made in one optimization iteration.

5.1 Template-based Approach for CFRP Laminate Optimization

The design process for vehicle structures can be cumbersome, as previously discussed with CFRP sandwich structures adding further complexity to the process. This limits the designer's ability to explore the design space manually and presents further challenges when automating the process. Each change the designer makes to the geometry or laminate design in the pursuit of their objective, requires a new model and analysis to ensure targets are met.

To tackle the challenge of applying automation to CFRP sandwich structures, the template-based approach is edited for 2D element laminate modeling. Templates developed in case studies two and three are altered for the new design exploration to investigate the application of existing templates to new design problems. In this exploration, the design space of a CFRP sandwich monocoque structure is explored to minimize mass while constraining stiffness and modal response. The geometry is modeled for common 2x2 twill weave, prepregnated carbon fiber with an aramid honeycomb core material. Stress is removed from this exploration as accurately modeling stress within composites presents unique challenges beyond the scope of this ideation study. This is chosen as adequately distributing loads into a CFRP sandwich structure generally requires insert components and core materials to be used, otherwise stress concentrations are likely to develop. This requires detailed modeling at the location of boundary conditions beyond the scope of the structural ideation stage. The overall stiffness and modal responses instead can be measured using simplified models for early exploration of the design space. Modal analysis in particular is identified as an area of concern as composite panels can experience resonance from ride and powertrain frequencies that might upset the performance of the vehicle or comfort of the driver.

The proposed changes to the template-based approach applies orientation, core thickness, application of plies over geometry, and ply shuffle optimization in one step through Tcl/Tk scripting. The procedure for generating and altering FEA script templates is therefore edited to accommodate composite modeling in HyperMesh:

1 - Part 4 is expanded to include the creation of plies following the creation of the mesh and application of element properties.

2 - Part 4 is expanded to include the modeling of an overall laminate following the creation of the plies.

Individual plies can be removed from the layup entirely, while shuffle optimization is performed through an entire optimization of the layup parameters directly. By providing an adequately large initial selection of plies, the optimizer can remove unnecessary plies, effectively shuffling the order of the stack up. Furthermore, parameterizing the geometry plies are applied to allows a greater exploration of the laminates' design space.

The design problem is formulated in the following way:

Given:

- Surface assembly model, divided into `nc` components
- Loading conditions
- Material properties of fiber and resin

Find design variables:

- Up to `maxP` plies divided equally on both sides of the core
- Core thickness chosen from discrete set of thickness values
- Vector of `maxP` Boolean values indicating whether a particular ply is present
- Vector of `maxP` ply directions chosen from discrete set of orientations
- `maxP` by `nc` Boolean array indicating whether each of the `maxP` plies is present in each of the `nc` component surfaces

That minimize total mass

Subject to constraints:

- Maximum displacement constraint
- Minimum first bending mode frequency

Two design explorations are used to explore this approach through the study of an example Formula SAE monocoque with the objective to minimize mass while reaching a target stiffness. This model is set up for a torsional stiffness analysis similar to previous FSAE steel tube space frames, constraining displacement relating to a target torsional stiffness at or under the target. A modal analysis is added to constrain the first bending mode frequency to meet or exceed the target.

5.2 Laminate Stacking Optimization Approach

This approach optimizes the model for laminate stacking by editing the number of plies, ply orientations, ply thicknesses, geometries plies are applied to, and core thickness, in one step through automation of the design process. Automation of the FEA is performed in modeFRONTIER based on the previously generated and refined design templates and procedures for the parametrization of CAD, FEA scripting, and modeFRONTIER workflow.

5.2.1 Altering the Parametric CAD Template

To begin, a CAD model is parameterized and prepared for FEA by discretizing the geometry and grouping surfaces for plies to be applied to. These geometries are modeled in CAD as separate components within an assembly to allow top-down editing of the component dimensions, while grouping surface geometries for importing into FEA. Like the earlier space frame CAD templates, this setup discretizes the surfaces where intersections and boundary conditions were located to improve meshing and ease load case scripting. The change to a top-down assembly represents the only alteration to the CAD template.

5.2.2 Altering the FEA Script Template

The largest departure from previous workflows is in the alteration of the FEA script template. As previously stated, additions are made to the procedure for Parts 1, 2, and 4 of the FEA script generation/alteration. To optimize the entire laminate, the number of plies, geometries the plies are applied to, orientation of plies, and thickness of the ply layers are all variable arrays defined in Part 1. This makes each input variable array available to editing by the optimizer. This effectively allows the optimizer to change the entire stack up of the laminates over the geometry.

To implement this, ply creation input arrays are added to Part 1's user input arrays to allow parametric creation of plies, like the approach used for 1D beam sections. Arrays are created for selecting the orientation, thickness, material, and geometry grouping the plies are applied to. This is done through 'sets' of elements in HyperMesh, grouped by component ID, highlighting the importance of grouping surfaces that are imported as separate components. To ensure a wide exploration of the design space, the arrays start with 51 inputs, with 25 carbon fiber plies on each side of the core layer. This is informed by a practical limitation of number of plies that can be feasibly manufactured by the composite engineers. The core, is applied to the entire structure as a design constraint driven by manufacturability and rules compliance. The core is modeled in this approach as another ply for simplification and is only optimized for thickness in this study, as it is representing a honeycomb material with minimal impact on performance along the shear directions as orientation is changed.

The input array for the orientation of the CFRP plies is from zero to 90 degrees in 15 degree increments, while the core is not optimized for orientation. Below is an example script for the array (the "..." is a place-holder for input array indexes, imputed by modeFRONTIER through the batch scripting node):

```
# Setting ply-section orientations
set ply_angles {0 45 ...  } # input array is inputted by modeFRONTIER
```

The thickness input array allows the thickness of the carbon fiber plies to be chosen. As this design exploration is only using one size of CFRP ply, the thickness for their indexes in the input arrays are either set to zero or their manufacturing thickness of 0.22 mm. This means the optimizer can remove plies from the model, but cannot add plies. This represents a limitation to the approach, requiring a large initial design space for the optimizer to explore. The core layer indexed in the middle of the array is available to change from 5 to 20 mm with 3 mm increments (based on off-the-shelf options chosen by the chassis engineers). Below is an example script for this input array:

```
# Setting ply-section thicknesses
set ply_ts {0.22 0.22 ...  }"
```

An array for material application is not included in the scope of the optimization, but is scripted to model the 51 plies in the input arrays using the appropriate properties for CFRP or aramid core. Below is an example script for this input array:

```
# Setting ply-section materials (1 = CFRP, 0 = aramid honeycomb core)
set ply_mats {1 0 ...  }"
```

For the input array applying plies to sections of the geometry, each index in the array relates to a ply layer in the laminate, where an input of zero will apply the ply only to lower section of the structure and an input of one applies it to the entire structure. Below is an example script for this input array:

```
# Setting ply elements for desired sets (1 = comps 1 and 2, 0 = comp 1)
set ply_sets {0 1 ...  }"
```

An additional input variable `layup_type` is created to determine if the layup is "symmetric" or "total" (symmetrically applies plies across geom vs. applying plies directly without mirroring). A selection of 0 relates to a "total" layup, where a selection of 1 relates to a "symmetric" layup model.

Next, a user-defined function "`create_mat`" is added to Part 2 for creating plies parametrically through the use of input arrays in Part 1 (this can be seen in Appendix B). This method is similar to the approach used by the 1D beam section creation from earlier case studies.

Lastly, two steps are added to Part 4 to create the plies and model the laminate respectively. A for-loop is used to sequentially create each ply and record its ID # for laminate modeling:

```
# Initializing ply ID array
set ply_IDs {};
# Creating plies, recording ply IDs in array for laminate modeling
for {set j 0} {$j < [llength $ply_ts]} {incr j} {
      create_ply "ply $j" [lindex $ply_ts $j]} ...
            ...  [lindex $ply_angles $j]} [lindex $ply_sets $j]} ...
            ...  [lindex $ply_mats $j]}
        # Appending ply ID array
        lappend ply_IDs [expr {$j + 1}]
}
```

The laminate is then modeled with the following example script:

```
# Layup order:  1 ply1 ply2 ply3 etc.  - "1" is required for total layup
# Creating input array for the plys IDs recorded in the creation step
eval *createlist plies 1 $ply_IDs
*laminatecreate "laminate name" [expr {$layup_type*5}] 0 0 0 1 1 0 0 0
*createmark laminates 1 "laminate name"
*dictionaryload laminates 1 "C:/.../optistruct/optistruct" "STACK"
*initializeattributes laminates "laminate name"
*createmark laminates 1 "laminate name"
*clearmark laminates 1 # note:"laminate name" can be scripted as a var.
```

The laminate is created following the creation of all plies, sandwiching them together and modeling their interactions over the geometry. This captures the modeling of separate laminates over the geometry, where unique ply stack ups are located. Figure 5.1 highlights this modeling where parts 1, 2, and 3 represent three geometries with overlapping plies A and B. Ply A is applied to geometries part 1 and part 2, while ply B is applied to geometries part 2 and part 3. The laminate process in HyperMesh then captures the three unique laminates created where the plies interact.

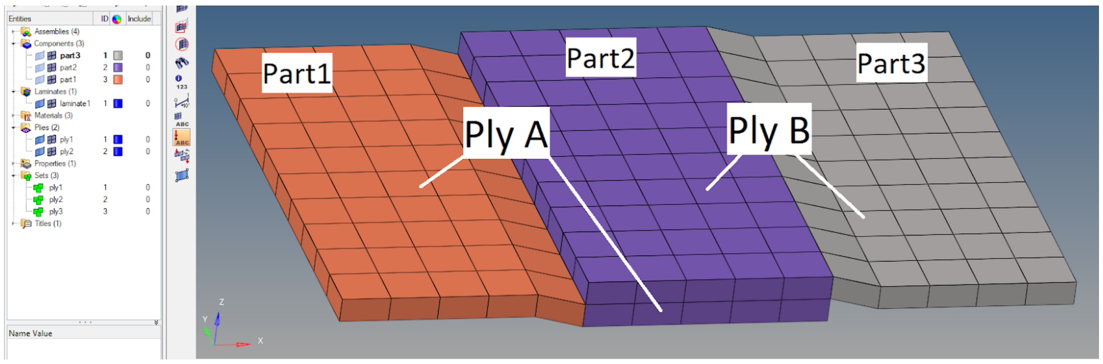


Figure 5.1: Example of a composite plate modeled with two overlapping plies across the center of the plate. The geometry is imported as an assembly with each part file representing a different laminate (only ply A, ply A + B, and only ply B). Part files are imported as surfaces for meshing as 2D elements.

