REVIEW AND DEVELOPMENT OF CENTRALIZED ENGINEERING TOOL FRAMEWORKS TO ENHANCE PRODUCTIVITY AND BRIDGE GAPS IN CURRENT ENGINEERING PROCESSES

By

Elijah M. Sierra

A THESIS

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Mechanical Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2025

This thesis has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Mechanical Engineering.

Department of Mechanical and Aerospace Engineering

Thesis Advisor:	*Paulus van Susante*
Committee Member:	*Andrew Oliva*
Committee Member:	*Chad Walber*
Department Chair:	*Jason Blough*

Table of Contents

List of Figures

List of Tables

Abstract

This thesis aims to tackle some of the inefficiencies with the current methods of engineering. The issues are specifically the inefficient iteration processes between preliminary design and detail level design, a lack of centralized resources, and poor project organization and management. The thesis provides a case study on the current tools that solve this issue and then presents a new proof of concept framework for an engineering tool. This tool addresses these issues by unifying calculators, databases, external tools, and project management systems under one modular platform.

The tool consists of one central GUI with multiple independent modules such as a beam calculator, CAM designer and material database that are displayed within the main GUI which enhances the modules with project management and metadata systems. Its architecture is designed for modularity and scalability, enabling future expansions of new tools.

Verification tests confirm that the system runs as intended and can create preliminary design CAD models using parametric knowledge-based engineering which can then be analyzed in FEA. This helps reduce time for initial iterations by eliminating initial guess work. The tool was then extensively tested in real engineering scenarios where it proved essential to the success of the projects. Expediting timelines and completing all requirements for the project proving that the tool works and is needed in engineering workflows.

1 Introduction

The engineering process has evolved significantly over the years and with the rapid advancement of modern computers the process has become more effective and efficient. New tools have been developed such as Finite Element Analysis (FEA) and Computational Fluid Dynamics (CFD) to help aid the process of designing. These tools are very resource and time-consuming, meaning that if a proposed design is too weak the designer will not know for some time. This requires the user to go back and forth between design and analysis. This iterative process takes considerable time and money for an engineer, and modern tools are being developed to shorten this iteration process. This paper will review what tools already exist that solve this, then propose a proof-of-concept framework for a tool that combines some features of existing tools and then adds to them.

1.1 Background

Before an attempt can be made to make a process more efficient it is important to understand it as much as possible. The engineering process is extremely complicated and there are many structures that attempt to standardize the process. The most common one and the one that will be used as reference in this paper is the Vee engineering diagram developed by Kevin Forsberg and Harold Mooz [1]. Figure 1 below shows a modernized version of the diagram.

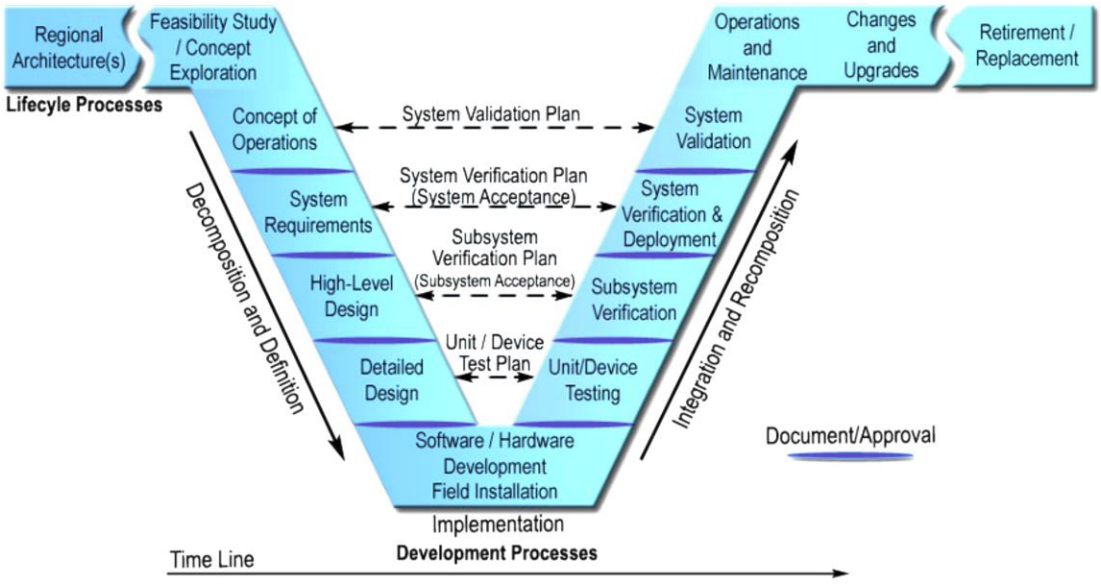


Figure 1 Engineering Vee diagram [1], [2]

The basic process of the engineering Vee diagram is to start with research and study of a field and topic of interest. During this research phase a problem must be identified before deciding on system requirements. After this point a high-level preliminary design can be created. This preliminary design will go back and forth between CAD and analysis software like FEA or CFD. This is where designs can spend a lot of time going through multiple iterations before reaching a final acceptable state. At this point a detailed design can be created which can then be machined and integrated with electrical, software or any other systems required for the design. The design will then be tested, verified, and validated where the term verification means that a problem was solved correctly and validation ensures that the right problem was solved [3].

1.2 Problem

Engineering design suffers from inefficiencies that stem not from lack of knowledge or capability, but from how knowledge and tools are used. There are three key issues: excessive iteration, information inaccessibility, and project disorganization. These problems slow design process down and can even kill a project.

1.2.1 Inefficient Design Iterations

The engineering V diagram depends entirely on by design iteration which is good for catching issues and creating a better design but is inefficient, time consuming, resource intensive and can hurt the project and company. Iteration should not be eliminated but should be balanced with time. Since time is a critical resource it is necessary to optimize the initial stages of the design process to decrease the number of iterations. The problem is that all designs begin with an initial guess on first design iteration and then is modified multiple times based on results from FEA or CFD. Or the engineer will spend extra time in the beginning calculating the optimal shape by hand before creating a first preliminary design. This may decrease total number of iterations, but the time saved from the eliminated iteration time is wasted on the time spent doing hand calculations. One company found that a standard design time for one turbine blade was found to take days to complete [4]. This was all due to iterations between the aerospace engineers and the mechanical engineers using CFD and FEA software respectively. This same company found that if they eliminated iterations, they could decrease design time to minutes and save up to $3.7 million per

engine project [4]. The tradeoff with eliminating iterations is quality so a balance must be struck between quality and time.

A case study conducted at an engineering firm found that time and money was wasted on design time. Even designs as simple as a beam took either multiple iterations to find an optimal beam shape or an over engineered solution be produced [5]. The issue that caused inefficient design iteration was that a user guessed an initial beam size based on loads, constraints and free body diagrams then conducted an FEA analysis on the beam. After the first analysis the engineer may either decide to pick a larger beam if the initial beam were too weak or pick a smaller beam if the initial beam was too strong and the engineer suspected they could save weight or money with a smaller beam. This cycle is repeated until an optimal design is found. This can lead to the engineer spending excess time on a design or choosing a less optimal design that is either too weak or too strong.

1.2.2 Resources Not Centralized

During the design process, engineers will have to solve many problems and to do this successfully they will need access to ample amounts of information and resources. There are plenty of tools and textbooks that when combined hold all the information required for an engineer to solve anything. The issue is that none of this information is easily accessible to engineers because the information is not centralized in a practical way. There are of course tools such as google or AI that can find any information on the internet. However, a recent case study found that engineers are dissatisfied with information retrieval systems like key word matching or AI based search tools [6]. They stated that these tools are useful only when it is known what to look for and that they would prefer an engineer-oriented resource list containing commonly used resources such as textbooks or calculators [6]. These useful tools are not as popular on the internet so when searched for they will not appear as easily. This is a clear issue of resource decentralization where engineers would benefit from a tool that had all important resources, tools and literature centralized. Without it engineers may make design decisions that are less optimal without realizing that a resource was available to them all along.

1.2.3 Project Disorganization

Project organization systems are implemented to ensure information is easily accessible and usable. But this is not a top priority for engineers, meaning that a lot of projects tend to be disorganized. This can lead to misplaced and/or lost information. Therefore, it is essential that projects are organized. But it does take effort to ensure a proper project setup. This includes things like folder directory set up for storing information, CAD file organization and notes to prevent knowledge loss. "When documents are scattered and disorganized, it impedes the decision-making process… This not only slows down the project but also increases the risk of making uninformed choices that could jeopardize the project's success" [7]. It is vital that an engineering tool supports project organization and does not impede existing organization. It is also important that a standard be created within a team so that if an engineer in one project needs information from another project, they can find what they need easily since file structure and project organization are standardized.

1.3 Aims and Objectives

The purpose of the work presented in this paper is not to solve all these problems with one solution but to create a framework that is able to support many solutions to the problems stated above. The objectives of this work are to

1. Conduct a thorough literature review of resources, tools and calculators that are already available that are solutions to the problems above,
2. Create a framework engineering tool proof of concept that is not complete but proves its importance and is able to be expanded in the future,
3. Verify that this tool outputs correct values and,
4. Validate the tool by demonstrating the potential on engineering projects.

1.4 Timeline

This section outlines the entire timeline of this thesis to provide context on the scope and pacing. And since this is a tool designed to enhance productivity and expedite timelines this section also shows just how much work was able to be completed with aid from the tool created. The entire thesis was completed within approximately 10 months with work beginning in early-January 2025 and concluding with the defense in mid-November 2025. Figure 2 below shows an overview of all the projects pertaining to the master's thesis. Each item on the timeline represents a full

engineering project that went through the standard engineering V diagram. It is important to note that the Lunar Terrain Vehicle (LTV) project started a couple months before the start of the master's thesis. It is also important to note that this timeline does not include class work, lab work, teacher assisting or other non-project-based work required for the thesis.

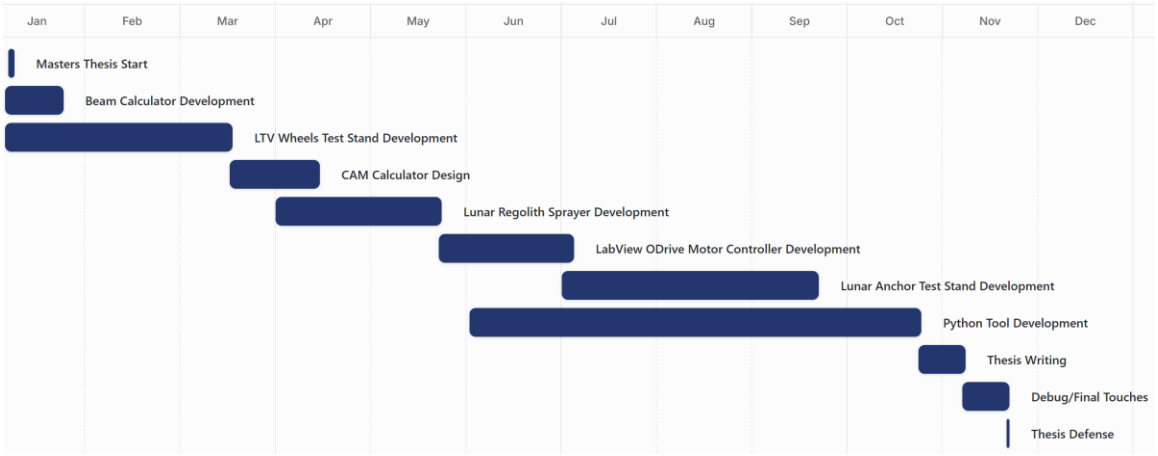


Figure 2 master's thesis timeline

1.5 Motivation

The motivation for this project originated from multiple firsthand experiences in both academic research and practical engineering settings. In the field of research, despite advanced research being conducted, there were many inefficiencies caused by disorganization and lack of easy access to tools. There was also a significant lack of proper analysis conducted on the products produced. Parts were either designed based on guesses or the engineers spent an excessive amount of time iterating between CAD and analysis. In comparison many large corporations have managed to combine the knowledge of mechanical engineers with the knowledge of software and IT developers to create tools that increase efficiency. The tools organize all information and tools that any engineer creates. They are also designed to be expandable so any new tool that any engineer develops can easily be integrated into the architecture. However, none of these tools are publicly accessible so this project exists as a vision for a more connected, accessible, and efficient engineering environment.

2 Literature Review

The invention and idea of an engineering tool to aid the process of designing a product is not a novel idea and many engineering tools have already been created. However, these tools are not perfect, and many gaps exist for these tools such as availability, design automation, expandability and more. To find and pinpoint the exact gaps in the current field of engineering tools an extensive literature review was conducted. Over 200 papers and tools were screened down to approximately 60 and then ultimately 30 were chosen as the best engineering tools currently in the field. This section will outline these 30 tools and identify any gaps.

2.1 Turbine Blade Automation

Turbine blade design is an exceedingly difficult task that requires many iterative steps to ensure perfection. Andrew Lowe proposes an automation tool to reduce the amount of time spent on turbine blade design [4]. He wrote a paper called A Case Study in CAD Design Automation where he not only looked at how to automate the design process of turbine blades but also the impact it may have on a company that might benefit from a design automation tool. Andrew Lowe also included an extensive case study on current tools in industry and how he could leverage them for his tool [4].

Lowe started his project for a company that designs its turbine blades manually through a process that requires a lot of back and forth between the aerospace and mechanical engineers. The Aerospace engineers would propose certain designs based on computational fluid dynamic (CFD) analyzers to optimize thrust and the mechanical engineers would change the design to optimize part strength based on Finite Element Analysis (FEA). This process, amongst other things relies heavily on luck for how long the design process will take and was found to take a couple of hours at best but would normally take a couple of days [4]. This process needs to happen up to 20 times for just one engine design thus this was identified as the main target for optimization and automation [4].

Lowe's case study found that knowledge-based engineering (KBE) had significant potential to decrease design times by a substantial amount. KBE would increase the total amount of knowledge easily available to engineers and would then be used to automate CAD design based on knowledge retained from previous projects or other

engineers. KBE can even go as far as to automatically create multiple CAD models and compare estimated costs and manufacturing [4]. Since Andrew's company primarily uses Siemens NX for CAD and then ANSYS for analysis he proposed a tool integrated with the APIs of NX and ANSYS.