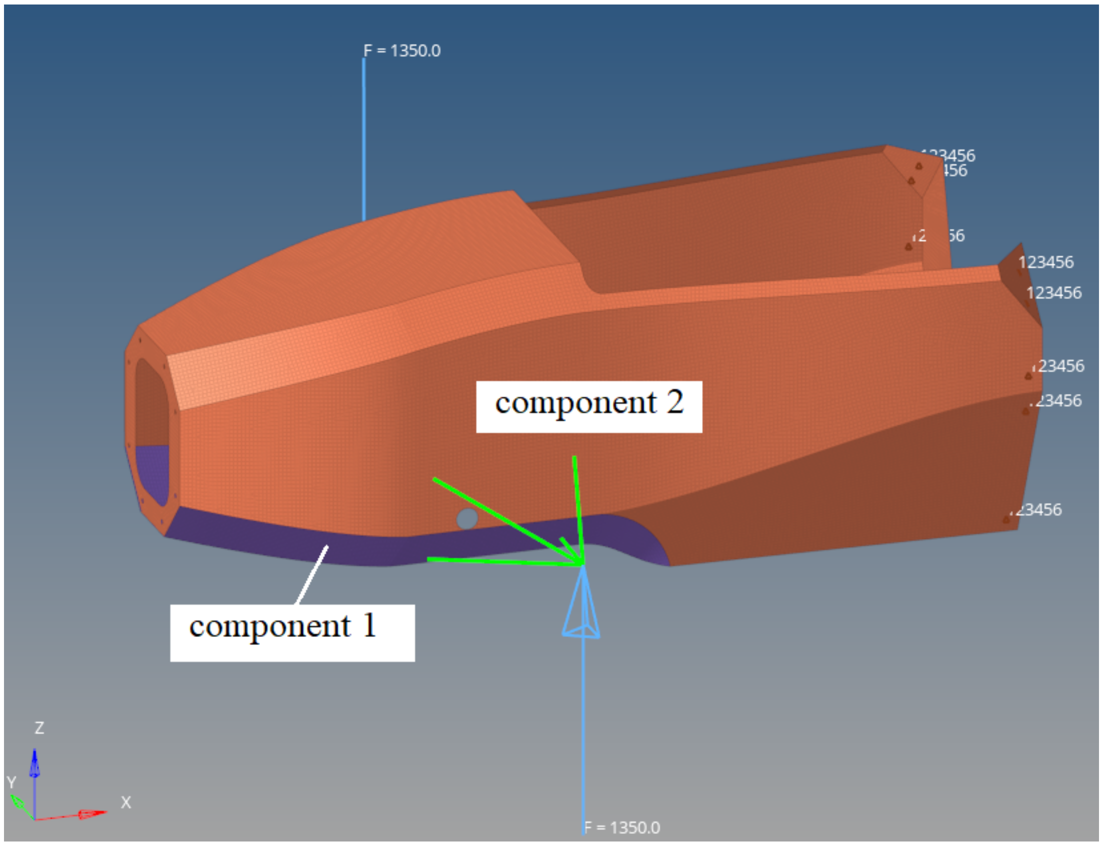


Figure 5.2: Composite monocoque FEA model in HyperMesh highlighting the two sections of the structure with differing laminates.

Figure 5.2 highlights how this is applied to the composite monocoque model where plies can be distributed over the main body of the structure component 2 as well as the the lower suspension bracing in component 1, or only the lower section component 1. This initial exploration is limited to two geometries initially and expanded in a following design exploration.

5.2.3 Altering the ModeFRONTIER Workflow Template

The modeFRONTIER workflow template from case study two requires minimal alteration. Input and outputs arrays are updated appropriately. The CAD node is deleted and the updated FEA script template is re-parsed for the ply input arrays, using a revised parsing script from the tube-frame structure setup. The outputs are similarly re-parsed by updating the script to find the bending frequency from the Out file. These three changes required relatively minimal effort to set up when compared to case study three, with the final workflow shown in figure 5.3.

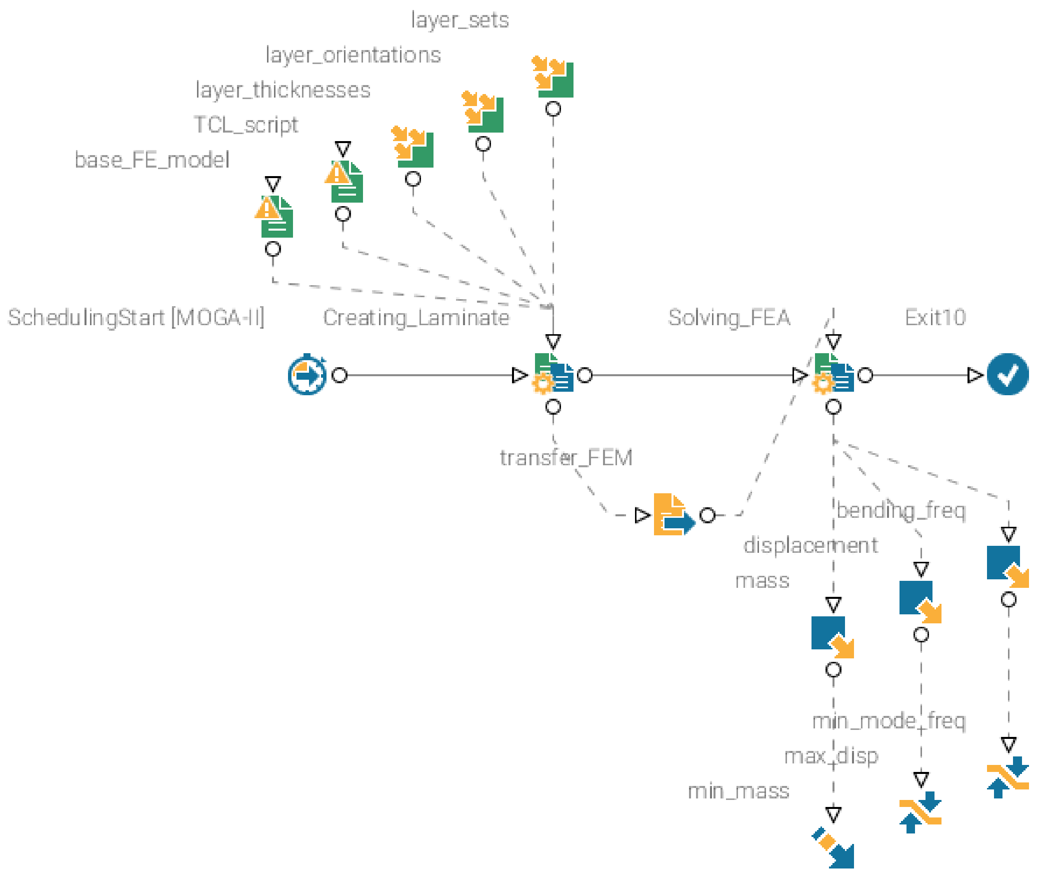


Figure 5.3: ModeFRONTIER workflow setup showing the flow with the FEA meshing and solving through their respective EasyDriver nodes.

The case study is a single objective optimization to minimize mass, while constraining torsional stiffness and modal bending response. The torsion load case is modeled as 1,350 N in opposing directions at wheel center nodes with simplified 1D rigid elements modeling the suspension. Torsional stiffness targets are measured like the first FSAE frame explored in this research, using displacement at the wheel center node in the Z axis and constrained to 6.6 mm maximum. Modal analysis showed the first bending mode to be the second modal response and is constrained to a minimum of 35 Hz. Outputs from the optimization are parsed from generated Out and Pch files by the OptiStruct solver like in previous workflows. Displacement at the wheel center is measured from an outputted Pch file, while the bending frequency is outputted from a Out file. The optimization workflow is a self-initialized MOGA-II genetic algorithm from modeFRONTIER. To reduce time required for each iteration of the optimization, the Tcl/Tk script was shortened to import a pre-meshed FE model with only the plies and laminates not created or applied. This simply replaced the geometry importing Part 3 and initial automated meshing from Part 4. This change is chosen to reduce computation time, following the setup of the entire workflow including the re-meshing of the geometry. This otherwise has no impact on the final results generated by the optimization. The following case study five requires re-meshing and will explore re-introducing that step for complex 2D structures.

5.2.4 Results

The optimization was limited in time to meet deadlines for the 2021 SAE World Congress submissions, limiting the iterations that could be ran. The optimization resulted in 8 feasible designs with a minimum mass of 19.48 kg after 500 iterations. A plot of the results is shown in figure 5.4. This optimization was not set to use convergence criteria. A longer optimization is also likely to generate an even larger set of minimum mass feasible designs. However, this ideation phase of the monocoque development did not require rerunning the optimization for longer, so the following case study instead increased the iterations run by the optimizer.

The first design alternative with a mass of 19.48 kg is found at iteration 310. This design alternative has 6.49 mm of displacement in the Z direction with a bending mode of 78.6 Hz shown in figure 5.5.

Table 5.1 shows the input variables for iteration 310 with a core thickness of 14 mm. The right most columns represent if the plies are applied to the components 1 or 2. The plies are

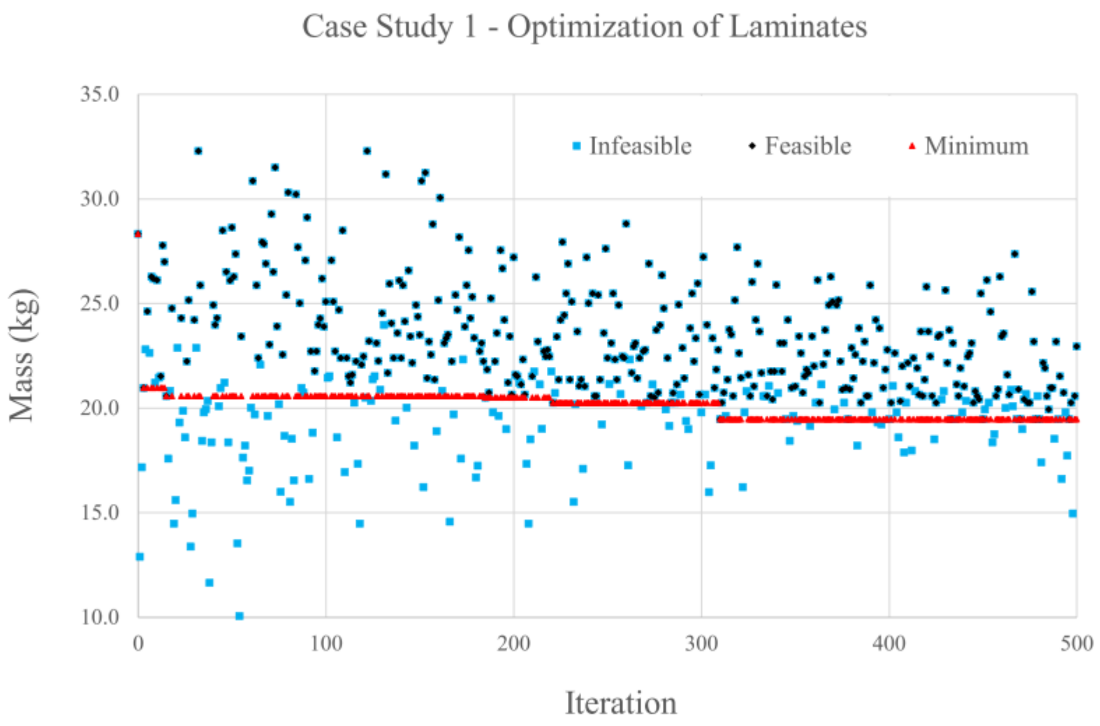


Figure 5.4: In the plot above, infeasible designs are highlighted in blue, feasible are highlighted in black, and the minimum designs by the respective iteration are marked in red. Additionally, three points are highlighted for further discussion below.