The tool designed by Andrew was a special NX ANSYS Application Programming Interface (API) integration tool that would be able to reference previous designs and use KBE to create parametric designs of turbine blades automatically. It would do this by first calculating the optimal 2D shape for a turbine at different radiuses. It would then discretize the shape into a point cloud. Multiple 2D point cloud files would then be stacked on top of each other linking point to point creating a 3D mesh. This final shape, however, would not be perfect and would still require some iteration through ANSYS CFD and FEA software. These software packages require complex meshing of the parts which can be time consuming but because the turbine's 3D shape is already meshed this transition was streamlined [4].

The benefits of this tool were quantified by taking surveys of multiple engineers through the typical engineering process of a turbine blade. Andrew found that a typical engineer would take "5 to 8 hours to create a solid model from Computational Fluid Dynamics point cloud" [4]. With the tool it took engineers 4 to 6 minutes to design the same turbine blade [4]. Based on this data the company estimated that the use of the tool could "save approximately three quarters of a million dollars in direct costs alone on a single engine" [4]. Another benefit from this tool includes easier and quicker editing of a turbine blade. If an FEA or CFD solution deemed the design to be inadequate it would be quicker to redesign a new blade. Also, since it only took 4 to 6 minutes to design a turbine blade more time could be spent perfecting blade design leading to higher performing engines. Another great benefit of this tool is that it standardized turbine design etiquette meaning that designers in the same company could better understand and edit each other's deigns saving even more time and money.

2.1.1 Tool Review

This tool overall is a great show of how much time and money can be saved from the use of an automative engineering tool. The paper does a great job in showing how much effort goes into designing one specific part and then how the tool can make it much easier. However, the issue with this tool is that it is proprietary to a specific

company, meaning it is not accessible by the public. Also, this tool requires expensive licenses from NX and ANSYS to use since it works with the software's APIs.

2.2 TRAnsmission DESigner (TRADES)

TRADES is a prototype tool that focuses on transmission design automation. It is a tool that can produce preliminary designs of transmissions based on user input [8]. The paper "A genetic algorithm based preliminary design system" writing by D T Pham, BE, PhD and Y Yang [8] explains the inspiration of the project and how TRADES works. They also include a brief literature review into tools and calculators that already exist but because they are obsolete, they will not be included in this paper.

The paper states that there are essentially 4 steps in the engineering process: preliminary or conceptual design, detailed design, evaluation, and redesign [8]. This is a quick dirty summary of the standard engineering V diagram. The point of TRADES is to focus on the first step in this process. The first step in using TRADES is to input the motion going into the transmission and then the desired output(s). Secondly, geometric constraints are added to such as where the input and outputs shafts are located as well as how large the transmission box can be[8]. TRADES then uses a genetic algorithm system to iterate through many designs combining different types of gears and pulleys, throwing out designs that violate requirements until an acceptable outcome is reached [8].

2.2.1 Tool Review

TRADES achieves success in creating preliminary 2D designs for transmissions. The entire software package is all encompassing and contains databases of different types of gears, pulleys, and other motion mechanics. The tool at the time of its creation was one of the first of its kind to use genetic algorithms and was the first transmission automation tool [8].

Although TRADES represents a great step forward in terms of early engineering tool development, it can only create 2D transmission designs aiding the engineering process only slightly. However, the use of databases to create designs is an interesting and innovative part of the tool. This system follows a couple of the ideals for a knowledge-based engineering system but falls short by not including straightforward ways to add new gears or information to the databases.

2.3 Virtual Product Modeling

This tool takes an interesting approach to engineering automation. Instead of using typical CAD platforms or other engineering software packages it instead uses UNITY, which is a game engine used to develop 3D models for video games. But the work presented by Guolong Zhong's PhD paper: A novel virtual product modelling framework for design automation in a knowledge-based engineering environment, gives great insight into how to design a useful engineering automation tool [3].

Zhong starts his PhD with a very extensive literature review and since his paper was published 3 years before this paper there are many still important and relevant lessons learned. Zhong spends some time talking about Computer Aided Engineering (CAE) and how all steps of the engineering process can be supported by a computer. "CAE is a broad concept that means using computers in all phases of engineering design work "[9]. This includes CAD, FEA, CFD, CAM and anything else where a computer is used to support engineering work. Zhong's literature review taught him more about these different types of computer aiding technology and how they interact with each other. He found that the most time-consuming part of designing a new product was the embodiment design and detail design [3].

Zhong then does a deep dive into knowledge-based engineering (KBE) explaining how it is the backbone of any engineering automation tool. He describes how KBE is a network of ideas and information that are stored in a database where tools can later access to create new designs [3]. He found KBE to be a very intriguing tool that had the potential to combine all engineering principles in an easy to access database where a tool could then reference it and make new designs based on the knowledge in the database [9]. The only issue was how to capture and retain this information. This is where knowledge-capture methodology (KCM) features come into play. The basic idea is that not only is information stored on the KBE database but all projects that use the KBE system document what information they used and when. This is important because an automated engineering tool that uses and stores information in a massive KBE database runs the risk of turning into a black box. This is bad because if the tool is used incorrectly to create a defective product the information used to generate that bad product is lost [9]. So, a KCM system must be implemented to help alleviate some of the symptoms of black box syndrome.

Model Based Engineering (MBE) and Model Based System Engineering (MBSE) are both important for generating an engineering tool. Zhong explains that for an automated engineering tool to work certain standards must be defined to keep the tool compatible with as many platforms as possible [9]. The most important standard to consider is what kind of files will be generated by the tool. Zhong recognizes that STEP is a very highly versatile file supported by every CAD platform. However, STEP files lack information regarding how the model was created, as well as any editing capabilities. Zhong recognizes that to use STEP files as a standard output of the engineering tool extensive knowledge retention measure would be required [3].

Design Engineering Assessment (DEA) is a final check process that a design goes through to ensure that all requirements have been met and that everything is verified and validated [9]. Essentially, DEA is a process to assess whether the design solves the right problem correctly. Zhong recognizes the importance of DEA and to ensure that the designs created by the tool have the highest chances of succeeding the tool must output something that has initial design intent, high compatibility, and the optimal solution given user inputs [9]. In order for an engineering tool to have a chance of creating viable designs it must adhere to DEA principles. This means that any and all information regarding design must be standardized and organized.

Zhong then spent the rest of his literature looking for existing tools that took principles from knowledge-based engineering and applied them to automating certain steps in the engineering process. It was quickly discovered that parametric modeling to alter and change a preexisting design had been done before but had not taken advantage of KBE techniques. These included platforms such as NX, SolidWorks, and many other software systems. Due to the limitations of these CAD software platforms Zhong made the decision to use Unity as the engine for creating his engineering tool [9].

Zhong identified a few research gaps based on his extensive literature review. The research gaps focus on knowledge retention and the prevention of black box syndrome [9]. Zhong's goal is to develop a tool that has "methods of capturing, reusing, and exchanging design knowledge for product modelling and to develop a knowledge-based product modelling environment for applying these methods" [3]. Zhong lists a couple more objectives revolving around finding a proper engine to develop the tool and then verifying and validating the tool.

The final product created by Zhong is an all-inclusive engineering tool that utilizes knowledge-based engineering combined with parametric modeling and knowledge retention. It is a very impressive product that allows users to input CAD models, and the tool will automatically fill out a knowledge database file based on the model. The tool will then allow the user to change certain parameters of the model and then it will compare the users model to existing engineering standards and warn users if they violate any standards [9]. The tool also allows users to easily input any information into the KBE database including materials, standards, requirements, functions, design intent, relationships, parameters, constraints, and dimensions.

The features of the tool are listed below [3]:

- Generative Representation
    - The tool can generate a parametric model based on the model uploaded by the user [3].
- Knowledge capture
    - The tool can capture all information regarding the uploaded part and create a knowledge file for the database [9].
- Product geometry and knowledge visualization
    - The tool can display the part and its information in an intuitive graphic user interface [9].
- Product relationship representation
    - The tool can keep track of and display relationships between parts and between assemblies [9].
- Knowledge reasoning and reuse
    - The tool can change the dimensions of the part based on information stored in the KBE database [9].
- Correctness of changes
    - The tool can reject any changes to a part that violates a standard, constraint or requirement set in the KBE database [9].
- Data exchanges of geometry
    - The tool can produce standard CAD file types in the form of STEP models [9].
- Data exchange
    - The tool is able to generate new knowledge files based on modified design [9].

The GUI shown below is the simple yet intuitive design of Zhong's engineering tool. In the box labeled as (a) there is a simple display box of the part where the user can see the part that they are modifying. Information regarding the part can be seen and edited in the box labeled as (b). This is where the knowledge file is created and edited for all parts that go through the tool. In the box labeled (c) is the parametric box where the user can change certain dimensions of the part. Finally, the box labeled as (d) is where any messages powered by the KBE database will be displayed. This includes all warnings of standards or constraints violations [9].

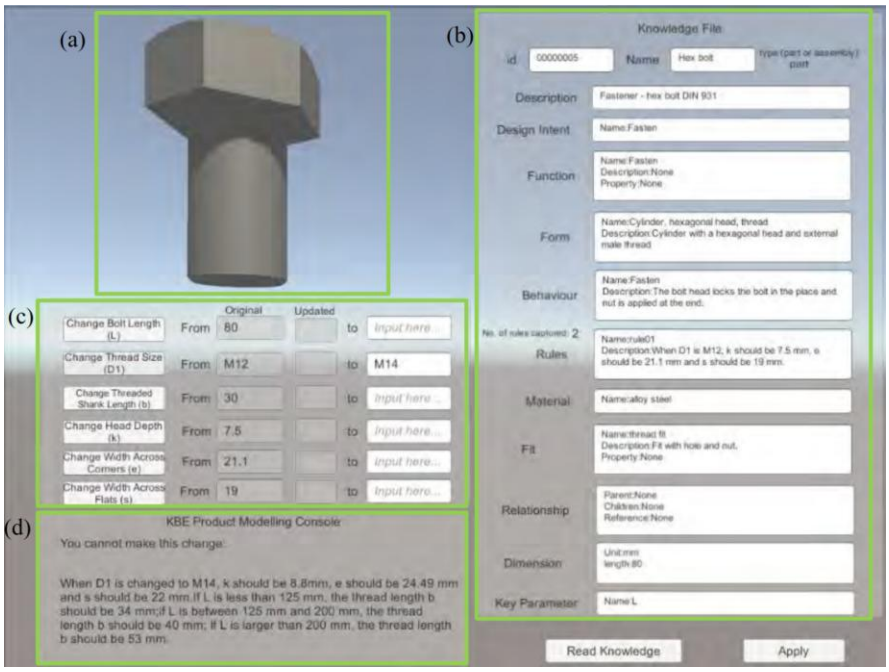


Figure 3 GUI of Zhong's virtual product modeling engineering tool [9]

2.3.1 Tool Review

The tool created by Zhong is by far one of the most advanced tools to tackle the beast, which is an all-in-one engineering tool. He started out strong with a very in-depth literature review finding many existing tools and learning about how different tools leverage different ideas such as KBE or parametric design. His tool does a great job in displaying CAD models and tracking information but the one thing that this tool does that is super important is the integration of a knowledge based engineering database that compares the design on screen to a multitude of standards, requirements and constraints and if the design violates one of these it will alert the user. All the user needs to do is upload a CAD model and then they are able to change it to match a

different standard or constraint. Now the first issue with this tool is that Zhong decided to use UNITY as the main engine for creating this tool. He pared this with C#, which is fine, but UNITY is a video game engine software meant for low fidelity nice looking graphics. It is not great for CAD modeling. The other issue with this tool is that it does not aid the engineer through calculations or preliminary design. The tool requires that the user uploads a preliminary design which can then be altered and changed to match standards and requirements in a database. This means that the tool cannot create a CAD model from scratch based on user inputs but instead requires a reference CAD model to be able to apply KBE ideals to.

2.4 Online Engineering Tools

On the internet there are many different types of tools that engineers can use to help aid their designs. These tools can range anywhere from quick educational calculators that help students learn how certain equations work to full online engineering packages that are able to automatically solve beam and bolt calculations. A list of the tools pertaining to this research is shown below.

Online Tools:

- ClearCalcs [10] https://calcs.com/
- SkyCiv [11] https://skyciv.com/
- WebStructural [12] https://webstructural.com/
- Engineering Toolbox [13] https://www.engineeringtoolbox.com/
- STRIAN [14] https://structural-analyser.com/
- MechaniCalc [15] https://mechanicalc.com/calculators/
- CalcForge [16] https://calcforge.com/
- BeamGuru [17] https://beamguru.com/
- StruCalc [18] https://strucalc.com/

2.4.1 ClearCalcs Review

ClearCalcs is an online engineering tool that has some free to use features, but much of the tool is behind a substantial pay wall that at the cheapest option will cost $790 per year [10]. The primary users of ClearCalcs are civil engineers who create bridges, structures, buildings, roads and anything related to infrastructure. So, the main features of this tool are free body analysis and beam calculations [10]. These beam

calculations include steel AISC beams, concrete with rebar beams, wood beams and more. All features on the ClearCalcs website adhere to all applicable standards and once a project is completed a professional document is automatically created with design choices and results [10]. Of course, access to these features depends on what subscription plan is selected. To use all of the features a user would have to pay $1,490 per [10].

ClearCalcs is a great tool, and many engineers use it, but it does have some downfalls. The first and largest issue is its substantial price tag which is a yearly subscription requiring users to pay $1,490 per year [10]. Another issue with the tool is that it is all online relying on external services and internet to view and work on projects. This means that all projects created with ClearCalcs have privacy concerns which could violate ITAR restrictions or other non-disclosure agreements [19]. This also means that if web services go down access to the project and all its information is lost until web services come back up. Unfortunately, web services outages are not uncommon as Amazon Web Services (AWS) recently experienced a full system outage on October 20th, 2025, causing many basic online features to break and it is not the first or last time this will happen [20], [21]. This tool also does not contain knowledge-based engineering as it does not create CAD models let alone KBE parametric models. It will only recommend which beams or structures to use based on user input and standards [10]. Finally, it is unclear what kind of knowledge retention system this tool has. This tool automates a lot of important engineering steps but does not include knowledge retention, increasing the likelihood of black box syndrome.

2.4.2 SkyCiv Review

SkyCiv is like ClearCalcs but is a little more advanced and targets the mid to late stages of design [11]. The tool's main feature that sets it apart from other online tools is its intuitive FEA based beam and structure solver. This tool allows users to draw 1D beam element lines to create 3D frames. These beam elements can then be assigned a beam type and size. The FEA solver will then solve stress on each beam in the frame based on loads and constraints [11]. Another feature of SkyCiv is that it autogenerates a full document with the calculations and equations used to get the results that the tool created. This is great for preventing block box syndrome and is a great inspiration for any tool to be created in the future.

There are some issues with SkyCiv that are identical to ClearCalcs. The first issue is that it is extremely expensive. Prices for this tool range from $79 per month ($948 per year) to $499 per month ($5,988 per year) [11]. This pricing is competitive compared to other FEA software like FEMAP Nastran which on the low end is $10,207.44 per year [22]. But this is not a tool for preliminary design and if professional engineering designs are required, tools like ANSYS, FEMAP Nastran and SolidWorks should be used, which does not leave much room for this tool. This is because SkyCiv only analyses beams selected by the user and does not recommend beams based on inputs for preliminary designs. SkyCiv therefore is positioned as an online alternative to traditional FEA software; however, its modeling capabilities and solver robustness are not comparable to established tools such as FEMAP Nastran or ANSYS, which limits its suitability for professional-grade structural analysis. Not to mention that everything is in the cloud which means that the user relies on the web services that are unsafe for privacy [19], and could become unavailable at any moment [20], [21].

2.4.3 WebStructural Review

WebStructural is a much smaller tool compared to SkyCiv where it focuses on basic stress calculations for preliminary designs. Their business angle is that although "modern structural analysis software is amazing. Sometimes you just need to design a beam" [12]. Their tool focuses on calculating stress in a beam based on constraints, loads, materials, and beam type. Their tool also includes calculators for concrete footing and fasteners. The tool aims to aid the process in finding a preliminary design for a beam, fastener, or concrete foot [12].

As great as this tool is, it does contain some issues. The first issue is that it is not free. It does not cost much but it is $19 per month ($228 per year). It does contain a free to use version, but this version is severely limited and is essentially a way for the company to show off their tools without giving anything away for free. Another issue with this project is that it only calculates one thing at a time so if a user wanted to figure out what beam to use, they would have to manually cycle through all of the beams until they found one that worked. Another major issue with this tool is that it does not contain any documentation for outputs, so it has severe potential to suffer from black box syndrome. Finally, just like all online calculators this tool also suffers from the risk of privacy issues [19] and data loss or unavailability [20], [21].

2.4.4 Engineering Toolbox Review

The engineering toolbox is the most popular and most widely used tool on this list. It is no surprise either because it is a free to use tool that has extensive resources ranging from unit conversions all the way to hydraulics [13]. The engineering toolbox is an online website used for quick calculations of very early-stage design decisions. The list of features offered on the engineering toolbox is way too long to include here but almost all algebraic equations for engineering have been organized in an easy to absorb fashion. If an engineer needs to solve a certain equation, they can search for which calculator they might need and then after putting in their inputs they get an answer [13]. A remarkable thing about the engineering toolbox is that for every calculator they show the equation and a brief explanation about how it works. This prevents black box syndrome almost entirely.

Although the engineering toolbox is a great tool and is free to use there are still some issues surrounding it. The first problem is that it has a horrible user interface that is not organized very well, and it is difficult to find anything which is bad for a toolbox containing an obscene number of calculators. The user interface is also completely drowned in ads that are super distracting that make the tool less pleasant to use. Another thing to point out is that this tool can only assist the very beginning stages of an engineering project. Meaning it can only aid an engineer in preliminary hand calculations for napkin sketches. This makes the tool great for educational purposes but not great for professional engineers who need a more advanced too to make their work easier. For this the tool would require automation systems for creating preliminary CAD models and storing information in easy to access databases. Finally, just like all online calculators this tool also suffers from the risk of privacy issues [19] and data loss or unavailability [20], [21].

2.4.5 STRIAN

STRIAN is a simple yet intuitive online free body diagram solver. It is a very basic but intuitive calculator that allows the user to place nodes on a grid and then connect nodes with beams. Each node can then be given either a load or a constraint. Beams can also be given either point loads or distributed loads. The calculator will then solve the reaction forces on each node and each beam [14].

STRIAN is a great simple tool that does one thing very well. Considering that free body diagrams have infinite different configurations of forces, constraints, and shapes this

is still impressive even though it is small. However, this online calculator still has potential to fall victim to privacy issues [19] and loss of data due to web service blackouts [20], [21]. Also, it should be mentioned that although this tool was designed to only accomplish one task an engineer is less likely to use it because it is not centralized to the tools that they use on a regular basis. This tool is great but should be created in a way that it can be integrated into a larger tool that centralizes multiple tools in one easy to access place for an engineer.

2.4.6 MechaniCalc Review

MechaniCalc is mostly used as an educational tool containing resources on how to study and pass the PE exam. The website can be treated more like a textbook, but it does have calculators for automating certain portions of the design process [15]. The list of calculators is extensive, but the most notable calculators are beam analysis, 2D FEA, bolt analysis, and stress concentrations. These calculators are for more educational purposes and thus have extensive validation and verification showing every equation and step in the process of calculating everything which is great for preventing black box syndrome.

MechaniCalc is a great tool but since it is oriented towards educating engineers and preparing them for a PE exam it is not ideal for automating tasks for engineers. The databases are not extensive and only contain enough information to give good examples for educational purposes. Although the calculator allows for custom information to be uploaded for specific beams, or bolts or materials it is not ideal for an engineer to have to look for that information before uploading it. Also, most of the features are hidden behind a pay wall costing $99.99 per year for individuals and 299.99 per year for classrooms [15]. Almost none of the calculators are free to use and the ones that are only in a demonstration mode showing off features without giving anything away. Finally, just like anything else on the internet MechaniCalc suffers from the risks of privacy infringements [19] and data loss due to web shutdowns and black outs [20], [21]. So, although this might be a great tool for educational purposes it is not great for engineering.

2.4.7 CalcForge Review

CalcForge is a useful tool for preliminary design for civil engineering containing multiple free body diagrams and beam calculators. There are many other calculators as well to help with designing foundations and tunnels [16]. Additionally, there are databases that allow users to calculate, compare, and contrast different beams based on force inputs. This tool allows users to see all beam types and whether or not they would work for their specific load cases. There are also databases of previous civil engineering projects where users can compare their project to previous projects to estimate total cost [16]. This tool is a useful resource for civil and mechanical engineers where the user can input a free body diagram, calculate reaction forces and then input forces into a beam calculator to determine preliminary beam design [16].

Although this tool automates a lot of the initial preliminary designs for engineers there are still some drawbacks. The first drawback is that it is not free coming in at about $96 per year [16]. Another issue is that there is no data tracking or logging meaning that all inputs that are used to create results are not logged meaning that if the tool is used incorrectly to specify a beam incorrectly it will be difficult to figure out where and when the mistake was made. This means that the tool suffers greatly from black box syndrome. Another issue is that their databases are incomplete and do not allow for custom materials or beams to be uploaded. Their beam calculator is missing T beams, L beams, and C beams and there is no option to choose material or add 6 degrees of force and loading. Finally, just like every other online resource CalcForge risks information leaks due to privacy issues [19] as well as information loss due to blackouts [20], [21]. Overall, this is a great tool for preliminary design that needs some upgrades but is a great inspiration for future tools.

2.4.8 Beam Guru

Beam guru is a beam calculator where a user can input simple 1D or 2D trusses select forces and constraints then calculate the reaction forces. What sets this tool apart from other similar tools that calculate beam forces is that it auto generates a report outlining every single calculation that went into creating the output [17]. This almost eliminates the probability of black box syndrome and is an excellent feature that all tools should strive to incorporate.

But just like all great tools there are downsides. This tool is very basic and only contains a beam calculator that only outputs forces and documentation. The documentation is unlike any other calculator but is only for basic 1D and 2D beam free body diagrams. Another issue with this tool is that most of its features are behind a pay wall costing $249 per year or $2 per calculation [17]. Also, since this calculator is so simple it does not contain advanced features like databases for beams or materials which means it cannot help specifying beam size and shape. Finally, since it is an online tool, it risks leaking data due to privacy issues [19] and/or losing data due to web blackouts or shutdowns [20], [21]. Overall, this tool's greatest strength is auto generation of incredibly detailed calculation documentation, but its scope is too small to be useful to most engineers.

2.4.9 StruCalc Review

StruCalc is an extremely expensive and professional calculator to aid civil engineers and non-civil engineers in designing structures [18]. This tool is not actually an online calculator but requires internet to communicate with StrCalc servers to verify license and access databases. The main feature of this tool is an all-encompassing beam calculator that takes user determined beam type, material, free body diagrams, forces and constraints then calculates the reaction forces, stresses, buckling, and deflection [18]. The tool also generates extensive documentation that is standardized and complete with all input and output details [18]. The beam calculator does every single beam type and all materials including wood, plastic, aluminum, and steel [18]. The rest of the tool is structural design for houses including foundation, walls, roofs, and decks which are not as important for mechanical engineers who are not designing a house. But they are key features to point out, nonetheless. The tool also contains numerous training and walkthroughs teaching new engineers or students how to use the tool which is great for knowledge retention. Lastly, it is clear that extensive work went into creating a proper user interface for this tool. The figure below shows the beam calculator user interface which is a very pleasant design and inspirational for future tool designs [18].

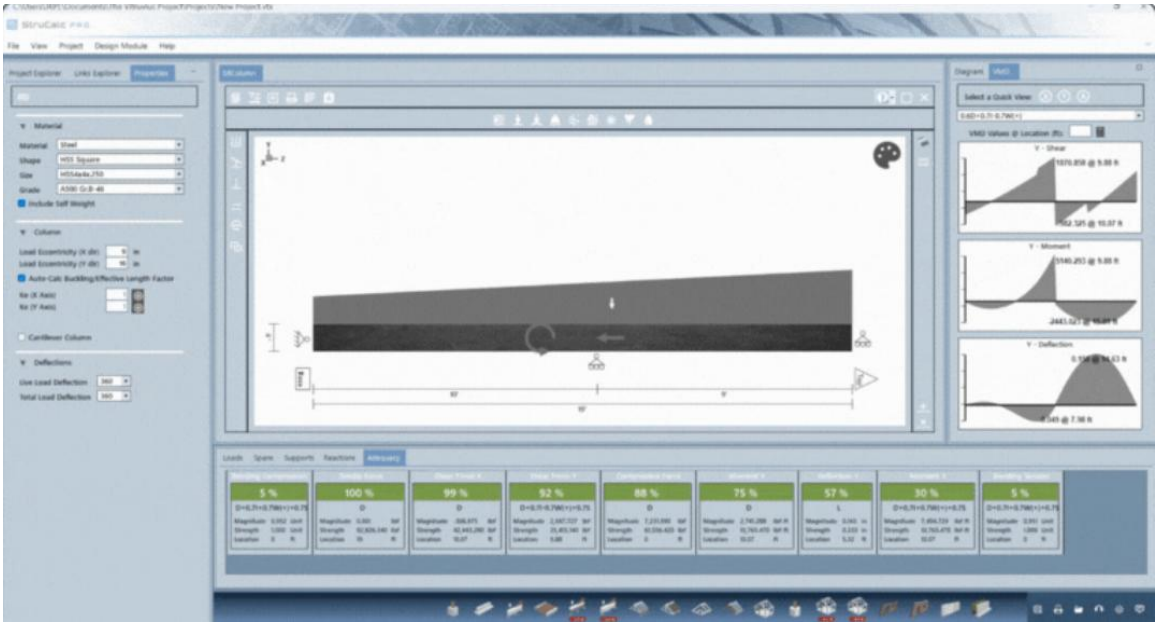


Figure 4 StruCalc user interface for their beam calculator [18]

StruCalc is an exceptionally large and professional tool that accomplishes a lot but there are a couple of shortcomings that can be improved upon. The first and most notable issue is the price tag sitting at ~$ 1,000 per year [18]. Also, this tool was designed for civil engineers which means that it has a lot of useful features for civil engineers but are missing some key aspects that would be nice for mechanical engineers. These features may include bolt analysis more dynamic features like CAM designers or motor controllers. The calculator also does not help reduce design iteration because it evaluates one design at a time that requires the user to set up an analysis similar to FEA, albeit much faster. But this means that if the design fails the analysis the user must reanalyze new beams until one passes. This process can be expedited by just running a quick analysis on all beams all at once and then finding the correct beam and then running a more detailed analysis on that one. Finally, even though this is an application software and not an online web tool it still requires internet services to access databases run by StruCalc which means that it still runs into data breaches [19] and information loss due to blackouts and downtime [20], [21].

2.5 Summary

This thesis is not going to be an entirely novel tool because there are many tools that already accomplish the aims and objectives of this project. However, that does not mean that there are not any gaps in the current field of engineering tools or that it is not worth learning more about how these tools work.

2.5.1 Information Learned

There are many tools available for engineers to use, and they each contain features that are remarkably interesting and are worth learning about and implementing into a new engineering tool. To address the repetitive iterations a system called Knowledge Based Engineering or KBE is used in many tools. This is a massive system most used in high production parts where slight changes of similar parts are required. This includes things like nuts and bolts where the overall shape is the same but vary slightly in thread or length. KBE is a database driven Computer Aided Design system that takes a parametric CAD model and makes slight changes based on user requirements. This system also cross references databases that contain things such as standards, requirements, or design intent. This is a tool used to automate CAD designs without sacrificing quality or safety. For example, if a user requires a ¼-20 bolt that is 0.45” long the KBE system will take the existing ¼-20 bolt CAD and modify it to match the user’s needs. However, it will recognize that a bolt length of 0.45” does not match any standards so it will either send a warning to the user saying model violates a standard or it will recommend 0.5” as an alternative bolt length. KBE is a super useful tool but in most cases it requires a preexisting CAD model to modify. It would be interesting if a modified KBE tool could be created that could create a CAD model from scratch utilizing only user inputs.

Another excellent feature that most tools have is a material database. This is a straightforward system where the tool would be able to store mass amounts of information about many different types of materials in a compressed binary file that can then be displayed for the user to use. This material database can also be paired with the Knowledge Based Engineering (KBE) system so that when a CAD model is altered or created a material can be assigned that best fits the user requirements.