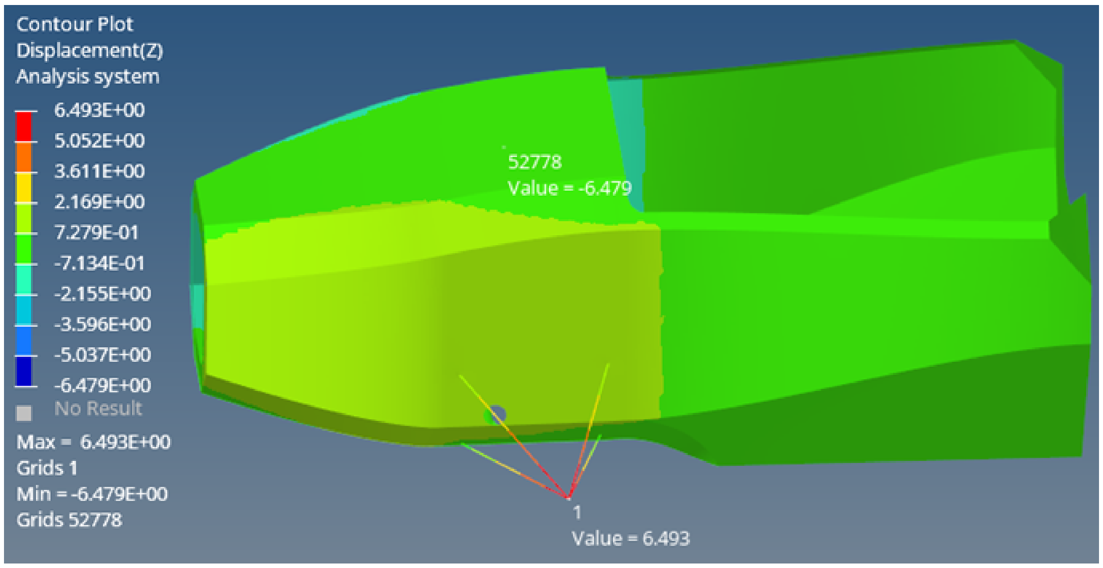
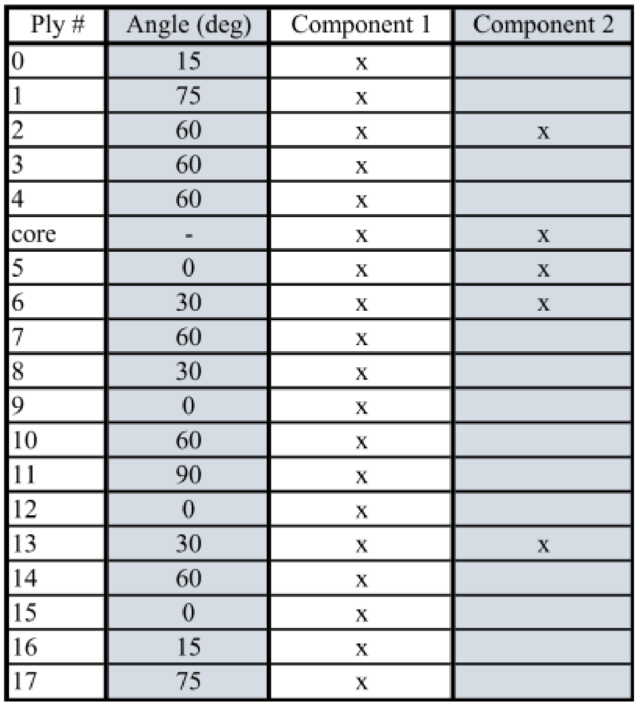


Figure 5.5: Displacement in the Z direction results from iteration 310.

numbered, starting from zero, with plies from zero to the core layer being applied to the outside of the structure, core layer to maximum ply number being on the inside.

Table 5.1: Input variables for design alternative 310.



Ply #	Angle (deg)	Component 1	Component 2
0	15	x	
1	75	x	
2	60	x	x
3	60	x	
4	60	x	
core	-	x	x
5	0	x	x
6	30	x	x
7	60	x	
8	30	x	
9	0	x	
10	60	x	
11	90	x	
12	0	x	
13	30	x	x
14	60	x	
15	0	x	
16	15	x	
17	75	x	

With the optimization completed, generated designs are analysed by opening the FEA output files directly generated in the optimization and by viewing the edited Tcl/Tk scripts for each iteration. The lack of plies applied to only the second component suggests discretization of the geometry in other areas might result in larger mass reductions. This result is less pronounced than first expected by the designers, as the lower suspension mounting generally plays a large role in stiffening the structure for FSAE vehicles.

5.2.5 Discussion

This initial case study four, was successful in applying the previous template-based approach to a new design problem for rapid generation of feasible, high-performance design alternatives. This suggests the proposed approach can be applied to a range of structural design problems. To alter the template-based approach for composite structures, one major change to the CAD template, the following major changes were required.

The CAD geometry was changed to an assembly in SolidWorks to allow grouping of surfaces as SolidWorks components.

Next, the FEA script required three major changes. Part 1 was updated to allow input arrays to be used in Part 2's new ply creation function, like the approach used for 1D beam section meshing. The ply and laminate creation steps were then added to Part 4 of the FEA script.

Finally, the modeFRONTIER workflow was updated to reflect the new input arrays with parsing scripts updated appropriately. The changes to the FEA script template took the most effort, motivating the expanded procedures. However, reusing the approach used to mesh 1D beam sections motivated the user-defined function for creating plies with a similar approach.

The lack of plies being applied only to the main structure of component 2, suggests the floor of the structure is most responsible for attaining the stiffness targets used in this exploration. This result is more pronounced than first expected by the designers, with very few plies being applied to both components 1 and 2. This motivates further discretization of the geometry for further studies by the chassis designers. Similarly, discretization of ply applications over components can be expanded to cover all combinations of components to allow the furthest exploration of the design space.

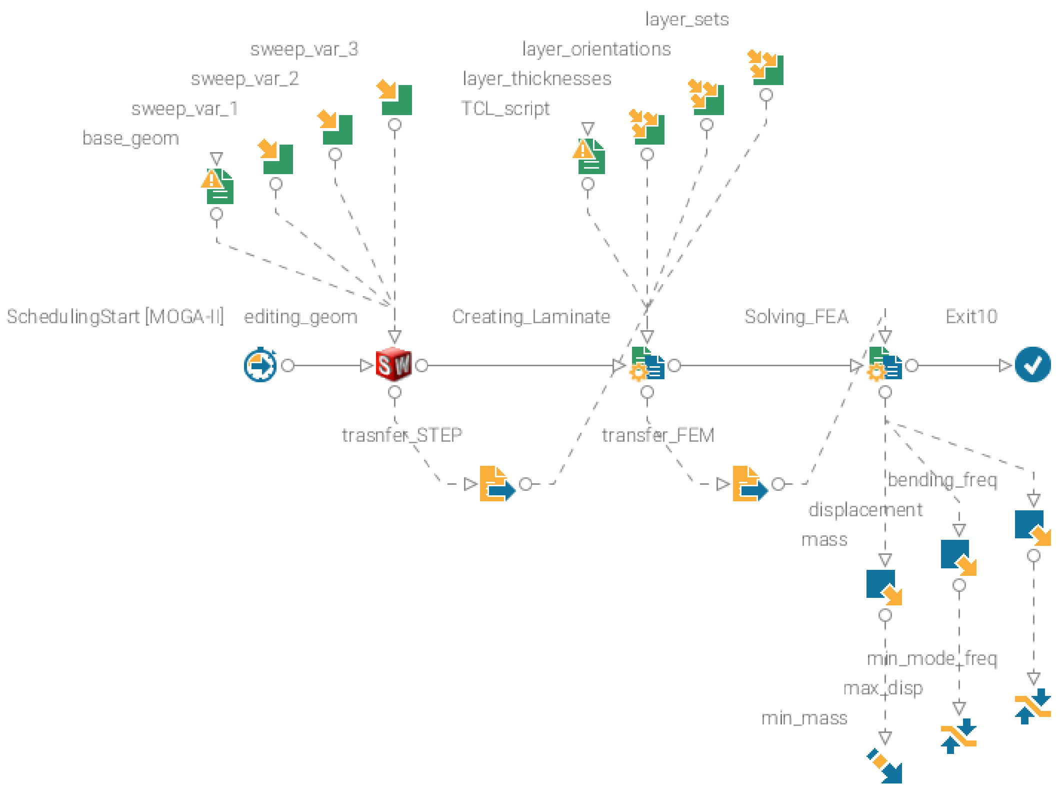


Figure 5.6: Workflow in modeFRONTIER with the additional integrated SolidWorks node and its respective inputs.

5.3 Expanding the Scope of the Exploration

While case study four's design exploration was running, the geometry and knowledge of the design space was being further developed by the chassis designers. Using information from case study four's exploration, the designers expand the automation to include geometry optimization with another section of the geometry discretized for ply application. Case study five therefore, expands on case study four to include geometric optimization like the first three case studies. This expansion of the geometry aims to allow further mass reduction while further expanding the application of the proposed template-based approach. Inputs for this case study are identical to the fourth case study with the addition of inputs for geometric dimensions to update the parametrically defined CAD assembly through modeFRONTIER's CAD node. The use of identical inputs allows the comparison of the initial results from case study four to analyze the impacts of added geometric input variables. Figure 5.6 shows the workflow in the optimization software with the additional CAD node.