An additional feature that some tools had was Preliminary Multidisciplinary Design Optimization or PMDO which is a feature that combines all engineering disciplines at

the beginning of the design process. This means that at the beginning of the design process all types of engineers have equal say in the initial design. So, if a turbine blade is to be produced both mechanical and aerospace engineers would design the initial blade at the same time. This is to decrease the back and forth between the two engineers later in the design process. This can be paired with knowledge-based engineering (KBE) where the knowledge used to create a design will take from standards in all fields pertaining to the design.

An Interesting Issue that most tools ran into was black box syndrome and difficulties with knowledge retention. The problem was that these tools were replacing a lot of the intermediate thinking between certain steps of the design process. Outputs produced by the tools could be used without knowing what information went in to creating them which could create issues. One interesting idea that very few tools incorporated was the use of mandatory metadata where the information used for the inputs for the tool was saved alongside the outputs. This allowed engineers to better understand what requirements and standards were used to create the part. Additionally, the retention of knowledge allows unaffiliated engineers to understand what the thought process was that went into the tool to create the part. This system, however, only reduces the chances of black box syndrome and does not eliminate it completely because any tool that helps automate any process will inherently be a black box. This metadata knowledge retention system does help with black box syndrome by providing as much information as possible but if a user refuses to view such information, then that user will suffer from black box syndrome.

Documentation automation is another tool that aims to prevent black box syndrome by not only tracking metadata but also presenting it in a way that displays all information important to the project. This feature was surprisingly in quite a few tools where after a calculation is completed, a document displaying all information about the output is included with the main output. Popular things included in this document are:

1. Estimate total cost of part (Cost to machine, labor costs, off the shelf costs, etc.)
2. Total tome spent on the project
3. Requirements for design
4. Equations used in calculations
5. All user-made notes

The final main feature learned from the literature review was project management features such as file directory automation. This system was not caught in the literature review and was only ever observed at SpaceX but is sure to exist elsewhere. The idea is that when a new project is created a properly named file directory and folder system are automatically created. This has two main purposes: 1) it saves the user several additional seconds of manually creating folders and 2) it standardizes folders layouts. The extra seconds of saved time might not sound like a lot, but it quickly adds up, and it also removes the thought process of setting up folders allowing the user to quickly get to work easier. Standardized folders also might not sound important, but it allows for easy and constant organization between projects meaning that if information is desired from another project, it is easier to find because the folder structure is the same.

Overall, there are many tools that all contain a different combination of the features listed above as well as other features that are listed in the table below. The table below contains all tools captured by the literature review. Each tool is compared to a multitude of features to better show the gaps in the current engineering tool field. These features are based on what was identified from the tools in the literature review. While all features are collectively covered when considering all tools together, no single tool provides complete functionality. If a tool contains a feature it is marked with a green yes, the red no means the tool does not contain the feature and the yellow question mark means that the tool either contains the feature but in a limited way or the tool doesn't specify the existence of the feature but most likely has a limited version of it. The table of tools can be found on the next page in Table 1.

Table 1 Table of Tools

Table of Tools

Source	Tool	Features														
		FBD	Beams	CAM	CAD	Bolts	Mechanics	Centralized	Tutorials	Materials	FEA	Available	Simulation	CFD	Fast?	Traceable
[4]	Turbine Blade Design Automation	no	no	no	yes	no	no	no	no	no	yes	no	no	no	yes	?
[8]	TRADES	no	no	no	no	no	yes	yes	no	?	no	no	no	no	no	no
[9]	Virtual Product Modelling	no	no	no	yes	yes	no	yes	no	yes	no	?	no	no	yes	yes
[10]	ClearCalcs	yes	no	no	no	no	no	yes	no	no	no	yes	no	no	yes	no
[11]	SkyCiv	yes	Yes	no	no	no	no	yes	no	no	yes	yes	no	no	yes	no
[12]	WebStructural	yes	yes	no	no	yes	no	yes	no	yes	no	yes	no	no	yes	no
[13]	Engineering Toolbox	yes	?	?	no	no	?	yes	no	no	no	yes	no	no	yes	no
[14]	STRIAN	yes	no	no	no	no	no	no	no	no	no	yes	no	no	yes	no
[15]	MechaniCalc	yes	yes	no	no	yes	no	yes	yes	?	yes	yes	no	no	yes	yes
[16]	CalcForge	yes	yes	no	no	no	no	yes	yes	yes	no	yes	no	no	yes	no
[17]	BeamGuru	yes	yes	no	no	no	no	yes	yes	no	yes	yes	no	no	yes	yes
[18]	StruCalc	yes	yes	no	no	no	no	yes	yes	yes	yes	yes	no	no	yes	yes
[23]	Transmission design	no	no	no	no	no	yes	no	no	no	no	no	no	no	no	no
[24]	Education Tool	no	no	no	no	no	no	no	yes	no	no	no	no	no	no	no
[25]	CAM shape optimization	no	no	yes	no	no	yes	no	no	no	no	no	no	no	no	no

Source	Tool	Features														
		FBD	Beams	CAM	CAD	Bolts	Mechanics	Centralized	Tutorials	Materials	FEA	Available	Simulation	CFD	Fast?	Traceable
[26]	Turbine Blade Design Automation	no	no	no	yes	no	no	yes	no	?	?	no	no	no	yes	yes
[27]	Elmer	no	no	no	no	no	no	no	no	no	yes	yes	yes	yes	no	no
[27]	OOFEM	no	no	no	no	no	no	no	no	no	yes	yes	no	yes	no	no
[27]	Moose	no	no	no	no	no	no	no	no	no	no	yes	yes	yes	no	no
[27]	OpenFOAM	no	no	no	no	no	no	no	no	no	no	yes	no	yes	no	no
[27]	SU2	no	no	no	no	no	no	no	no	no	no	yes	no	yes	no	no
[27]	Kratos	no	no	no	no	no	no	no	no	no	no	yes	yes	yes	no	no
[27]	CoolFluid3	no	no	no	no	no	no	no	no	no	no	yes	yes	yes	no	no
[27]	MBDyn	no	no	no	no	no	yes	no	no	no	no	yes	yes	?	no	no
[27]	OpenCASCADE	no	no	no	no	no	no	no	no	no	yes	yes	no	no	no	no
[27]	Salome	no	no	no	no	no	no	no	no	no	yes	yes	no	no	no	no
[28]	Variational Design Technology (VGX)	no	no	no	?	no	yes	no	no	no	no	no	no	no	yes	no
[29]	Inventor automation	no	no	no	yes	no	no	no	no	no	no	yes	no	no	yes	yes
[30]	AI enhanced engineering to order tool	no	no	no	yes	no	no	no	no	yes	no	no	no	no	no	yes
[31]	Engineering Handbook	no	no	no	no	no	no	yes	yes	yes	no	no	no	no	yes	yes

2.5.2 Research Gaps

Extensive has gone into creating various tools for making the engineering process more streamlined. But there are still gaps in the current field of engineering tools. The identified research gaps are:

1. There is no existing centralized, automated engineering tool that combines knowledge-based engineering, preliminary parametric CAD generation, material databases, and knowledge retention in a single, open-source framework. Existing solutions are fragmented across proprietary CAD platforms, scripting tools, and isolated calculators.
2. Most existing KBE and automation tools are built on proprietary or legacy platforms (e.g., Visual Basic, C++, commercial CAD APIs), limiting accessibility, extensibility, and openness. There is a lack of research exploring the use of modern open-source technologies (Python, CADQuery, PyQt) to create scalable, modular engineering frameworks that democratize KBE.
3. While KBE tools often focus on automating the design process, there is little integration with knowledge retention, and file structure generation. Current tools fail to automatically capture the metadata, requirements, and design rationale alongside the CAD files, leading to inefficiencies and miscommunication in engineering workflows.

2.5.3 Research Questions

From the identified research gaps above the following research objectives were created.

1. How can an open-source engineering framework integrate knowledge-based engineering with parametric CAD and knowledge retention to streamline preliminary design?
2. Can a minimum viable product, open-source engineering tool bridge the gap between hand calculations and high-level simulations by automating CAD design and knowledge retention?
3. Can an open-source engineering tool streamline project management by automating file structures and capturing metadata to increase knowledge retention and decrease black box syndrome?

2.5.4 Tool Requirements

From the research questions above the following tool requirements were formulated to best bridge the gaps.

Table 2 Tool Requirements

Tool Requirements

Number	Name	Requirements
		Openness and Licensing
1	Open Source	Tool must be freely accessible to the public for editing and use.
2	Open-Source Utilization	Tool must Utilize free, open-source libraries.
		Integration and Framework
3	Centralized	All tools must connect through a main GUI.
4	Framework	Tool must be a frame that supports expansion.
5	External Links	Tool must have a section containing links to external tools.
		Engineering Functionality
6	KBE Utilization	Tool must use Knowledge Based Engineering to generate parametric CAD models.
7	Engineering Efficiency	Tool must target iterative inefficiencies in the design process.
8	Engineering Aids	Tool must support engineers through as much of the design process as possible.
9	Engineering Database	Tool must contain section for databases and be able to reference the databases for KBE design. The user must also be able to view and edit the database.
10	Portable	Must be able to work on any computer without need of internet.
11	Standalone	Tool must not rely on any external resources including internet and webservices.
		Organization
12	Knowledge Retention	Tool must include user calculator inputs and notes as part of the results to reduce black box syndrome and increase knowledge retention.
13	File Organization	Tool must be able to create project folders, and everything created by the tool for a project must be saved under that project folder to prevent disorganization.

3 System Design Overview

This section will give an overview of what technical design decisions were made to best accomplish the requirements and overall goals of this project. This includes software, libraries, design philosophy, architecture, and overall GUI consideration. It is important to note that this software project is being developed by a mechanical engineer who has little experience coding, so some decisions were made to ease the coding process knowing that a better but more difficult choice probably existed.

3.1 Software Choice

The choice of software for this project is imperative and must balance capability with ease of use. The software must also be able to support open source but remains modular, expandable and maintainability. For this project a couple of software packages were looked at including excel [32], MATLAB [33] and Python [34]. Excel is extremely easy to use and is the most common platform out of the three considerations. However, Excel is not free and does not have enough capabilities to perform the required functions in a reasonable way. MATLAB is another great tool that is not as easy as excel but is not difficult either. MATLAB also has great capabilities being able to expand with installation of libraries. However, MATLAB is not free nor is it popular, meaning it does not have as many libraries as Python. So, for this project Python will be used as it is a free programming language that balances capabilities with ease of use. Python is also extensively used in industry, being the most popular programming language in the world [35]. Python's extensive libraries on top of this make it the clear choice for developing this engineering tool.

The main idea behind this tool is not to reinvent any technology or information so if there is a tool that exists that can do a feature required for this engineering tool it will be used. This is where Python's extensive libraries come in. The table below shows all of the libraries used in the engineering tool as well as what each library does. The most important libraries in this project are PyQt6 [36], CADQuery [37], and SQLite [38]. PyQt6 is the library that makes the GUIs that the user can interact with. This library is very intuitive and comes with a drag and drop GUI creator called QT designer [39] meaning that a developer who is not well adjusted to programming can easily create GUIs. This helps make the tool future proof allowing future engineers to create GUIs easily for custom calculators. CADQuery is the next most important library which is a library that creates CAD models using parametric code in python. This

means that any 3D model can be created using only parametric code. The final most important library used in this project is SQLite which is a database library that stores mass amounts of data in a local compressed file. The rest of the libraries can be seen in table 3 below.

Table 3 Libraries used on project

Libraries	Description
	Core Libraries
Python [40]	Main programming language used for this project
PyQt6 [36]	A Python version of Qt which is a GUI and sub widget language
CADQuery [37]	A Python based parametric CAD modeling library
SQLite [38]	Embedded database system
	Numerical and Computational Tools
Numpy [41]	Provides basic mathematical features such as matrices
Pandas [42]	Used for reading and managing Excel files
Nlopt [43]	
CasADI [44]	Libraries for numerical optimization
Multimethod [45]	Enabled function overloading in Python
Mesh [46]	
NumPy-STL [47]	
EZDXF [48]	libraries for 3D mesh handling
	Plotting
Matplotlib [49]	Used to generate 2D and 3D plots
PyQtGraph [50]	Used to generate 2D plots in real time within PyQt

3.2 Tool Architecture

This section will cover the code structure of the engineering tool. Since the idea behind this tool is to make it as modular and future proof as possible the architecture needs to be well thought out and documented. It must also be able to set the rest of the tool up to satisfy all requirements. The figure below shows the overarching structure of the project where the Engineering tool contains many domains pertaining to the engineering process. Currently the domains that the tool covers are mechanical design, project management, and controls. The number of domains can be expanded to include more domains.

Each domain then contains tools, calculators or databases which can also be expanded.

- A calculator is a python-based system that exists within the engineering tool system that takes inputs and calculates outputs based on equations and algorithms.
- A database is an SQLite based system that stores large amounts of information that can then be accessed by the user.
- A tool is any external resource that does not necessarily fall under the scope of the project but is still nice to have access to. This includes textbooks, excel based calculators, or LabView scripts.

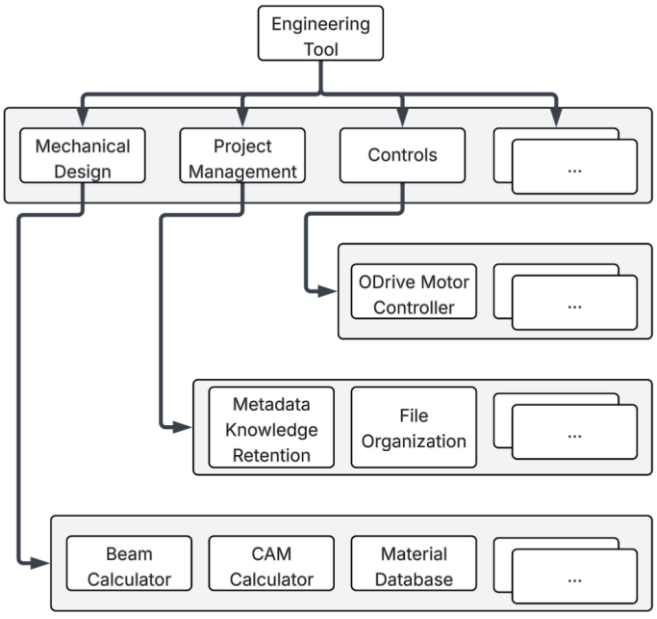


Figure 5 Overall tool structure

3.2.1 File Directory Setup

To support the overall structure of this tool a proper file structure must be set up that allows proper communication between scripts and files. The folder structure can be seen in the figure below. The top folder contains the virtual environment folder and the Tool folder. Under the tool folder is the Gloabls.py that contains all of the global variables and the batch file that launches the tool.