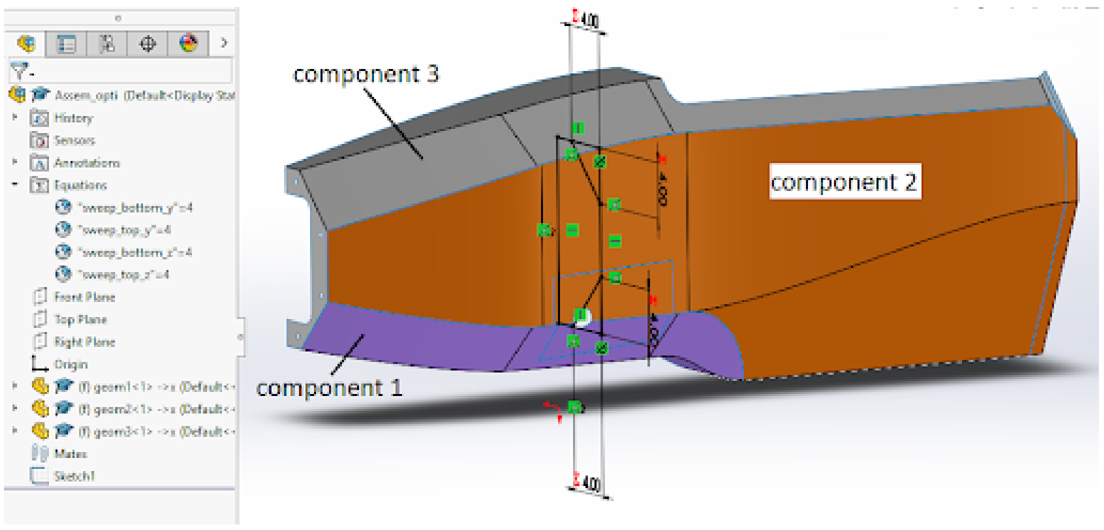


Figure 5.7: CAD assembly used to edit the geometry with four equations available for editing by the optimizer through the SolidWorks node. Each component is shown in three different colors, relating to different laminates over the structure to which plies can be applied by modeFRONTIER.

The largest departure from the first exploration is the addition of geometric optimization. Geometric modification is automated using the previously defined parametric CAD template and adding in a SolidWorks node into the modeFRONTIER template. This study continues the use of a parametrically defined top-down assembly to group surfaces into components when importing into FEA. A top-down approach through an assembly is preferred for this study as grouping surface geometries into SolidWorks components allows direct importing into FEA as individual components. Sketches are defined in the assembly and used to edit the individual component's features top-down. SolidWorks equations allow connection to the modeFRONTIER workflow through the SolidWorks node, like the first three case studies. In this example, three equations defining the cross section of SolidWorks sweep features along the top and bottom of the geometry can be edited by the optimizer. Figure 5.7 shows the SolidWorks assembly with the sketches used for the SolidWorks sweep features and their equations. As the sketches in the assembly are updated, the components are updated accordingly as each component is built of an identical base component using the same relations to the assembly file. This allows relatively robust updating of the geometry where each component is known to have identically defined geometries, driven by the top level assembly. This helps to keep the connectivity of the geometries within the CAD model and eases cleaning connectivity of exported geometries from CAD after importing into FEA.

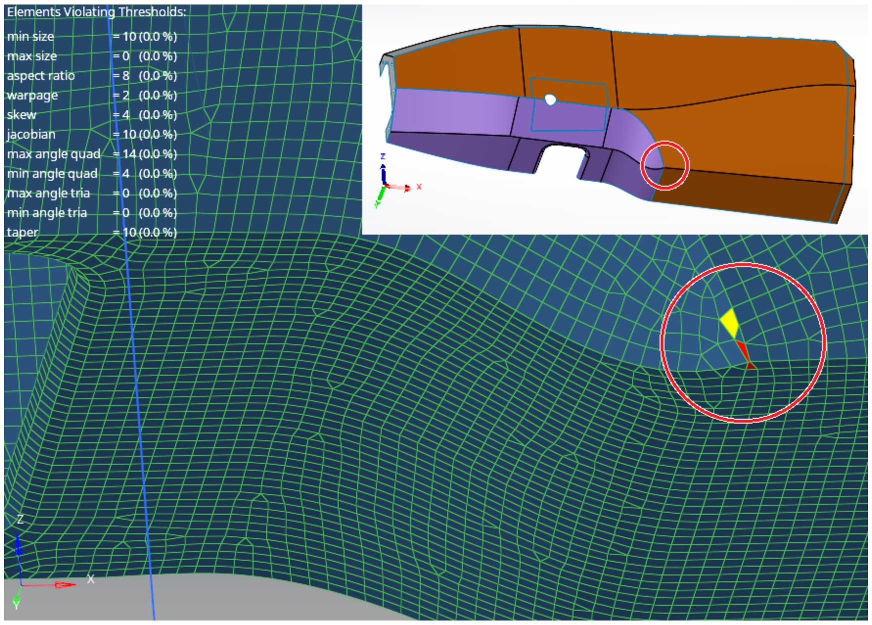
The previous Tcl/Tk template script is expanded back to the initial template with the automated meshing steps defined. The template requires only one update to the user-defined input arrays in Part 1 which apply CFRP plies to the desired geometries. This is to account for the design space increasing to three components, to ensure all combinations of component geometries can be selected. Inputs of zero through two only cover components 1 through 3 respectively. An input of three covers components 1 and 2, an input of four covers components 1 and 3, and an input of five covers components 2 and 3. Finally, an input of six covers all components. The ply creation function `create_ply` in Part 2 is similarly updated to account for the increased input options using if-else statements to apply the plies to the desired geometry, based on the inputted variable. This approach to parameterizing the application of plies presents difficulties when expanding the design space to include many more components. This is a clear direction for improvement in further Tcl/Tk templates.

The re-meshing of the model presents a challenge in keeping connectivity and high mesh quality [9] [27]. To overcome these challenges, the equivalence feature in HyperMesh was used to connect elements meshed surfaces that have broken continuity. The Tcl/tk script was also manually ran with varying mesh sizes to find a range that was generally reliable for a number of edited geometries. This allows the connection of elements along the model while keeping mesh quality to acceptable levels for all iterations tested manually. However, during the optimization, some iterations were found to contain poor quality elements at sections of the body where separate geometries met, highlighted in figure 5.8. This can be avoided in future explorations by defeaturing the geometry in this region to allow easier meshing, or by implementing further tools to cleaning up existing meshes. This is discussed further in later sections of the paper following the optimization.

5.3.1 Results

The optimization was run for 1,000 iterations with a single minimum mass design alternative generated of 18.38 kg. Figure 5.9 plots the design iteration's mass over iteration number.

To review the highest performance design, table 5.2 shows the final input variables generated for the minimum mass design alternative from the second case study. The setup of the table is identical to that of case study four's table, but expanded for the additional component carbon plies can be applied to. Similarly to case study four, the minimum mass design alternatives have a core thickness of 14 mm and the majority of the plies are applied to the first component. Similarly,



```
Elements Violating Thresholds:

min size          = 10 (0.0 %)
max size          = 0   (0.0 %)
aspect ratio      = 8   (0.0 %)
warpage           = 2   (0.0 %)
skew              = 4   (0.0 %)
jacobian          = 10 (0.0 %)
max angle quad    = 14 (0.0 %)
min angle quad    = 4   (0.0 %)
max angle tria    = 0   (0.0 %)
min angle tria    = 0   (0.0 %)
taper             = 10 (0.0 %)
```

Figure 5.8: This highlights a region of the geometry that was found to experience poor quality elements after meshing.

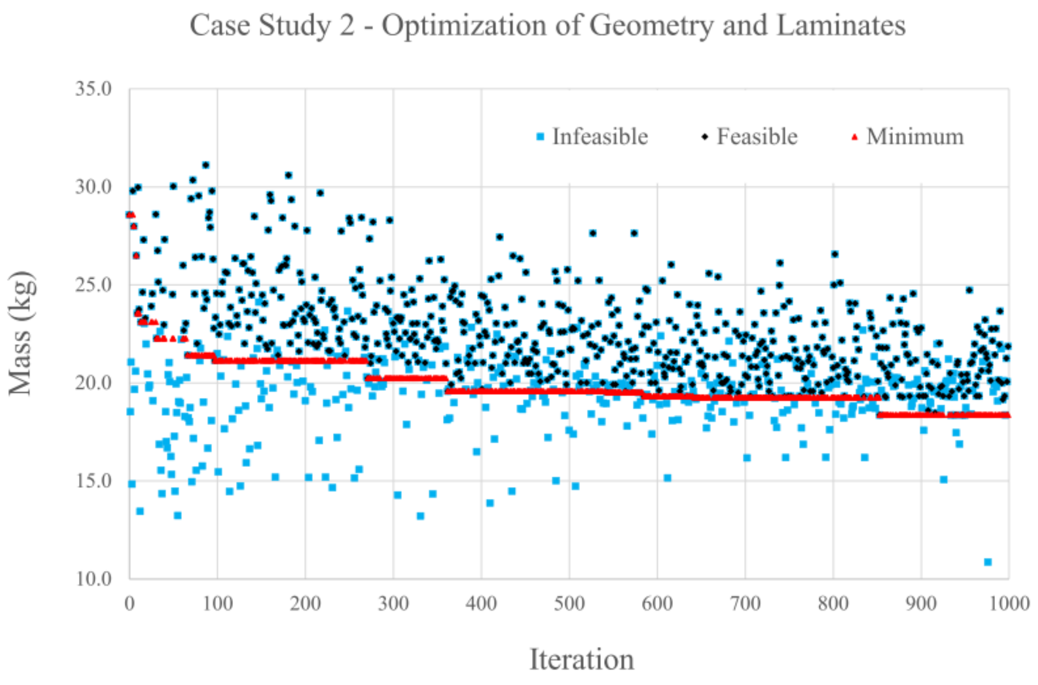


Figure 5.9: In the plot above, infeasible designs are highlighted in blue, feasible are highlighted in black, and the minimum designs by the respective iteration are marked in red. Additionally, three points are highlighted for further discussion below.

Table 5.2: Input variables for one of the minimum mass feasible design alternative.

Ply #	Angle (deg)	Component 1	Component 2	Component 3
0	15			x
1	0		x	
2	45	x	x	
3	90	x		
4	45			x
5	90			x
6	0	x		x
7	15	x	x	
8	75	x		x
9	15	x	x	x
10	60	x		
11	60	x		
12	45	x		x
core	-	x	x	x
13	30	x		x
14	60	x		
15	60	x		x
16	30	x		x
17	15	x		
18	15	x		x
19	60		x	x
20	15	x		
21	0	x		
22	30		x	x

plies are applied to the top of the structure on component 3 suggesting bracing along the top and bottom of the monocoque is the most important for reaching stiffness goals while remaining light weight. These results suggests this approach can be useful in informing design directions even with simplified models and design spaces.

During the optimization, 17 of the 1,000 iterations failed a meshing quality check following changes to the geometry. As discussed earlier, changes to the mesh cleanup process and element sizing were added to address the issue. However, around 2% of the iterations from the optimization fail quality checks, resulting in no outputs for that iteration. To overcome this, an initial quality checking loop may be implemented into the script or optimization software directly. Alternatively, a flag can be scripted to allow manual cleaning of the mesh before moving on for instances such as this where only a small fraction of iterations fail. These approach ensures all models have high quality models meaning no possible designs are skipped over in automation. Additional quality checking steps such as this often are used for applications in computational fluid dynamics [28] or complex FEA models [29]. However, due to the exploratory nature of this optimization, the failed results would likely have minimal impact on the performance of the optimization. The failed designs are further analyzed manually following the optimization by running the generated Tcl/Tk scripts and manually cleaned to pass the quality check. A minimum mass feasible was measured at 22.57 kg suggesting there is likely minimal impact to the final minimum mass generated designs and minimal impact on the design directions suggested by the case study.