Each tool, calculator and database has its own folder that contains all the information required to run that system. A python-based system will contain three primary files.

- A GUI file (.ui or PyQt) that includes display information,
- A Functions script (.py) that contains equations and algorithms to run the system, and
- A runner script (.py) that ties the GUI and function script together.

This modular system is a relatively simple way to keep things organized but also allows for information to be transported easily between systems while also allowing future expansions.

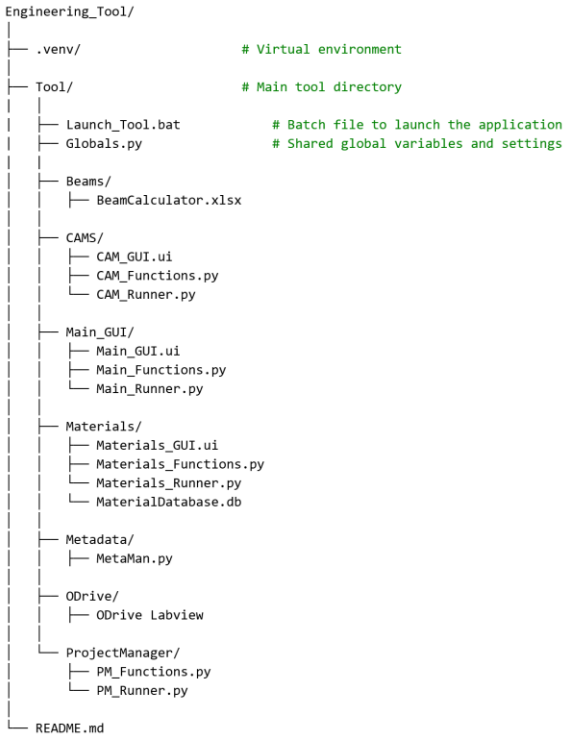


Figure 6 File directory structure of the engineering tool

3.2.2 Code Communication Structure

The overall structure of how the code communicates information between systems can be seen in the figure below. When the tool starts the Main Runner python file runs and opens the main GUI. There is then a widget container in the center of the main GUI that contains all sub-GUIs or widgets. Currently, there are two sub-GUIs, the CAM designer, and the material database. However, a widget container can store an unlimited number of widgets allowing the tool to accommodate. These widgets are stacked on top of each other in the widget container like a stack of cards. When a widget is called, it is brought to the top of the deck. The user is then able to interact with the widget which communicates all data to the secondary layer of the code, which is the project management system Database and global variables. The global variables allow for temporary information to be stored in variables that can be accessed by all scripts in the project. The database is for long-term storage of large amounts of data using SQLite. The project manager then takes this information and stores it in an auto-generated folder file directory where metadata and notes from the tool are stored as well. Finally, the third level contains all of the individual widgets which each contain a GUI, a functions script and a main runner that runs everything. This main runner also communicates with the rest of the tool. This section containing the widgets is infinitely expandable and as more calculators and databases are made they can be added to this section.

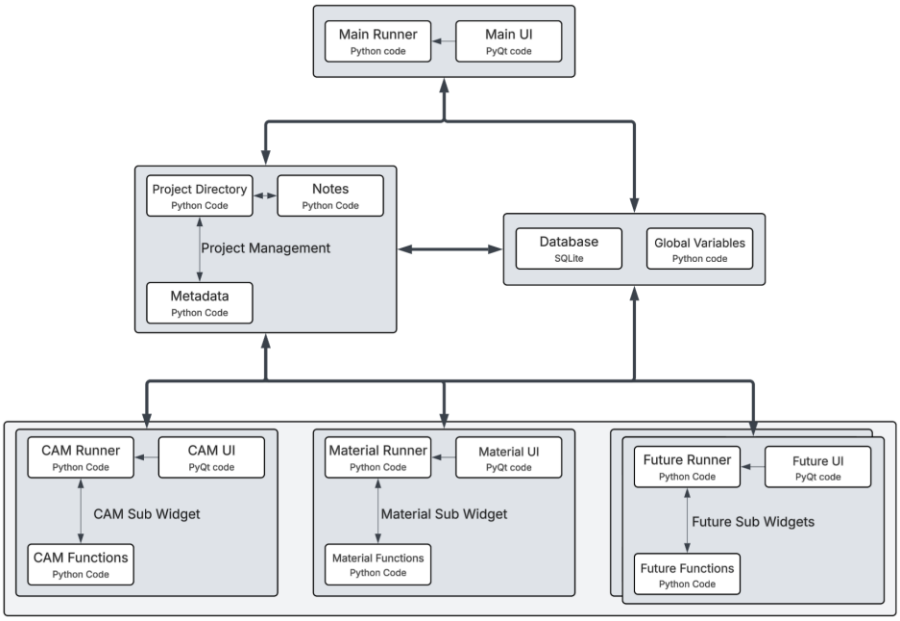


Figure 7 structure of how different systems communicate with each other

3.2.3 User Interface Considerations

The design of the graphical user interface (GUI) plays a key role in creating an engineering tool that will be used frequently by engineers. The interface has been designed to be intuitive and easy to navigate without the need for an external guidebook. The layout follows a consistent structure, with a central area dedicated to displaying the active tool or calculator, while the perimeter of the screen presents key project information and navigation elements.

Another important aspect of the GUI is the color palette. Although color is often regarded as an artistic or subjective choice, it has measurable effects on user focus, mood, and productivity. Studies have shown that blue tones can improve concentration and performance [50]. For this reason, the Engineering Tool employs a blue-gray theme throughout the interface. The main GUI uses a light blue background with dark blue accents, and buttons are styled in light gray for contrast. Widgets use a dark gray background with dark blue accents and light blue buttons for visual consistency.

To maintain a cohesive appearance across the system, any future widgets or modules should follow this same color palette and layout hierarchy. This ensures a unified look and feel throughout the Engineering Tool and helps users transition smoothly between different tools and calculators.

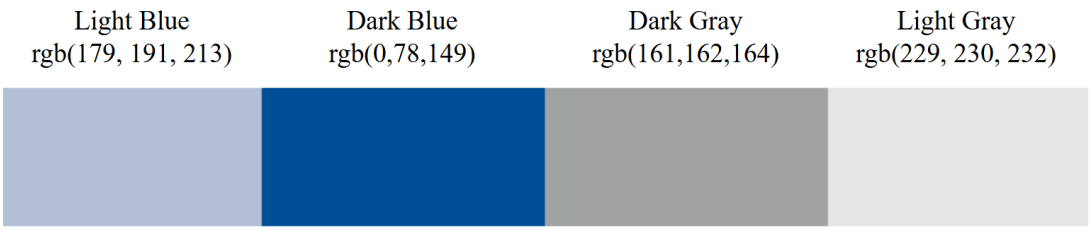


Figure 8 Color palette

The key advantage of this architecture is its scalability and modularity. New tools, calculators, or databases can be added by placing a new folder in the directory and then linking it to the main GUI widget stack to be displayed without standing out from the rest of the widgets. This allows the engineering tool to easily grow as new engineering functions are developed.

3.3 Portability and Deployment

An important part of this project is that the engineering tool is easily accessible and portable without the reliance on external resources. This includes reliance on the internet or anything else that uses web services. Especially with the recent web service crashes it is necessary to be able to function without internet. But since this tool must also be able to be used on any computer and be portable from device to device it cannot just be stored locally on a hard drive. So, for this project a flash drive was chosen to host the entire project. This allows the tool to be portable and work on any computer whilst also remaining isolated from any external resources like the internet.

The flash drive chosen for this project was a 2TB USB SSD from SSK [51]. This SSD takes the benefits of flash drives: portability, compatibility and eliminates the issues: low disk space, slow speeds, risk of corruption. This satisfies the requirements of being portable and independent of external resources.

Figure 9 USB SSD Flash drive used for tool [51]

However, a USB SSD flash drive can only hold files and cannot support full applications which rely on external tools and directories stored on the computer. So, a virtual environment with a portable version of python is installed on the flash drive.

This virtual environment stores all library information as well as directory information meaning that so long as the virtual environment is active all information is available to run. A portable version of python is required for this project because Python requires resources that are only able to be stored on a boot drive containing the operating system. And since overwriting an operating system with one stored on a flash drive is invasive a portable version of python that does not rely on an operating system drive is required. This is where WinPython is used [52]. WinPython is a portable version of python that does not require access to boot drive resources, meaning that it can be stored on the USB SSD flash drive. However, WinPython can only work on computers with a windows operating system which is a limitation of this system.

To launch the tool a simple batch file with two commands is used. The commands first activate the virtual environment and then run the main GUI of the project. The batch file is a windows file that when clicked runs the commands. This is like the operations of an application executable but allows for easy changes which are ideal for software development. The batch file commands can be seen below.

```
@echo off
REM Activate virtual environment
call "E:\Py Engineering Tool\Engineering-Tool\.venv\Scripts\activate.bat"

REM Run main script
python "E:\Py Engineering Tool\Engineering-Tool\Tool\Main GUI\MainRun.py"

REM Keep the window open after completion
pause
```

4 Tool Overview

This section describes what the Engineering Tool is, what it looks like, and how it functions. It also provides an example scenario that demonstrates what a typical day might look like for an engineer using the system.

The figure below shows the main GUI of the engineering tool. The left-hand side contains buttons that link to either tools, calculators, or databases. When one is clicked it will either take the user to an external tool, or it will display the calculator or database in the center of the screen. The top and bottom center of the screen contains the project name and project folder directory. The right-hand side contains the metadata section and the notes sections. The metadata section will display collected input and output data from every used calculator. This information can be loaded and displayed in the metadata section to help reduce black box syndrome. The notes section is where engineers can write notes that can be saved to a .txt file and then loaded later.

Figure 10 Main GUI of engineering tool

The figure below shows an example of what an active project might look like. The name of the project is called “Thesis Demo” which is displayed on the top of the screen and the directory that the project is stored in is displayed at the bottom middle of the screen. This represents a unique feature for this tool that every time a new project is created the tool will prompt the user for a directory to store the project, then

it will create a file directory automatically. This saves the engineer time from having to create the folder structure and it standardizes the folder structure between projects meaning things are better organized and information is easier to find between projects. The autogenerated folder structure can be seen in the figure below. The ProjectConfig file contains useful information such as project name and creation date as well as information that allows the tool to identify which folders are project folders created by the tool.

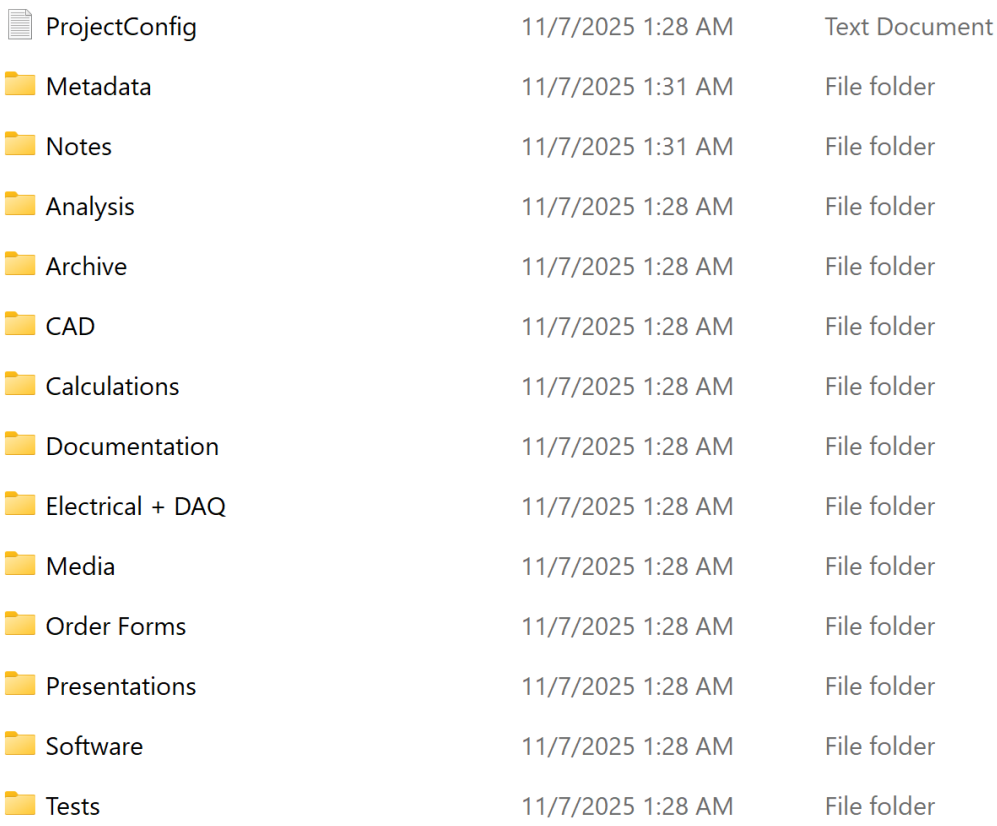


Figure 11 Auto-generated project folder structure

The main part of the screen is an example of one of the calculators. The CAM calculator simply takes user input and displays a 2D preview of the CAM model then generates a 3D CAD model. As you can see though on the right-hand side of the GUI is the metadata section which has noted down all of the input information as well as the output which is a directory to a CAD.step model. This system helps prevent black box syndrome by automatically collecting and saving the inputs and outputs of the CAM calculator. This is so that in the future if someone needs to know how a model was created, they have the inputs, outputs as well as the tool used. The metadata

system takes this a step further by also including a run button which when pressed will automatically populate the inputs with the inputs from the metadata. Meaning that if the CAM CAD model was lost or deleted a new one can be created very easily. The time and date of when the calculator was run is also recorded so that it is easier to track down when a design was created. All metadata .json files are automatically saved to the metadata folder in the project directory. The notes section also contains examples of notes showing that when a note is created and saved, it can be loaded again, and the project name and date of note creation is automatically appended to the top of the note.