5.3.2 Discussion

This case study successfully expanded the design process automation, applying the refined template-based approach from previous studies to implement geometric optimization. To apply the approach to case study five, three changes were required from case study four's setup. The geometry was further discretized in the CAD model template, the meshing process was added back into the FEA script template, the ply creation function in the script's Part 2 was updated for the increased geometries, the modeFRONTIER workflow added the SolidWorks node, and the input arrays were updated to reflect the added geometries. These changes took relatively little time and effort when compared with previous case studies, allowing expansion of the design space motivated by case study four. However, further weaknesses are identified when moving from 1D structures to 2D in the robustness of meshing for FEA. This weakness however, likely has minimal impacts on the early design exploration of stiff, lightweight structures and can be addressed in further revisions to the automation templates.

Overall, this exploration's proposed optimization approach allowed the designers to perform early exploration of CFRP sandwich structures for stiff, lightweight design alternatives. The template-based approach presents a base for the designers to build from, with the case studies suggesting this approach may help designers to reduce the time and effort required to implement automation. Following design work and validation will allow the designers to create a detailed laminate and insert model for evaluation and possible analysis using the proposed optimization approach.

5.4 Summary and Conclusions

This chapter explores the application of template-based automation to composite monocoque structures, taking existing templates and altering them for use with new design problems. The case studies presented suggest this approach can be successfully adapted to a wide range of design problems, guided by the proposed procedures for each template. The design problems increased in complexity from 1D space frame structures to 2D composite models with variables optimizing the layup stacking design over the geometries. However, as design problems vary from the initial template approaches proposed, further alterations and debugging is likely required for robust implementation. This suggests this approach is might best suit iterative design problems that benefit from expanding the design space, when compared to applying the template-based approach to completely unique design problem. However, this chapter highlights the ability for these templates to be adapted to a unique vehicle structural design problem. This suggests further refinement might allow general application to a wide range of structural design problems. Overall, the application of a template-based approach presents the opportunity to reduced time and effort in automating the design process of vehicle structures.

Chapter 6

Conclusions and Discussion

In this thesis, we have investigated a template-based approach for setting up structural analysis and design problems. Through several case studies, tasks have been identified that make automating structural problems for design optimization time-consuming, and proposed a template-based approach for simplifying this process. Supported by a process integration and design optimization tool call modeFRONTIER, the overall process can be broken down into three main steps. These steps can be further defined with procedures that enable the designer to generate or adapt template files for each step, used to automate the design process. First, the CAD geometry must be defined such that each section of the geometry that requires unique properties in the FEA analysis can be grouped and imported as independent components. This allows automation of altering geometries, applying properties, and controlling the distribution of elements in the FEA model. Additionally, the CAD must be parametrized through a part file or top-down assembly level to allow automated editing of the geometry. Second, the FE model is prepared in HyperMesh, where the operations are stored as a Tcl/Tk script. This is where the template-based approach really becomes significant. In this research, we have proposed to edit the structure of the Tcl/Tk script according to a set of procedures that can be easily adapted and modified for different analyses. For instance, a script that was initially set up for design optimization could be reused with minimal modification for sensitivity analysis. Similarly, the same template can quickly applied to different geometries and even different parameterizations with minimal need for modification. In a final step, the template-based analysis is incorporated into a modeFRONTIER optimization workflow model. Again, the optimization workflow can be easily adapted to different CAD geometries and analysis types.

To explore this approach, the design of FSAE vehicle structures were used as case studies. FSAE presented a unique platform for such case studies as the design of the frame is well established, heavily iterative, and has a short design cycle of around three months. The first case study was used to help create the templates refined in further studies while manually setting up the models and workflow. This approach took a large amount of time and effort to apply, due to the inexperience of the designers. However, it allowed a much larger exploration of the design space than previous FSAE vehicles while helping to define the procedures used to generate initial templates for later studies to refine.

The following case study two generated templates using procedures refined from the first case study outlined in chapter three, to explore an expanded design space. The use of the procedures and alteration of case study one models allowed the expansion and refinement of the simulations within the available design cycle. For example, suspension links, modeled joints, and refinement of the load cases were added, further integrating the subsystem development beyond the chassis division. Further manufacturing considerations were also included to minimize post-processing time of optimized results. The optimization generated sets of high-performance feasible design alternatives for the entire FSAE team to discuss, leading to a final product designed to accomplish vehicle targets, rather than independent division targets. This study helped the designers to integrate and expand the exploration of the design space for the chassis and suspension divisions, within the required design window.

These two case studies show promise for applying the proposed template-based approach to design optimization. However, the third case study expanded this beyond optimization of structural designs, by applying the template based approach to a sensitivity analysis of the frame and suspension. The templates and models developed in the second case study were reduced to a quarter car model with CAD parameterization in the form of adjustments of the suspension links and mounting nodes in the X, Y, and Z axes. This was tied to a Matlab script to measure the bending experienced by the suspension links to quantify the impacts of geometric error in manufacturing on the frame and suspension. This study was relatively quick to set up with the majority of the setup effort coming from tediously setting up the modeFRONTIER workflow when connecting input arrays for all the node's X, Y, and Z axes, as well as integrating the outputs for each node's two load cases to the Matlab script. However, this study gave valuable insight into the tolerance selection of the frame and suspension, as well as highlighting critical sensitivities of the geometry to tolerances that were previously unknown. This highlights the adaptability of this template-based approach to the exploration of the design space beyond optimization of design parameters.

With case studies applying the template-based approach to space frame 1D structures, the following case studies four and five explored the application of this approach to more complex 2D sheet material structures. This is explored through the design and optimization of a CFRP sandwich structure monocoque for stiff, lightweight designs. An initial study explored optimization of the laminate stacking, updating the proposed procedures to include modeling of composites for the FEA script template. This optimization took a relatively short amount of time and effort to set up, compared with previous case studies. The majority of alterations were required in updating the FEA script to include the modeling of composites. This optimization was then simplified for faster computation, as the optimization could start from a base, pre-meshed model, only requiring the laminate properties to be created and applied. This initial study allowed early exploration of the design space for the application of plies over the geometry, their orientation, and the thickness of the core. The optimization resulted in a set of high-performance feasible results that helped inform the regions of the structure that contributed most to stiffness.

A following case study five expanded on this work, bringing geometric optimization back into the design process, requiring re-meshing of the geometry and a further expansion of geometries carbon fiber plies could be applied to. This study identified challenges to expanding the approach in the robustness of the model's mesh under certain geometric changes. While this did not likely impact the results from the exploratory study, these limitations should be addressed in future work. However, the study successfully generated high-performance feasible designs with trends carried over from the initial study. The changes made to the Tcl/Tk template were similarly minimal from the previous study with the meshing step and property creation added back into the template. This study highlights possibility to apply this approach to further types of structural design problems.

Design tools such as parametric modeling and analysis allow the designer to explore the design space of structural design problems through simulations. The design and validation process can then be reduced by supplementing physical testing with these models and simulations. However, these tools are often cumbersome to implement manually, motivating the automation of this design process and allowing the designer to explore the design space further. However, application of automation might not be the best option if the cost in setting up exceeds the time available to explore the design space. The template-based approach proposed by this research allows the designer to mitigate set up time when applying automation, by presenting a baseline of design tools, models, and workflows supported by a procedure directing their implementation.

The designer can simplify the setup for automation by implementing standardized methods of CAD preparation, template-based Tcl/Tk scripts, and template modeFRONTIER workflows. Iterative design problems especially provide a good opportunity to apply this approach as they require minimal adaptation for each iteration's design process. This has been explored through the application of a consistent base set of design tools and workflow for optimization and design analysis of varying vehicle structures. For an FSAE frame, the design space was increased from the tens of iterations to the thousands by implementing automation. Following design cycles re-applying the automation templates required less time than the prior manual design cycles, allowing the designers to expand the design space within the same time constraints.

Further applications expanded on this work to 2D CFRP sandwich models, showing the adaptability of the proposed approach. This allowed the exploration of a new form of vehicle structure for the FSAE structural designers, while generating high-performance design alternatives used to study the design space.

To successfully apply this approach, designers must ensure the tools are debugged for automation. This is especially true if new software is introduced, as it is likely to present unique challenges to run reliably. One particular case study experienced this when setting up Matlab nodes in the workflow, requiring specific formatting for use in modeFRONTIER. However, even when unreliability is present in the automation, useful designs and analysis can still be generated through optimization. This is explored in the design exploration of a composite monocoque structure for geometry and analysis where useful high-performance design alternatives were generated despite some iterations failing quality mesh checks.

6.1 Recommendations for Further Research

The implementation of template based models allows designers to simplify the design process for iterative design problems. These models show potential for expansion into new design problems or further simplification of the proposed template-base approach for wider implementation.

A recommendation for further work is to expand the template-based approach beyond structural problems such as heat transfer simulations or CFD. Both of these design problems can be automated through similar analytical software and may benefit from broadly applicable workflow templates. To apply this problem to heat transfer problems, the approach may require alteration to parametrically define the boundary conditions and material properties for the model. Automating CFD for a template-based approach presents unique challenges in robustly meshing the design space and automated interpretation of results. Challenges such as these may be overcome through detailed discretization of the geometric and simulation models to ensure robust creation. Similarly, alternative output strategies or file formats may allow for the automation of interpreting complex results such as flow behavior or efficient thermal transfer through a body.

A user-study would help to quantify the affects of implementing a template-based approach to the design space. This research direction has many variables that may affect the outcome such as the type of problem, difficulty of the problem, time-frame required, experience of the designers, presentation of the problem to the designers, etc. One such study might present a simplified design problem such as the cantilever beam example to a group of designers familiar with the design tools. This simplified example with given inputs, outputs, constraints, and objectives may help to shorten the time requirements, while providing a non-trivial design process. Some designers might be provided with templates for a simplified workflow for a similar problem, while the others may act as a control group. A study such as this might also highlight bottlenecks in design such as design fixation or the tendency of designers to jump into detailed design before adequately breaking down the problem.

This research was additionally refined while working with Clemson University's International Center for Automotive Research on their Deep Orange structure team. The Deep Orange project designs and builds a unique vehicle concept in two years, requiring fast design cycles for all sections of the vehicle. Work with the team highlighted the need for automation of the design process and allowed the refinement of the models presented in this work. However, as Deep Orange creates unique projects each year, the application of this approach is more difficult for incoming designers. I therefore recommend that a simplification of the templates used, may allow designers with unique design problems to more widely apply this method of automation.

Bibliography

[1] Hong-Seok Park and Xuan-Phuong Dang. Structural optimization based on CAD–CAE integration and metamodeling techniques. *Computer-Aided Design*, 42:889–902, 10 2010.

[2] Timothy Simpson and Joaquim Martins. Multidisciplinary Design Optimization for Complex Engineered Systems: Report From a National Science Foundation Workshop. *Journal of Mechanical Design*, 133, 10 2011.

[3] Tong S. Nicklaus, D. and D. Russo. Engineous, A Knowledge Directed Computer Aided Design Shell. 2 1987.

[4] Mohammad Fard, Jianchun Yao, Richard Taube, and John Laurence Davy. The Concept Modeling Method: An Approach to Optimize the Structural Dynamics of a Vehicle Body. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 234(13):2923–2932, 2020.

[5] D. Mundo, R. Hadjit, S. Donders, M. Brughmans, P. Mas, and W. Desmet. Simplified Modelling of Joints and Beam-like Structures for BIW Optimization in a Concept Phase of the Vehicle Design Process. *Finite Elements in Analysis and Design*, 45(6):456 – 462, 2009.

[6] Peter Persson, Ola Flodén, and Björn Pedersen. Predicting Vibroacoustic Performance of Thin-Walled Lightweight Structures During Conceptual Design. *Finite Elements in Analysis and Design*, 169:103342, 2020.

[7] A. A. Korostelkin, O. Klyavin, M. Aleshin, Wang Guo-dong, Wang Suifeng, and L. Jing-yi. Optimization of Frame Mass in Crash Testing of Off-Road Vehicles. *Russian Engineering Research*, 39:1021–1028, 2019.

[8] Lightweight SUV Frame Design Development - Altair Enlighten. May 2003.

[9] Fabian Duddeck and H Zimmer. New Achievements on Implicit Parameterization Techniques for Combined Shape and Topology Optimization for Crashworthiness based on SFE Cncept. 07 2012.

[10] Abbas Bayatfar, Jose Mishael, Renaud Warnotte, and Philippe Rigo. An Integrated Framework for Ship Structural Optimisation in Contract Design Phase. 42:109–116, 12 2019.

[11] J. Rao. *Recent Advances In Optimization Of Aerospace Structures And Engines*, pages 323–333. 10 2008.

[12] T. Stangl, M. Pribek, and S. Wartzack. Integration Of Structural Optimization in the Engineering Design Process. 2014.

[13] Paul Gembarski, Haibing Li, and R. Lachmayer. Template-Based Modelling of Structural Components. *International Journal of Mechanical Engineering and Robotics Research*, 6:336–342, 09 2017.

[14] M. Tarkian. Design Automation for Multidisciplinary Optimization : A High Level CAD Template Approach. 2012.

[15] Libin Duan, Haobin Jiang, Guoqing Geng, Xuerong Zhang, and Zhanjiang Li. Parametric Modeling and Multiobjective Crashworthiness Design Optimization of a New Front Longitudinal Beam. *Structural and Multidisciplinary Optimization*, 59, 05 2019.

[16] R. Galgalikar. Design Automation and Optimization Of Honeycomb Structures for Maximum Sound Transmission Loss. *All Theses*, page 1514, 2012.

[17] A. Mönicke, H. Katajisto, M. Leroy, N. Petermann, P. Kere, and Marco Perillo. Design-Optimization of Cylindrical, Layered Composite Structures using Efficient Laminate Parameterization. *European Space Agency, (Special Publication) ESA SP*, 691, 01 2012.

[18] Ziad K. Awad, Thiru Aravinthan, Yan Zhuge, and Felipe Gonzalez. A Review of Optimization Techniques used in the Design of Fibre Composite Structures for Civil Engineering Applications. *Materials Design*, 33:534 – 544, 2012.

[19] Dawei Gao, Haotian Liang, Guijie Shi, and Liqin Cao. Multiobjective Optimization of Carbon Fiber-Reinforced Plastic Composite Bumper Based on Adaptive Genetic Algorithm. *Mathematical Problems in Engineering*, 2019:1–12, 11 2019.

[20] Do-Hyoung Kim, Dong-Hoon Choi, and Hak-Sung Kim. Design Optimization of a Carbon Fiber Reinforced Composite Automotive Lower Arm. *Composites Part B: Engineering*, 58:400 – 407, 2014.

[21] Jianguang Fang, Guangyong Sun, Na Qiu, Nam Kim, and Qing Li. On Design Optimization for Structural Crashworthiness and its State of the Art. *Structural and Multidisciplinary Optimization*, 55:1091–1119, 03 2017.

[22] Jami Shah. *Designing with Parametric CAD: Classification and Comparison of Construction Techniques*, pages 53–68. 01 2001.

[23] Jami J Shah. Philosophical Development of Form Feature Concept. *CAM-I report P-90-PM-02*, pages 55–70, 1990.

[24] Review of Research in Feature-Based Design. *Journal of Manufacturing Systems*, 12(2):113 – 132, 1993.

[25] Fanter K. Spiess B., Höschler K. A Feature-Based Approach for an Automated Simplification of Structural Aero Engine Components. 2020.

[26] P. Beardmore and C.F. Johnson. The Potential for Composites in Structural Automotive Applications. *Composites Science and Technology*, 26(4):251 – 281, 1986.

[27] Mezentsev, Andrey and Woehler, Thomas. Methods and algorithms of automated cad repair for incremental surface meshing. 03 2000.

[28] Tommaso Lucchini, Augusto Della Torre, Gianluca D'Errico, Gianluca Montenegro, Marco Fiocco, and Amin Maghbouli. Automatic Mesh Generation for CFD Simulations of Direct-Injection Engines. volume 2015, 01 2015.

[29] C. Groth, A. Chiappa, and M.E. Biancolini. Shape optimization using Structural Adjoint and RBF Mesh Morphing. *Procedia Structural Integrity*, 8:379 – 389, 2018. AIAS2017 - 46th Conference on Stress Analysis and Mechanical Engineering Design, 6-9 September 2017, Pisa, Italy.