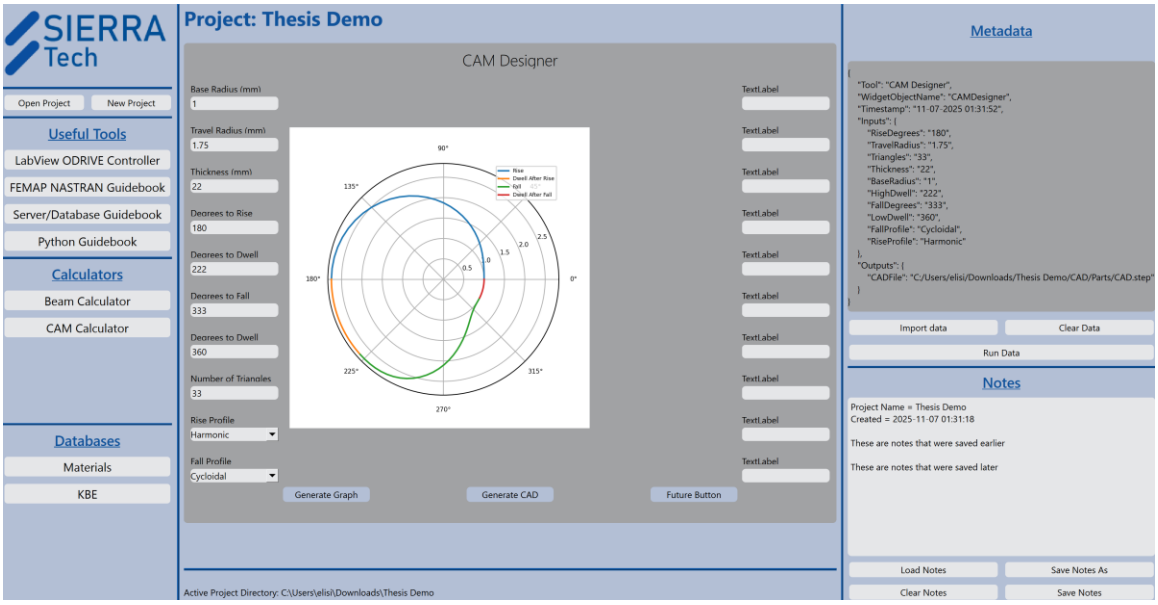


Figure 12 Main GUI displaying CAM, metadata and notes

Some of the tool examples provided are a couple of guidebooks including the FEMAP Nastran guidebook, Python guidebook and the server/database guidebooks which are all guidebooks created to increase knowledge retention. Each guidebook contains information learned from corresponding projects. The LabView ODrive controller is an interesting way of controlling BLDC motors. ODrive is a versatile way of controlling any BLDC motor [53]. It is typically controlled using ROS or Python but a good number of engineers like controlling systems with LabView because it is simple and easy to use. But there is not a library out there that allows LabView to communicate with ODrive. So, This ODrive LabVIEW tool fixes that allowing the user to control any BLDC motor based on speed, torque or position. The tool will also record voltage, current, power, RPM, and position of the motor.

4.1 Day in the Life Demo

This section will run through a quick technical demonstration of the engineering tool. The idea is that an engineer will start a project by clicking on the "new project" button. A file directory will pop up prompting the user to select a place to store the project. Once chosen the following prompt will show up prompting the engineer to name the project.

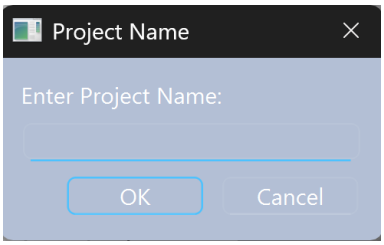


Figure 13 Project name prompt GUI

Once the project is named the name will appear in the upper middle section of the main GUI. This will then allow the engineer to use the rest of the tool. If the user tries to use a calculator without loading or creating a new project. The following warning will appear prompting the user to select a project.

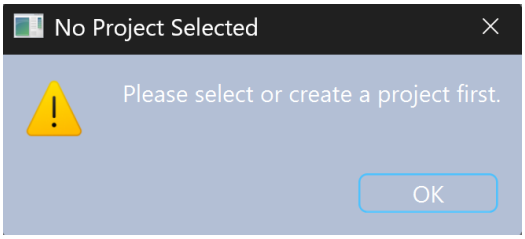


Figure 14 No project selected warning GUI

Next, the engineer may need to figure out the optimal material to choose for their design. They can click on the material database button and pull up the materials database widget which can be seen in the widget below. The engineer can obtain the information that they need by searching for the material in the search bar and then reviewing the material properties displayed. but there is one material that they cannot find in the database. So, they decide to find the material properties online and then upload the information to the database. They select an excel file containing the material properties by clicking the three dots in the upload material section. They then click the add material and the material data is compressed and then added to the database.

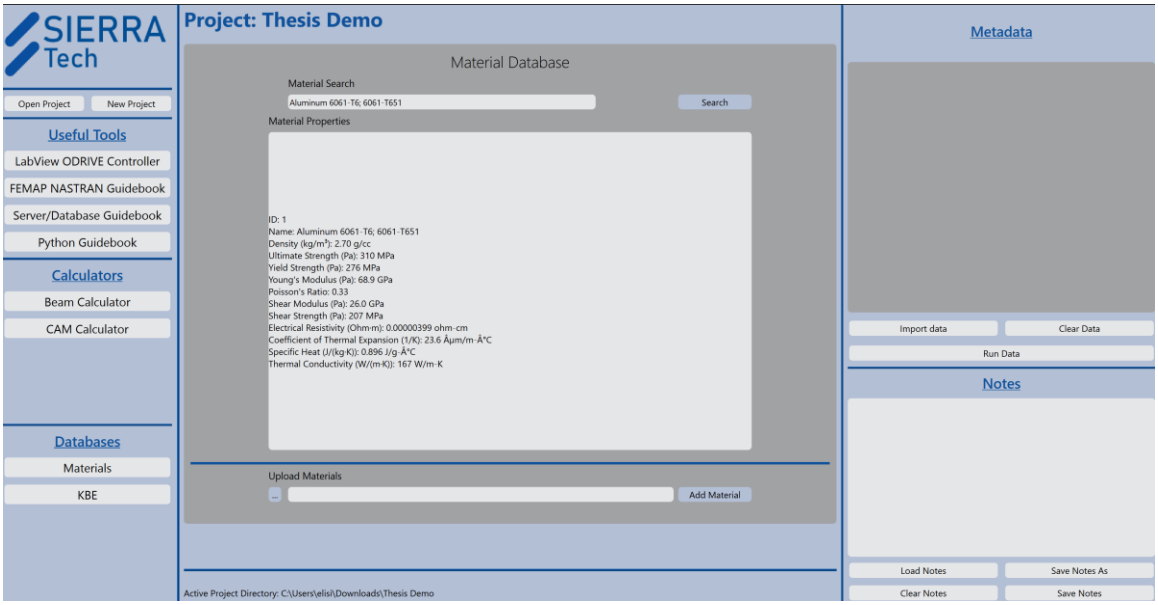
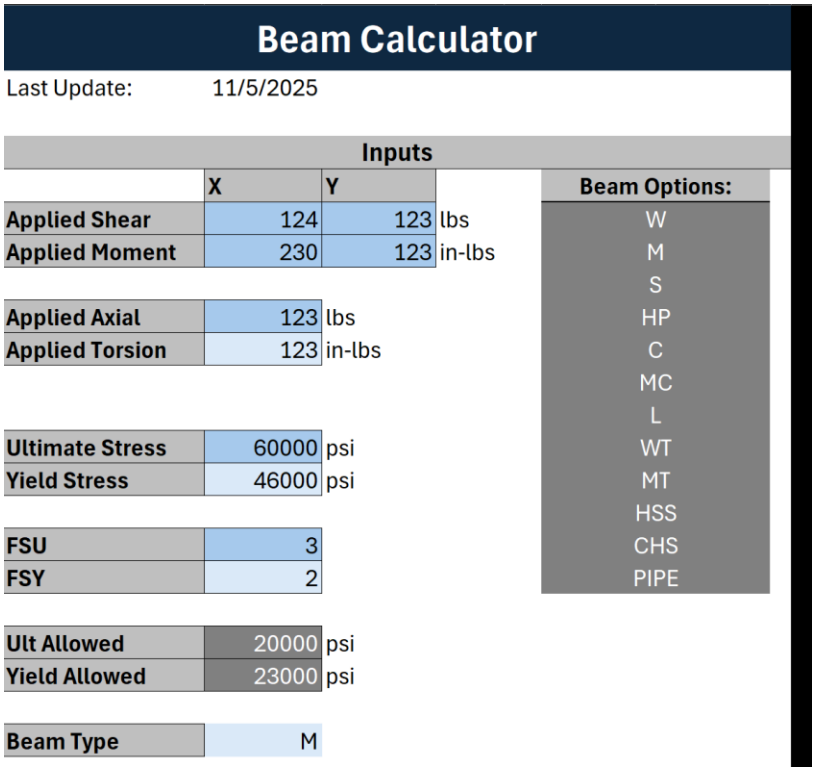


Figure 15 Main GUI displaying material properties

The engineer then needs to develop a CAM design using the material gathered from the database. So, they open the CAM designer and input their desired base radius, travel radius, thickness, number of triangles to develop and degrees to rise, dwell, fall and dwell. The engineer then clicks on the generate graph button and a 2D polar plot is created. Once the engineer is satisfied with the CAM design he clicks on the generate CAD button which generates the CAD model as either a STEP or STL file. The engineer then takes the CAD model and uploads it to their CAD assembly.

A little while later, the engineer realizes that they need specific beams for their design. So, they click on the beam calculator button which opens an excel calculator. The engineer then inputs the forces and moments that the beam must be able to endure. The beam calculator then displays the stresses and margin of safety for every beam. The beam calculator input section can be seen in the figure below.



Beam Calculator

Last Update: 11/5/2025

Inputs

	X	Y	
Applied Shear	124	123	lbs
Applied Moment	230	123	in-lbs
Applied Axial	123		lbs
Applied Torsion	123		in-lbs
Ultimate Stress	60000		psi
Yield Stress	46000		psi
FSU	3		
FSY	2		
Ult Allowed	20000		psi
Yield Allowed	23000		psi
Beam Type	M		

Beam Options:

W
M
S
HP
C
MC
L
WT
MT
HSS
CHS
PIPE

Figure 16 Beam calculator inputs GUI

The engineer can then search through the multitude of beams in the calculated until they find one that has a safety margin close to zero. The results section of the beam calculator can be seen in the figure below.

Results			Axial	Bending Stress		Shear Stress		Torsional Stress		Combined Stress			MS
			Stress	X-axis	Y-axis	X-axis	Y-axis	Flange	Web	Flange	Web	Intersection	
Type	EDI_Std_Nomenclature	AISC_Manual_Label											
S	S24X121	S24X121	3.46	0.89	5.96	31.66	7.44	10.47	7.69	64.98	18.86	10.28	306.80
S	S24X106	S24X106	3.95	0.96	6.30	34.22	9.38	13.27	7.55	73.61	21.23	11.17	270.71
S	S24X100	S24X100	4.20	1.16	9.41	49.16	8.26	14.10	12.07	99.41	25.68	14.72	200.19
S	S24X90	S24X90	4.64	1.23	9.81	52.13	9.62	17.69	12.71	110.44	27.99	15.63	180.09
S	S24X80	S24X80	5.23	1.31	10.25	55.48	11.83	21.88	12.58	123.82	30.36	16.75	160.53
S	S20X96	S20X96	4.36	1.40	8.87	38.49	9.05	13.47	11.71	81.42	26.01	14.57	244.65
S	S20X86	S20X86	4.86	1.49	9.32	41.22	10.81	17.02	12.21	92.11	28.66	15.60	216.13
S	S20X75	S20X75	5.59	1.80	13.32	57.90	11.44	21.30	17.02	127.17	35.95	20.64	156.26

Figure 17 Beam calculator results

After figuring out which beam the engineer requires, they realize that they have misplaced their CAD for the CAM design. Instead of trying to redesign the CAM from scratch they look in the Metadata folder under the main project folder to find the metadata file created from when they made the CAM design in the first place. Once located they can load and then run the metadata file generating the 3D model all over again allowing the engineer to continue working with much faster work processes thanks to the tool.

The engineers then start putting everything together. The beams get bolted together and a CAM motor system gets added. The engineers then want an ability to control the CAM system, so they hook the CAM up to a brushless DC motor which is then hooked up to an ODrive motor controller. But unfortunately, none of the engineers know Python, C or Ros very well and the online ODrive motor controller does not give enough versatility needed to control the CAM. So, they use the LabVIEW ODrive BLDC motor controller. This controller can control the motor with velocity, torque or position, is able to record data and external LabVIEW commands can be sent to automate some processes. This system can be seen in the figure below.

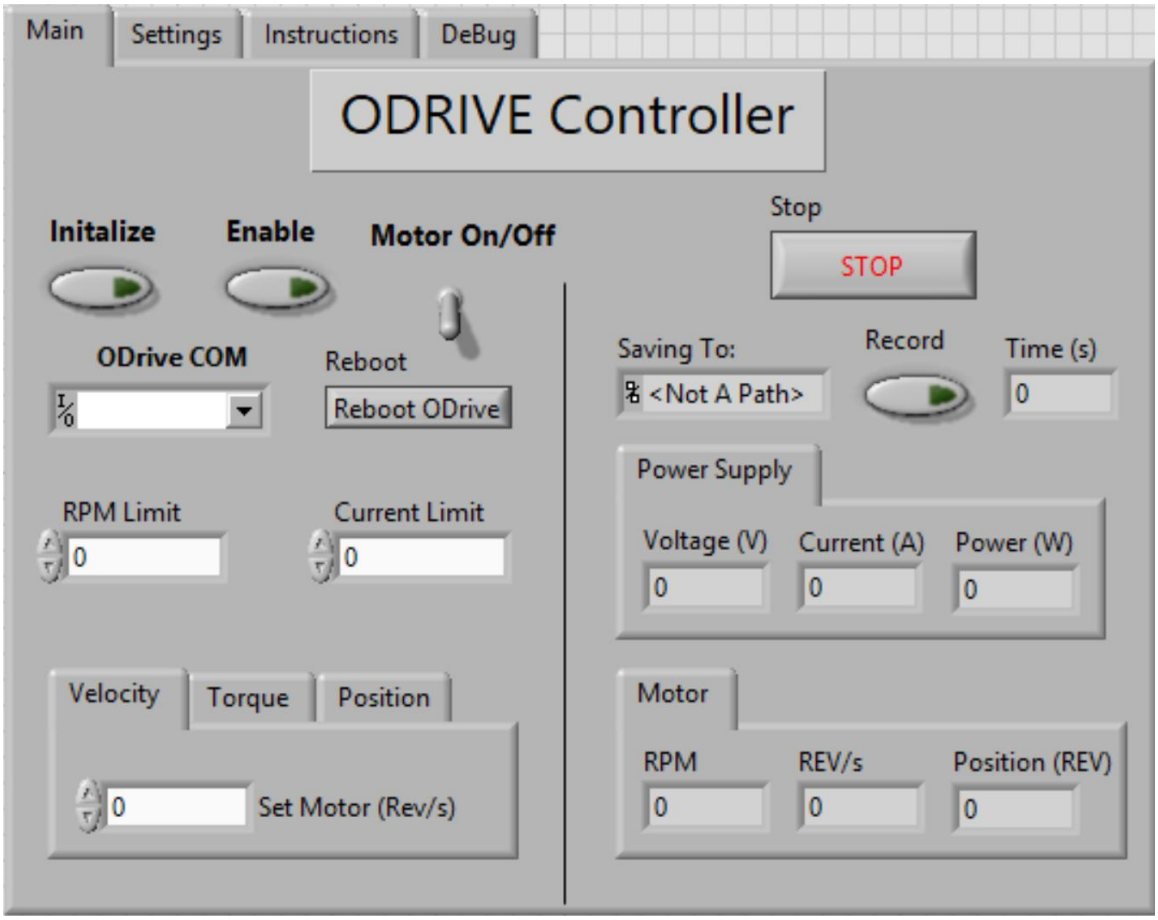


Figure 18 BLDC ODrive controller

5 System Methodology

This section will go into detail about how every system works. This includes the python code and the equations used to drive the calculators. This includes the PyQt user interfaces, the project management systems, the materials database, the CAM calculator, and the beam calculator.

5.1 Main GUI

The main graphic user interface (GUI) is the main and first thing that users interact with when using the engineering tool. It is split up into two main parts: the GUI that the user interacts with and the runner which connects everything to the GUI and makes everything work. The GUI was created in QT Designer which is a drag and drop software that creates UI files. For development purposes this UI file is linked to the runner code as a file but in the final version the UI files will be converted to python code and then linked to the runner as a python class. This Python code can be found in the appendix.

The python code for the runner for the main GUI can be seen below. The first part of the runner is to link the buttons, labels and other features to the python code so that they become functional. This is shown below for buttons, labels, widgets and the logo. When a button is pressed it runs the proper function. Only a couple of examples for each are shown, the rest of the features are shown in the appendix.

```
super().__init__()
loadUi(MainPath, self)
### Logo SetUp ###
self.SierraTech: QLabel #label that displays Sierra Tech logo
### Button Set Up ###
self.CAMCall: QPushButton #button that opens CAM GUI
self.MaterialCall: QPushButton #button that opens Material GUI
### Label Set Up ###
self.ProjectTitle: QLabel #Label that displays the active project
self.Metadata: QLabel #Label that displays the Metadata
### Center Widget Setup ###
self.MainStack: QtWidgets.QStackedWidget #stacked widget
self.cam_widget=CAMWidget() #define CAM widget
self.mat_widget=MaterialWidget() #define Material widget
self.MainStack.addWidget(self.cam_widget) #add CAM widget to stack
self.CAMCall.clicked.connect(lambda:
self.MainStack.setCurrentWidget(self.cam_widget)) #Display CAM Widget
```

The GUI is designed to be modular and expandible so if future features are needed, they can be coded in a separate sub-GUI or widget. So long as the widget is the correct size it can be added to the widget stack then a button can be added to the main GUI to call the widget. The idea is represented in the figure below where widgets are stacked on top of each other like a deck of cards. More widgets can be added to the stack easily.

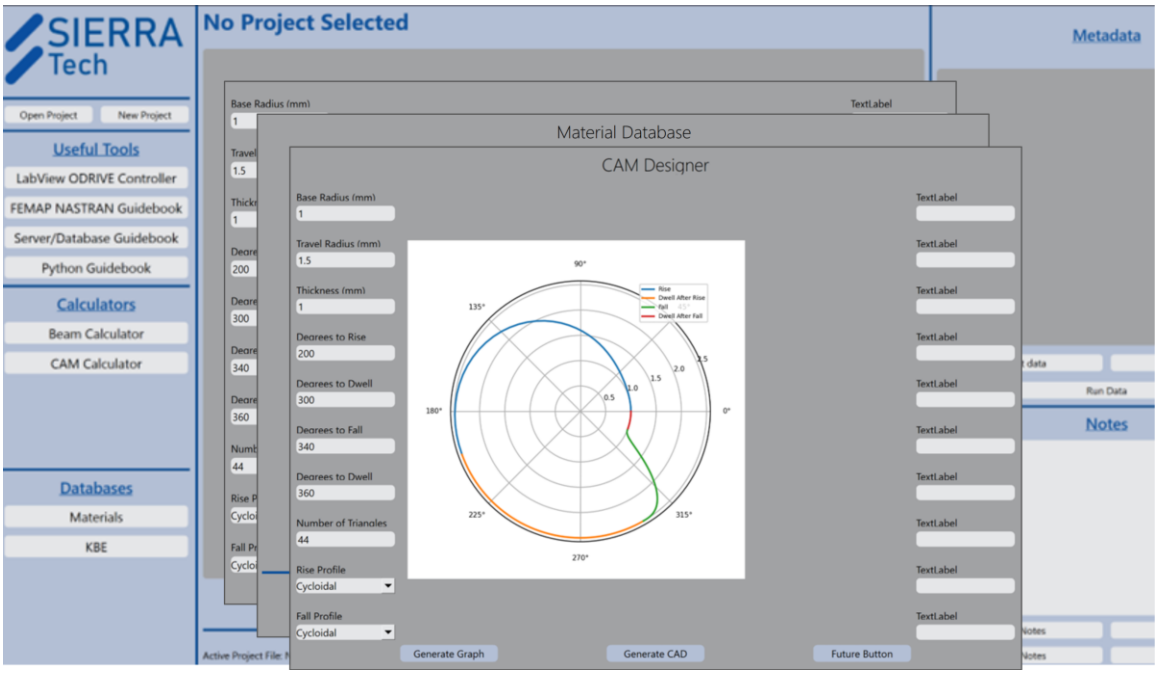


Figure 19 Modular widget stack design

5.2 Global Variables

This engineering tool is a large project that requires proper organization. To place all code within one script would quickly become messy and disorganized so multiple scripts are used. This has the benefit of keeping everything organized but it becomes difficult to move information from one script to another. So, a global variables script is used to contain all essential information that needs to be shared between scripts. When information needs to be shared the script called Globals.py is imported to the file and then any of the global variables can be used. The global variables and example use case of the global variables can be seen below.

Global variable code:

```
### Global Variables ###
ProjectDirectory= None
ProjectName= None
CurrentNotePath= None
Metadata = {}
MetadataPath = None
```

Example use case:

```
import Globals as g
FilePath = g.MetadataPath
g.CurrentNotePath = FilePath
```

5.3 Project Management System

A large part of the engineering tool is to manage projects and information to prevent disorganization and black box syndrome. The first function of this system is to create and load projects and project directories. To do this, the system first displays a file browser for the user to choose a folder to where they want the file to be stored. After that, the user will be prompted to name the project. This name will be stored in a global variable to be used for creating folders and labeling things with the project name. The project folder will then be populated with multiple folders and a project config file. A few snippets of code to accomplish this are shown below. The folder is selected first then the project is named and then the two are combined to create a project path which is then stored in a global variable.

```
folder = QFileDialog.getExistingDirectory(parent, title)
ProjectName, ok = QtWidgets.QInputDialog.getText(parent, "Project Name",
"Enter Project Name:")
ProjectPath = Path(folder) / ProjectName.strip()
ProjectPath.mkdir(parents=True, exist_ok=True)
g.ProjectDirectory=str(ProjectPath) # update project global variable
```

After the main project folder is created the sub folders and project config can be created. The project config folder is a text file that contains the project name and the date of creation. The purpose of this text file is so that the tool can recognize the folder as a project folder. Therefore, it is particularly important that the project config file is not renamed or deleted. The project config and folder creation code is shown below. For simplicity most of the folders are not shown. The full code can be seen in the appendix.

```
folders= ["Presentations",
        "Order Forms",
        "Documentation/Templates",
for f in folders:
        (ProjectPath / f).mkdir(parents=True, exist_ok=True)
### Create the ProjectConfig File ###
configfile = ProjectPath / "ProjectConfig.txt"
with open(configfile, "w") as f:
    f.write(f"Project Name = {ProjectName}\n")
    f.write(f"Created = {dt.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
```

5.4 Metadata System

To prevent black box syndrome the metadata system is used to track information. The metadata system is designed to scan and save data automatically when a calculator is run. The first step for this process is identifying the inputs and then saving them to a JSON file. The code below shows the ScanInputs functions which scans the widget for QLineEdit's and then records the value into a dictionary. The same bit of code is repeated for the QComboBox, but it is not shown here as it is identical. The full code can be located in the appendix.

```
def ScanInputs(ui, ToolName=None):
    inputs={}
    for widget in ui.findChildren(QLineEdit):
        name=widget.objectName()
        value=widget.text().strip()
```

The ScanInputs functions must be used in all widgets that contain a calculation. An example of how to use the code in a widget can be seen below. It is ideal to place the below line of code directly after the values from the GUI are defined. The values are stored and tracked using the global variables system.

```
### Get input from UI ###
Input = float(ui.Example.text())
g.Metadata=ScanInputs(ui)
```

The next step of the metadata system is to save everything to a JSON file to be used later. The code below shows a snippet of the SaveData function. Its primary job is to combine the tool name, widget name, timestamp, inputs and outputs into a JSON file. To use this function when developing a new widget, the developer can simply place the function at the end of the output section of the widget input the tool name and

output and then when the widget is run it will save the metadata automatically. A use-case example of using the code can be seen below.

```
def SaveData(ui, ToolName = None, outputs=None):
    data={
        "Tool": ToolName,
        "WidgetObjectName": ui.objectName(),
        "Timestamp": dt.datetime.now().strftime("%m-%d-%Y %H:%M:%S"),
        "Inputs": g.Metadata.get("Inputs",{}),
        "Outputs": outputs or {}
    }

    with open(MetaPath, "w", encoding="utf-8") as f:
        json.dump(data, f, indent=4)
```

Example use case:

```
outputs = {"CADFile": filename}
SaveData(ui, ToolName="CAM Designer", outputs=outputs)
```

5.5 Notes System

The notes system is another system that aims to retain knowledge and decrease black box syndrome. The note system is simple in that it allows users to write anything in the notes section of the main GUI and then save it. The user can also load previous notes and save new information to that file. The code below shows how the SaveNotesAs function works. It takes the current project directory global variable and then opens the notes folder under it and creates the text file. The text file is then opened and the notes from the user are written onto the file as well as the project name and date of when the notes were created.

```
def SaveNotesAs(self,parent=None):
    NotesPath=Path(str(g.ProjectDirectory)) / "Notes"
    NotesPath.mkdir(exist_ok=True)
    NotesFile = NotesPath / f"{NotesName}.txt"
    text= self.Notes.toPlainText()
    with open(NotesFile, "w", encoding="utf-8") as f:
        f.write(f"Project Name = {g.ProjectName}\n")
        f.write(f"Notes Created = {dt.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')}\n")
        f.write("\n")
        f.write(text)
```

The load notes and save nates functions are simple and work by loading a text file to the main GUI and allowing the user to edit and add new information. The file can then either be saved to the file or saved as a new text file. Snippets from both the load notes and save notes function can be seen below.

```
def LoadNotes(self):
    NotesFolder = Path(str(g.ProjectDirectory)) / "Notes"
    FilePath, _ = QtWidgets.QFileDialog.getOpenFileName(self, "Open Notes",
str(NotesFolder), "Text Files (*.txt)")

def SaveNotes(self,parent=None):
    text = self.Notes.toPlainText()
    with open(g.CurrentNotePath, "w", encoding="utf-8") as f:
        f.write(text)
```

5.6 Material Database

An important part of designing something is figuring out what material to use for it. So having a tool such as a material database that can search, and display material properties can be very powerful. Plus, to keep with the modular and expandability requirements of the tool the user is also able to easily upload materials to the database. Shown below is the main GUI of the material database. The user is able to search for materials which complete the text input based on a custom search engine. The material properties are then displayed in the main window. To upload material properties that do not exist in the database the user can select an excel file containing the desired material information and then upload it to the database.

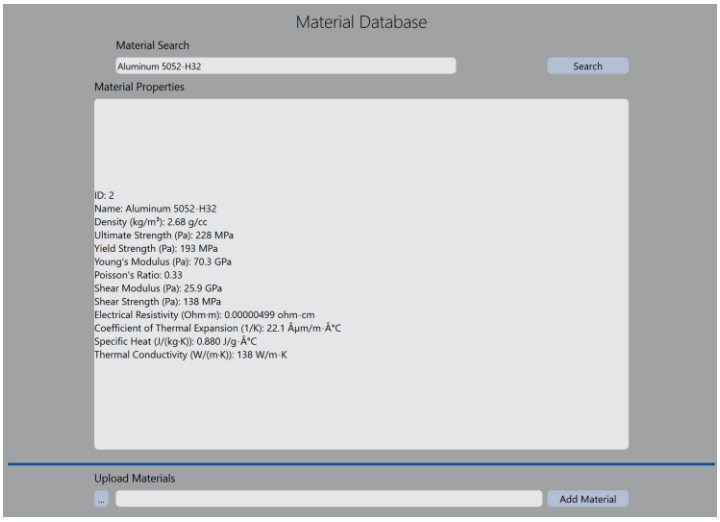


Figure 20 Material Database GUI

The database uses SQLite [38] to store large amounts of information in a long-term local compressed file. This is different from regular SQL databases because it can be stored locally and does not require cloud services to a server. A database is advantageous and used for this project because as more materials get added the database will grow and might run into size issues with the drive. If the system used something like Excel the tool would quickly run out of space as one material is approximately 20 KB but the database which holds 10 materials is only 16 KB. This comes with the cost of not being easily accessible but with a decent search engine the database can display information better than excel.