Appendices

Appendix A Cantilever Beam Example - Tcl/Tk Script Template

```
# William McCormack - ME Masters Thesis
# tcl Script for meshing example cantilever beam
# Design template script for modeFRONTIER automation

# Initializing OptiStruct
*templatefileset "C:/Program Files/Altair/2019/templates/feoutput/optistruct/optistruct"



# ****** Section 1: User-Defined Inputs ******

# Inputted name for CAD geometry and scaling factor (if CAD is in metric = 1, imperial = 25.4)
set CAD_geometry "geom1.igs"
set Scale_Factor 25.4

# *** Creating Beam-Section Variables ***
# Mesh size for length of 1D elements (mm)
set mesh_size_1D 20

# Selecting the type of beam to test (1 = round, 2 = box)
set beam_type {2 1 1}

# Setting outer dimension (OD), local to Z-axis, (0 1 2 3 4 5) = (0.5 0.625 0.75 0.787 1.0 1.125)
set beam_OD {1 1 1}

# Setting thickness (t) beam dimensions
set beam_t {0.049 0.095 0.035}

# Setting loop variables for number of beams and components
set beam_total [expr {[llength $beam_type]}]
set comp_no [expr {$beam_total + 4}]


# *** Creating Plate Variables ***
# Mesh size for length of 2D elements (mm)
set mesh_size_2D 2

# Plate thickness
set plate_t 4.0

# *** input geom vars ***
set dim1 8.0;
set hole_height 0.0;


# *** Creating Geometry Variables for Finding Desired Nodes ***
# Mounting hole dimensions for creating 6 DOF fixed SPC (inch)
set hole_width 5
set hole_height 5


# *** Creating Loading Variables for Script Organization and Cleanup ***
# Setting array of loadcoll name strings for organization
set loadcoll_names {Load1 Load2}
set spccoll_names {SPC1 SPC2}

# Setting array of loadstep name strings for organization
set loadstep_names {LoadCase1 LoadCase2}

# Creating loading magnitude variables
# Load 1 - inputted in X, Y, Z coordinates
set load1 {}; lappend load1 200.0 300.0 -1000.0

# Load 2 - inputted in X, Y, Z coordinates
set load2 {}; lappend load2 0.0 -300.0 200.0


# *** Creating Material Properties ***
# Steel property card (E, G, nu, rho)
*createentity mats cardimage=MAT1 name=steel
*setvalue mats id=1 name=steel
*setvalue mats id=1 STATUS=1 1=210000 # GPa
*setvalue mats id=1 STATUS=1 2=80000 # GPa
*setvalue mats id=1 STATUS=1 3=0.3 # n/a
*setvalue mats id=1 STATUS=1 4=7.85e-09 # tonne/mm3



# ******************************************************************************
# *********************** End of User Defined Variables ***********************
# ******************************************************************************



# ****** Section 2: User-Defined Functions ******
# *** Adding Beam-Section creation command to .tcl script ***
proc beam_make {type OD t iter type_no mat} {
        # Setting OD and t
        set OD [expr {$OD * 25.4}]
        set t [expr {$t * 25.4}]

        # IF the desired beam is round
        if { $type == 1 } {
                set OD [expr {$OD / 2}];
                set ID [expr {$OD - $t}];
                *beamsectioncreatestandardsolver 7 1 "Tube" 0
                *beamsectionsetdataroot $iter 1 1 2 7 1 0 1 1 0 0 0 0
                *createdoublearray 6 $ID 4 4 $OD 5 5
                *beamsectionsetdatastandard 1 6 $iter 7 0 "Tube"
                *createmark beamsects 1 "tube_section.$iter"
                *updatehmdb beamsects 1
                set type_no [expr {$type_no + 1}];
                # Creating props
                *createentity props cardimage=PSHELL name=property1
                *setvalue props id=$iter cardimage="PBEAML"
                *setvalue props id=$iter name="Rnd_$type_no"
                *setvalue props id=$iter materialid={mats $mat}
                *setvalue props id=$iter STATUS=2 3186={beamsects $iter}
                *createmark properties 1 "Rnd_$type_no"
        # ELSE, IF the desired beam is box
        } elseif { $type == 2 } {
                *beamsectioncreatestandardsolver 0 1 "Box" 0
                *beamsectionsetdataroot $iter 1 1 2 7 1 0 1 1 0 0 0 0
                *createdoublearray 12 $t 0.5 0.5 $OD 10 10 $OD 10 10 $t 0.5 0.5
                *beamsectionsetdatastandard 1 12 $iter 0 0 "Box"
                *createmark beamsects 1 "box_section.$iter"
                *updatehmdb beamsects 1
                set type_no [expr {$type_no + 1}];
                # Creating props
                *createentity props cardimage=PSHELL name=property1
                *setvalue props id=$iter cardimage="PBEAML"
                *setvalue props id=$iter name="Box_$type_no"
                *setvalue props id=$iter materialid={mats $mat}
                *setvalue props id=$iter STATUS=2 3186={beamsects $iter}
                *createmark properties 1 "Box_$type_no"
        } else {
                set temp2 0;
        }

        # Finishing setting up BeamSections
        *syncpropertybeamsectionvalues 1
        *setvalue props id=$iter STATUS=2 500=1
        *setvalue props id=$iter STATUS=0 36=0
        *setvalue props id=$iter STATUS=0 37=0
        *setvalue props id=$iter STATUS=0 38=0
        *setvalue props id=$iter STATUS=0 39=0

        # Returning var
        return $type_no;
}

# *** HM get closest node function - cleaning up the code ***
# Inputs:
#         - comp_number = component ID# to ensure grabbing nodes from desired component
#         - node_coords = X Y Z coordinates of the node being grabbed inputted as an array
#         - option1 / option2 are two options for the builtin function hm_getclosestnode
#         - axis_sign = sign of the coordinates on their axes as an array {X Y Z}
proc find_node_number {comp_number node_coords option1 option2 axis_sign} {
        *createmark elems 1 "by comp id" $comp_number
        set node_number [hm_getclosestnode [expr {[lindex $axis_sign 0]*[lindex $node_coords 0]}] ...
            ... [expr {[lindex $axis_sign 1]*[lindex $node_coords 1]}] ...
            ... [expr {[lindex $axis_sign 2]*[lindex $node_coords 2]}] $option1 $option2]
        *clearmark elems 1

        # Returning var
        return $node_number
}

# *** Creating SPC Load Collector, fixing DOFs ($DOFs string) at nodes ($node_nums) ***
proc create_SPC {node_nums DOFs} {
        # Creating BC
        for {set i 0} {$i < [llength $node_nums] } {incr i} {
                # Setting size of icon in GUI
                *loadsize 3 50 1 1

                # Grabbing desired nodes by inputted node number
                *createmark nodes 1 [lindex $node_nums $i]

                # Creating SPC BC
                *loadcreateonentity_curve nodes 1 3 1 [expr {-999999*[lindex $DOFs 0]}] ...
                    ... [expr {-999999*[lindex $DOFs 1]}] [expr {-999999*[lindex $DOFs 2]}] ...
                    ... [expr {-999999*[lindex $DOFs 3]}] [expr {-999999*[lindex $DOFs 4]}] ...
                    ... [expr {-999999*[lindex $DOFs 5]}] 0 0 0 0 0
                *createmark loads 0 1 2
                *loadsupdatefixedvalue 0 0
        }
}

# *** Creating Force Load Collector and applying Load at desired nodes ***
proc create_force {node_nums loads_X loads_Y loads_Z} {
        # Creating BC
        for {set i 0} {$i < [llength $node_nums] } {incr i} {
                # Calculating magnitude of X Y Z loading
                set load_magn [expr {sqrt(pow([lindex $loads_X $i], 2) + ...
                    ... pow([lindex $loads_Y $i], 2) + pow([lindex $loads_Z $i], 2))}]

                # Grabbing desired nodes by inputted node number
                *createmark nodes 1 [lindex $node_nums $i]

                # Creating force BC
                *loadcreateonentity_curve nodes 1 1 1 [lindex $loads_X $i] ...
                    ... [lindex $loads_Y $i] [lindex $loads_Z $i] 0 0 $load_magn 0 0 0 0 0
        }
}



# ****** Section 3: Importing and Preparing Geometry ******
*start_batch_import 3
*setgeomrefinelevel 1
*geomimport "auto_detect" "./$CAD_geometry" "CleanupTol=-0.01" "CreationType=TreeOfComponents" ...
    ... "DoNotMergeEdges=on" "ImportBlanked=on" "ScaleFactor=$Scale_Factor" "TargetUnits=Scale factor"
*end_batch_import
*drawlistresetstyle
# *** Creating Beam-Sections and Properties ***
# Initializing vars
set beam_rnd_no 0; set beam_box_no 0; set beam_shell_no 0;

# Looping for number of beams and creating Beam-Section / Property
for {set beam_no 0} {$beam_no < $beam_total } {incr beam_no} {
        set beam_no2 [expr {$beam_no + 1}];

        # If statement to create the properties based off beam-shape input array
        if { [lindex $beam_type $beam_no] == 1} {
                set beam_rnd_no [beam_make [lindex $beam_type $beam_no] ...
                    ... [lindex $beam_OD $beam_no] [lindex $beam_t $beam_no] ...
                    ... $beam_no2 $beam_rnd_no 1];
        } elseif { [lindex $beam_type $beam_no] == 2} {
                set beam_box_no [beam_make [lindex $beam_type $beam_no] ...
                    ... [lindex $beam_OD $beam_no] [lindex $beam_t $beam_no] ...
                    ... $beam_no2 $beam_box_no 1];
        } else {
                set temp3 = 0;
        }
}

# Creating plate property
*createentity props cardimage=PSHELL name=property1
*setvalue props id=[expr {$beam_total + 1}] name="plate"
*setvalue props id=[expr {$beam_total + 1}] materialid={mats 1}
*setvalue props id=[expr {$beam_total + 1}] STATUS=1 95=$plate_t


# *** Prepping model for meshing ***
# Hiding sketches
for {set i 1} {$i <= $beam_total} {incr i} {
        *createmark components 3 "3DSKETCH.$i"
        *createstringarray 2 "elements_off" "geometry_on"
        *hideentitybymark 3 1 2
}

# Hiding extra 'points', 'lvl0', and "3DSKETCH"
*createmark components 3 "Points" "lvl0" "3DSKETCH"
*createstringarray 2 "elements_off" "geometry_on"
*hideentitybymark 3 1 2

# Creating new component for Meshed elements
*createentity comps includeid=0 name=meshed_model
*createmark components 1 "meshed_model"
*clearmark components 1

# Applying palte property to it for 2D elements
```

```
*setvalue comps id=$comp_no propertyid={props [expr {$beam_total + 1}]}



# ****** Section 4: Meshing and Applying Element Properties ******
# Meshing 1D Beams
for {set i 0} {$i < $beam_total } {incr i} {
        set j [expr {$i + 1}]
        *createmark components 3 "3DSKETCH.$j"
        *createstringarray 2 "elements_off" "geometry_on"
        *showentitybymark 3 1 2
        *elementsizeset $mesh_size_1D
        *createmark lines 1 "displayed"
        *linemesh_preparedata1 lines 1 0 60
        *createvector 1 1 1 0
        *linemesh_savedata_bar1 lines 1 60 $j 1 0 0 0 0 0 0 0 0
        *createmark components 3 "3DSKETCH.$j"
        *createstringarray 2 "elements_off" "geometry_on"
        *hideentitybymark 3 1 2
}


# *** Meshing 2D surfaces in "lvl0" component ***
*createmark components 3 "lvl0"
*createstringarray 2 "elements_off" "geometry_on"
*showentitybymark 3 1 2

# Meshing "displayed" geometries
#*setedgedensitylinkwithaspectratio -1
#*elementorder 1
*createmark surfaces 1 "displayed"
*interactiveremeshsurf 1 $mesh_size_2D 2 2 2 1 1
*set_meshfaceparams 0 5 2 0 0 1 0.5 1 1
*automesh 0 5 2
*set_meshfaceparams 1 5 2 0 0 1 0.5 1 1
*automesh 1 5 2
*storemeshtodatabase 0
*ameshclearsurface


# *** Cleaning and Merging Elements ***
# Clearing temp nodes
*nodecleartempmark

# Cleaning elements for connectivity with 'equivalance'
*createmark components 1 "all"
*equivalence components 1 0.05 1 0 0




# ****** Section 6: Adding BCs ******

# *** Calculating coordinates for mounting bolts --> used to breate SPC in 6 DOF ***
# Organized into X, Y, Z coordinates, appended by hole # --> holes 1, 2, 3, 4
set mount_holes {};
lappend mount_holes 0.0 [expr {$hole_width*25.4/2}] -[expr {$hole_height*25.4/2}]
lappend mount_holes 0.0 -[expr {$hole_width*25.4/2}] -[expr {$hole_height*25.4/2}]
lappend mount_holes 0.0 [expr {$hole_width*25.4/2}] [expr {$hole_height*25.4/2}]
lappend mount_holes 0.0 -[expr {$hole_width*25.4/2}] [expr {$hole_height*25.4/2}]

# Finding mounting hole nodes for use in SCP
# Initializing arrays for grabbed nodes
set SPC_nodes {};
```

```

# Looping to grab desired SCP nodes for SCP
for {set i 0} {$i < 4} {incr i} {
        # Counter that multiplies by 3 for each set of X,Y,Z coordinates
        set j [expr {$i*3}]

        # Grabbing left outboard CA nodes
        lappend SPC_nodes [find_node_number $comp_no [lrange $mount_holes $j ...
            ... [expr {$j+2}]] 1 0 {1 1 1}]
}

# Finding node for applying the loads
set load_XYZ {}; lappend load_XYZ [expr {($dim1 + 1)*25.4}] 0...
        ... [expr {(($hole_height/2) + 0.5 - 1.5)*25.4}]
set load_node [find_node_number $comp_no $load_XYZ 1 0 {1 1 1}]

# Creating SCP1 load collector
*createentity loadcols name=[lindex $spccoll_names 0]
# Creating SPC in all 6 DOF
create_SPC $SPC_nodes {0 0 0 0 0 0}

# Creating load 1 collector
*createentity loadcols name=[lindex $loadcoll_names 0]
# Creating load 1
create_force $load_node [lindex $load1 0] [lindex $load1 1] [lindex $load1 2]

# Creating SCP2 load collector
*createentity loadcols name=[lindex $spccoll_names 1]
# Creating SPC in all 6 DOF
create_SPC $SPC_nodes {0 0 0 0 0 0}

# Creating load 2 collector
*createentity loadcols name=[lindex $loadcoll_names 1]
# Creating load 2
create_force $load_node [lindex $load2 0] [lindex $load2 1] [lindex $load2 2]


# *** Creating Loadsteps ***
# Creating LoadStep variables --> number of loadsteps
set loadstep_no [llength $loadstep_names]

# Looping to create LoadSteps
set loadstep_name_counter 0
for {set i 1} {$i < [expr {$loadstep_no + 1}]} {incr i} {
        *createentity loadsteps name=[lindex $loadstep_names $loadstep_name_counter]
        set loadstep_name_counter [expr {$loadstep_name_counter + 1}]
        *setvalue loadsteps id=$i STATUS=2 3240=1
        *setvalue loadsteps id=$i STATUS=2 289=0
        *setvalue loadsteps id=$i STATUS=2 288=0
        *setvalue loadsteps id=$i STATUS=2 4059=0
        *setvalue loadsteps id=$i STATUS=2 710=0
        *setvalue loadsteps id=$i STATUS=2 4347=0
        *setvalue loadsteps id=$i STATUS=2 4034=0
        *setvalue loadsteps id=$i STATUS=2 4037=0
        *setvalue loadsteps id=$i STATUS=2 9891=0
        *setvalue loadsteps id=$i STATUS=2 4230=0
        *setvalue loadsteps id=$i STATUS=2 10089=0
        *setvalue loadsteps id=$i STATUS=2 10079=0
        *setvalue loadsteps id=$i STATUS=2 9599=0
        *setvalue loadsteps id=$i STATUS=2 2315=0
        *setvalue loadsteps id=$i STATUS=2 1150=0
        *setvalue loadsteps id=$i STATUS=2 1182=0
        *setvalue loadsteps id=$i STATUS=2 1168=0
        *setvalue loadsteps id=$i STATUS=2 10701=0
```

```
    *setvalue loadsteps id=$i STATUS=2 7202=0
    *setvalue loadsteps id=$i STATUS=2 8142=0
    *setvalue loadsteps id=$i STATUS=2 4722=0
    *setvalue loadsteps id=$i STATUS=2 10839=0
    *setvalue loadsteps id=$i STATUS=2 3391=0
    *setvalue loadsteps id=$i STATUS=2 3396=0
    *setvalue loadsteps id=$i STATUS=2 7408=0
    *setvalue loadsteps id=$i STATUS=2 8897=0
    *setvalue loadsteps id=$i STATUS=2 9416=0
    *setvalue loadsteps id=$i STATUS=2 351=0
    *setvalue loadsteps id=$i STATUS=2 3292=0
    *setvalue loadsteps id=$i 4709=1 STATUS=1
    *setvalue loadsteps id=$i STATUS=2 4059=1
    *setvalue loadsteps id=$i STATUS=2 4060=BUCK
    *setvalue loadsteps id=$i 707=0 STATUS=0
    *setvalue loadsteps id=$i 9293={Loadcols 0} STATUS=0
    *setvalue loadsteps id=$i 4709=1 STATUS=1
    *setvalue loadsteps id=$i STATUS=2 4059=1
    *setvalue loadsteps id=$i STATUS=2 4060=STATICS
    *setvalue loadsteps id=$i 707=0 STATUS=0
    *setvalue loadsteps id=$i 9293={Loadcols 0} STATUS=0
    *setvalue loadsteps id=$i STATUS=2 4152=0
    *setvalue loadsteps id=$i STATUS=2 4973=0
    *setvalue loadsteps id=$i STATUS=2 4143=1
    *setvalue loadsteps id=$i 4144=1 STATUS=1
    *setvalue loadsteps id=$i 4145={Loadcols [expr {($i * 2) - 1}]} STATUS=1
    *setvalue loadsteps id=$i STATUS=2 4143=1
    *setvalue loadsteps id=$i 4146=1 STATUS=1
    *setvalue loadsteps id=$i 4147={Loadcols [expr {$i * 2}]} STATUS=1
    *setvalue loadsteps id=$i 7763=0 STATUS=0
    *setvalue loadsteps id=$i 7740={Loadcols 0} STATUS=0
    *setvalue loadsteps id=$i STATUS=2 351=1
    *setvalue loadsteps id=$i STATUS=2 3321=0
    *setvalue loadsteps id=$i STATUS=2 9630=0
    *setvalue loadsteps id=$i STATUS=2 9307=0
    *setvalue loadsteps id=$i STATUS=2 9317=0
    *setvalue loadsteps id=$i STATUS=2 9327=0
    *setvalue loadsteps id=$i STATUS=2 4119=0
    *setvalue loadsteps id=$i STATUS=2 4114=0
    *setvalue loadsteps id=$i STATUS=2 7121=0
    *setvalue loadsteps id=$i STATUS=2 2938=0
    *setvalue loadsteps id=$i STATUS=2 10688=0
    *setvalue loadsteps id=$i STATUS=2 2385=0
    *setvalue loadsteps id=$i STATUS=2 4052=0
    *setvalue loadsteps id=$i STATUS=2 3712=0
    *setvalue loadsteps id=$i STATUS=2 274=0
    *setvalue loadsteps id=$i STATUS=2 3057=0
    *setvalue loadsteps id=$i STATUS=2 10833=0
    *setvalue loadsteps id=$i STATUS=2 7113=0
    *setvalue loadsteps id=$i STATUS=2 8500=0
    *setvalue loadsteps id=$i STATUS=2 2419=0
    *setvalue loadsteps id=$i STATUS=2 8493=0
    *setvalue loadsteps id=$i STATUS=2 9709=0
    *setvalue loadsteps id=$i STATUS=2 3809=0
    *setvalue loadsteps id=$i STATUS=2 7125=0
    *setvalue loadsteps id=$i STATUS=2 4877=0
    *setvalue loadsteps id=$i STATUS=2 9337=0
    *setvalue loadsteps id=$i STATUS=2 9347=0
    *setvalue loadsteps id=$i STATUS=2 9357=0
    *setvalue loadsteps id=$i STATUS=2 3325=0
    *setvalue loadsteps id=$i STATUS=2 7093=0
    *setvalue loadsteps id=$i STATUS=2 3333=0
    *setvalue loadsteps id=$i STATUS=2 2423=0
    *setvalue loadsteps id=$i STATUS=2 4887=0
```

```
        *setvalue loadsteps id=$i STATUS=2 4047=0
        *setvalue loadsteps id=$i STATUS=2 9275=0
        *setvalue loadsteps id=$i STATUS=2 5463=0
        *setvalue loadsteps id=$i STATUS=2 8949=0
        *setvalue loadsteps id=$i STATUS=2 10440=0
        *setvalue loadsteps id=$i STATUS=2 7329=0
        *setvalue loadsteps id=$i STATUS=2 7333=0
        *setvalue loadsteps id=$i STATUS=2 2427=0
        *setvalue loadsteps id=$i STATUS=2 8153=0
        *setvalue loadsteps id=$i STATUS=2 8150=0
        *setvalue loadsteps id=$i STATUS=2 8144=0
        *setvalue loadsteps id=$i STATUS=2 3642=0
        *setvalue loadsteps id=$i STATUS=2 2431=0
        *setvalue loadsteps id=$i STATUS=2 7337=0
        *setvalue loadsteps id=$i STATUS=2 7117=0
        *setvalue loadsteps id=$i STATUS=2 3329=0
        *setvalue loadsteps id=$i STATUS=2 2938=1
        *setvalue loadsteps id=$i STATUS=0 1901=1
        *setvalue loadsteps id=$i ROW=0 STATUS=2 4871= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 4315= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 4008= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 4876= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 2174= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 2287= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 2175= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 9621= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 10026= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 10027= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 11069= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 2939= {ALL}
        *setvalue loadsteps id=$i ROW=0 STATUS=2 4315= {PUNCH}
        *setvalue loadsteps id=$i STATUS=2 2431=1
        *setvalue loadsteps id=$i STATUS=0 1923=1
        *startnotehistorystate {Updated string array}
        *setvalue loadsteps id=$i ROW=0 STATUS=2 4873= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 4325= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 3386= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 3387= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 4839= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 1221= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 11070= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 2295= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 8136= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 8430= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 9932= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 8429= {        }
        *setvalue loadsteps id=$i STATUS=0 9254={0}
        *setvalue loadsteps id=$i STATUS=0 9255={0}
        *setvalue loadsteps id=$i STATUS=0 9280={0}
        *setvalue loadsteps id=$i STATUS=0 9281={0}
        *setvalue loadsteps id=$i ROW=0 STATUS=2 11072= {        }
        *setvalue loadsteps id=$i ROW=0 STATUS=2 4873= {SORT1}
        *setvalue loadsteps id=$i STATUS=1 9280={2}
        *setvalue loadsteps id=$i ROW=0 STATUS=2 2432= {ALL}
        *setvalue loadsteps id=$i ROW=0 STATUS=2 4325= {PUNCH}
        *setvalue loadsteps id=$i ROW=0 STATUS=2 3387= {VON}
}


# *** Grabbing node 1 and renumbering to be well above other node IDs ***
*createmark nodes 1 1

*renumbersolverid nodes 1 10000 1 1 0 0 0 0

```

```
# Renumbering inboard CA nodes
*createmark nodes 1 $load_node
*renumbersolverid nodes 1 1 1 1 0 0 0 0

# *** Setting Output from Optistruct  ***
*createstringarray 1 "CONNECTORS_SKIP "
*feoutputwithdata "C:/Program Files/Altair/2019/templates ...
        ... /feoutput/optistruct/optistruct" ./output_fem.fem
```

Appendix B Tcl/Tk User-Defined Function for Ply Creation

```
# Ply creation function
# Inputs:
#       - ply_name         = name of ply for design tree organization
#        - ply_t         = ply thickness (mm)
#        - ply_angle        = angle of ply orientation
#       - ply_set         = array of component IDs - used to grab desired elements for the ply's set ...
            ... (0 = only geom 1, 1 = geom 2, etc.)
#       - ply_no        = ply number used for indexing $ply_sets matrix's rows --> ply arrays
#        - mat_names        = name of material prop to assign to ply
proc create_ply {ply_name ply_t ply_angle ply_set ply_no ply_mat} {
        # Setting elements plies will cover
        if { $ply_set == 0 } {
                set ply_elem {0 1}
        } elseif { $ply_set == 1 } {
                set ply_elem {0 2}
        } elseif { $ply_set == 2 } {
                set ply_elem {0 3}
        } elseif { $ply_set == 3 } {
                set ply_elem {1 2}
        } elseif { $ply_set == 4 } {
                set ply_elem {1 3}
        } elseif { $ply_set == 5 } {
                set ply_elem {2 3}
        } elseif { $ply_set == 6 } {
                set ply_elem {1 2 3}
        } else {
                set ply_elem {0 0}
        }

        *createmark lines 1
        *clearmark lines 1
        *createmark surfaces 1
        *clearmark surfaces 1
        *createmark sets 1
        *clearmark sets 1
        # Grabbing elements for the ply sets, grouped elements by component ID# in $ply_sets array
        *clearmark
        *createmark elements 1 "comps" "geom[lindex $ply_elem 0]"
        for {set i 1} {$i < [llength $ply_elem]} {incr i} {
                if {[lindex $ply_elem $i]  > 0} {
                        *appendmark elements 1 "comps" "geom[lindex $ply_elem $i]"
                }
        }
        *entitysetcreate $ply_name elements 1
        *createmark sets 2
        *clearmark sets 2
        *createmark sets 2 $ply_name
        *dictionaryload sets 2 "C:/Program Files/Altair/2019/templates/ ...
            ... feoutput/optistruct/optistruct" "SET_ELEM"
        *initializeattributes sets $ply_name
        *createmark sets 1
        *clearmark sets 1
        *createmark sets 1 $ply_name
        *plycreate $ply_name 5 sets 1 $ply_mat $ply_t $ply_angle 3 1 0 0 0 0 0
        *createmark plies 1
        *clearmark plies 1
        *createmark plies 1 $ply_name
        *dictionaryload plies 1 "C:/Program Files/Altair/2019/templates/ ...
            ... feoutput/optistruct/optistruct" "PLY"
        *initializeattributes plies $ply_name
}
```