The search engine works by taking the current text in the search bar and fuzzy matching it with the material names in the database. This fuzzy search engine will also look for matches anywhere in the material name. This allows the user to type in a string and then select a material without having to type out the entire thing. This search engine system can be seen in the figure below.

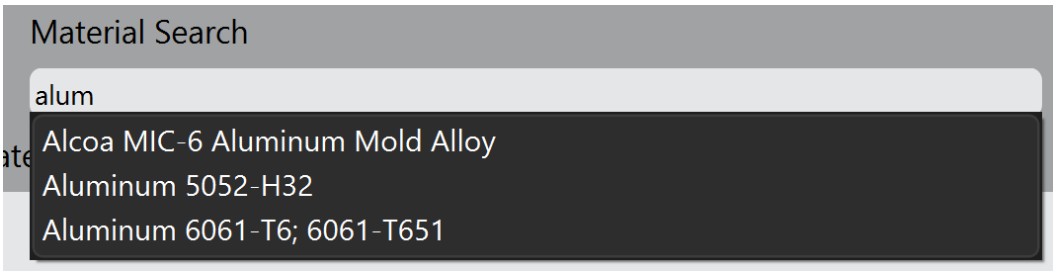


Figure 21 Search engine example

The search engine works by using the get_close_matches function from the difflib library. A small snippet of code from the matsearch function shows how this function is used. It searches all material names in the database and compares them to the user's search. It then returns all close matches.

```
def matsearch(Search, db_path: Path = db_path):
    if len(substringMatches) < limit:
        fuzzy_matches = get_close_matches(Search, names, n=limit, cutoff=0.5)
        substringMatches.extend(m for m in fuzzy_matches if m not in
substringMatches) # merge lists without duplicates
    return substringMatches[:limit]
```

These close matches are then displayed to the user. When the user clicks on one it replaces the users search with that string. When the user clicks the search button it searches for the material in the search bar and displays the properties in the large

results window. The function below shows the readmat function which takes a material name and then returns all the properties of that material.

```
def readmat(name, db_path: Path = db_path):
    with sqlite3.connect(str(db_path)) as con:
        cur = con.cursor()
        cur.execute("SELECT * FROM materials WHERE name = ?", (name,))
        return cur.fetchall()
```

Another important aspect of the material database is its ability to expand with ease. If the user finds that a specific material does not exist in the database, they can upload an excel document containing the material name and properties. The excel document must be formatted correctly for the material database to recognize the information. The general format is that the first cell (A1) must be the name of the material. Then the names of the material properties must exist under that in the A column, and the corresponding property values must be next to the property name in the B column. The order of the material properties does not matter. The function that allows the upload and search of an excel document and then store that information in an SQLite database can be seen below. The function uses the Pandas library [42] to interact with the excel document. A dictionary of words is then used to search for material properties. For simplicity only a few words are shown in the word dictionary the full dictionary can be seen in the appendix. The function then loops through the cells in the excel sheet and at each cell it loops through the words in the dictionary and if there is a match the value in the next column B is returned.

```
def excelSearch(excel_path):
    data=pd.read_excel(excel_path, header=None)
    words = {
        'density': ['density'],
        'shear_modulus': ['shear modulus'],
for _, row in data.iterrows(): # Iterate through each row in the DataFrame
        a=row[0] # First column of excel
        b=row[1] # Second column of excel
        if isinstance(a, str): # Ensure 'a' is a string to avoid errors
            a_stripped = strstrip(a) # get rid of spaces and cases
            for key, variants in words.items():
                if any(strstrip(variant) in a_stripped for variant in
variants):
                    results.setdefault(key, b)
    return results
```

The excelSearch function can then be used in the MatWrite function which takes the properties and writes them to the SQLite database. The code to do this is shown below. The first step is to run the excelSearch function to retrieve the properties from the chosen excel file. These properties are then thrown into a variable that specifies to insert the properties into the material database and then lists off the names of the properties. For simplicity only two properties are shown here but the full list is located in the appendix. The function then calls the SQLite database and then inserts the values.

```
def MatWrite(excel_path, db_path: Path = db_path):
        properties=excelSearch(epath)
        name=properties.get('name')
        SqlInsert=("INSERT INTO materials (name, density...
        values=(
             name,
             properties.get('density'),
             properties.get('ultimate_strength'),
        with sqlite3.connect(str(db_path)) as con:
                cur = con.cursor()
                cur.execute(SqlInsert, values)
                return cur.lastrowid
```

5.7 CAM Calculator

The cam calculator is meant to be an easy-to-use tool that will create 2D and 3D models of cams based on user inputs. The user can input base radius, travel radius, thickness as well as rise, fall and dwell degrees. This will then create a 2D polar plot of the CAM profile and then the user can generate a 3D CAD model that will extrude the 2D polar plot by the thickness which will then be stored in either an STL or STEP file.

The main user interface can be seen below. The user interface was designed using the QT designer tool and the resulting PyQt code is in the appendix. The left side of the GUI are the inputs to the calculator where the user can input the base radius and travel radius of the CAM. The travel radius is the travel distance not the total radius. Total radius is base radius plus the travel radius. The user can then input the degrees to rise, fall and dwell which must start at 0 degrees and end at 360 degrees. The rise and fall profiles can be determined by the user as well to be either linear, parabolic, harmonic or cycloidal. The generate graph button will display the 2D polar plot in the

center of the GUI and the generate CAD button will generate a CAD model of the CAM by extruding the 2D polar plot by the specified thickness. The remaining button and textboxes are for future use as further development is made.

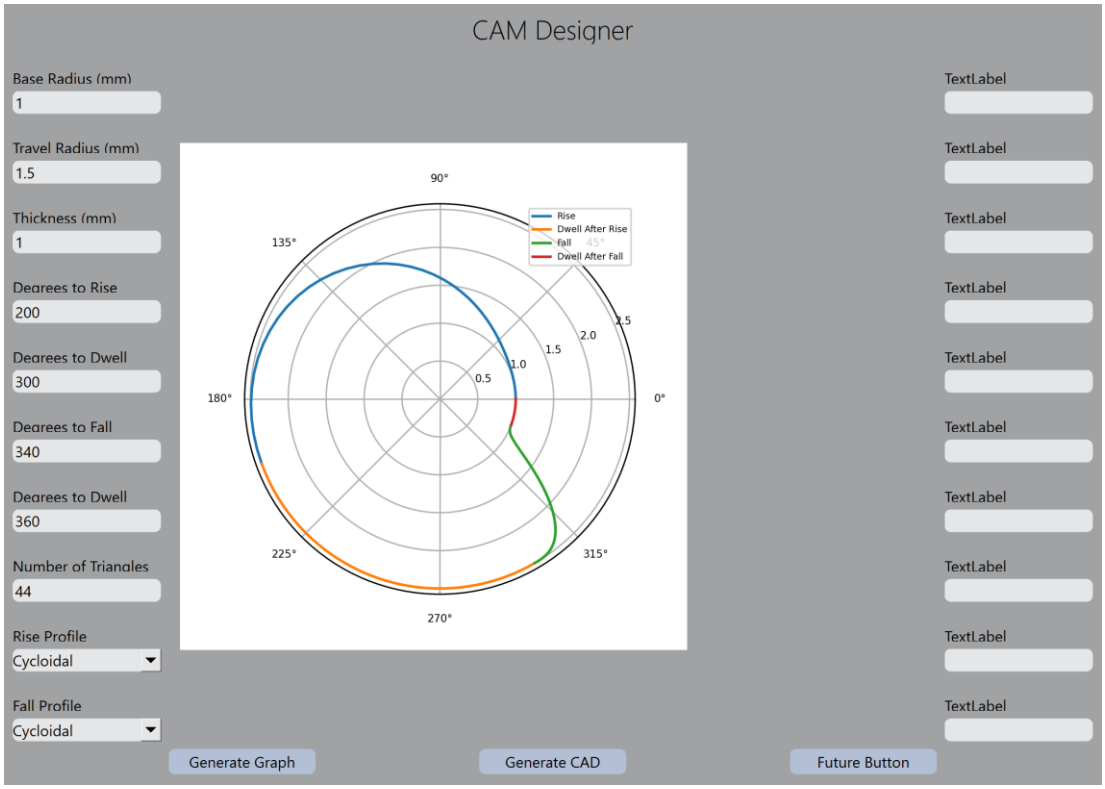


Figure 22 CAM designer user interface

5.7.1 Linear Profile

To make the CAM designer work there are only four equations one for each of the profiles. These equations are then translated into Python code which also contains code to generate the plot and create the 3D CAD models.

The linear profile rise equation is the simplest out of the 4 profiles which is great for machinability, but it can create large and abrupt acceleration changes [54]. The equation can be seen below where L is the target travel radius, β is the target angle, θ is the variable angle and y is the height of the CAM profile given θ. The return/fall equation is pretty much the same, but y is replaced with y-L, θ is replaced with $\theta-\beta_1$ and β is replaced with $\beta_2-\beta_1$. β_1 is the starting angle and β_2 is the final angle.

$$y = \frac{L}{\beta}\theta \tag{1}$$

This equation was then translated to a function in python which is shown below. The function works by using travel radius, base radius, number of triangles and a degree vector containing D1, D2, D3, and D4 which are rise, high dwell, fall, and low dwell, respectively.

Inside the function, the rdfd, rf, and rd arrays are used as logical indicators that determine which segment (rise, dwell, or fall) is being computed. The function then loops through all four motion stages, generating evenly spaced angular points for each using np.linspace [41]. During the rise segment, the displacement is calculated by adding the base radius (BR) to a scaled rise profile, producing a smooth linear increase in cam height. The fall segment mirrors this behavior, decreasing linearly back to the base radius.

Finally, the function returns two arrays: one containing the radial displacement values (Y) and one containing their corresponding angular positions (angle, in radians). These arrays can then be used to generate a polar plot or to construct the 3D cam surface in later functions.

The Python function for the linear equation is located below.

```
def Linear(TR, D, BR, Tri):
    # TR: Travel Radius, D: Degree vector containing D1,D2,D3,D4,
    # BR: Base Radius, Tri: number of triangles
    rdfd = np.array([1, 0, 1, 0])  # Rise/Fall indicators
    rf   = np.array([0, 0, 1, 0])  # Fall indicator
    rd   = np.array([0, 1, 0, 0])  # Previous dwell (adds BR shift)
    angle = []
    Y = []
    for i in range(4): # For loop that runs through the 4 components
        d_start = D[i]
        d_end = D[i + 1]
        delta = d_end - d_start
        theta = np.linspace(d_start, d_end, Tri)
        phi = (theta - d_start) / delta  # Normalized range [0,1]
        if rf[i] == 0:
            y = TR * phi * rdfd[i] + BR + TR * rd[i]
        else:
            # Fall: y = h * (1 - phi)
            y = TR * (1 - phi) + BR
        angle.append(np.radians(theta))
        Y.append(y)
    return Y, angle
```

5.7.2 Parabolic Profile

The parabolic profile is another profile for rise and fall for CAM designs. Parabolic CAM designs are little more difficult to machine than linear profiles but offer lower peak accelerations [54]. The equation can be seen below where L is the target travel radius, β is the target angle, θ is the variable angle and y is the height of the CAM profile given θ. The return/fall equation is pretty much the same, but y is replaced with y-L, θ is replaced with $\theta-\beta_1$ and β is replaced with $\beta_2-\beta_1$. β_1 is the starting angle and β_2 is the final angle.

$$y = \frac{2L}{\beta^2}\theta^2 \tag{2}$$

The equation was then implemented into Python as a function shown below. Like the linear function the parabolic function loops through the four motions segments (rise, dwell, fall, dwell) using the rdfd, rf and rd array indicators to enable and disable certain parts of the equations to either rise, fall or dwell. For the rise motion the displacement is calculated using the parabolic function, while the fall motion uses the mirrored version of this.

The function outputs two arrays: Y containing the radial displacement and the corresponding angles. These arrays can then be used for 2D and 3D plotting in other functions.

```
def Parabolic(TR, D, BR, Tri):
    rdfd = np.array([1, 0, 1, 0])  # Rise/Fall/Dwell flags
    rf   = np.array([0, 0, 1, 0])  # Fall indicator
    rd   = np.array([0, 1, 0, 0])  # Previous dwell
    for i in range(4):
        d_start = D[i]
        d_end = D[i + 1]
        delta = d_end - d_start
        theta = np.linspace(d_start, d_end, Tri)
        beta = delta
        phi = (theta - d_start) / beta  # Normalize to [0,1]
        if rf[i] == 0:
            y = TR * (2 * phi - phi ** 2) * rdfd[i] + BR + TR * rd[i]
        else:
            y = TR * (1 - 2 * phi + phi ** 2) + BR
        angle.append(np.radians(theta))
        Y.append(y)
    return Y, angle
```

5.7.3 Harmonic Profile

The Harmonic CAM profile is a great profile for smooth motion having small acceleration, but it suffers from high jerk values. This makes it great for low-speed applications but are not recommended for high-speed applications [54]. The equation for the Harmonic profile can be seen below where L is the target travel radius, β is the target angle, θ is the variable angle and y is the height of the CAM profile given θ. The return/fall equation is pretty much the same, but y is replaced with y-L, θ is replaced with $\theta-\beta_1$ and β is replaced with $\beta_2-\beta_1$. β_1 is the starting angle and β_2 is the final angle.

$$\frac{L}{2}\left(1-\cos\left(\frac{\pi\theta}{\beta}\right)\right) \tag{3}$$

Like the other profiles, the Harmonic equation was formatted in Python using the same approach. The function shown below again loops through the four different segments (rise, dwell, fall, dwell) using the rdfd, rf and rd array indicators. The main equation inside the loop is the Harmonic equation which uses the indicators to either rise dwell or fall using the mirrored version.

Also like the previous profiles the function outputs two arrays: Y containing the radial displacement and the corresponding angles. These arrays can then be used for 2D and 3D plotting in other functions.

```
def Harmonic(TR, D, BR, Tri):
    rdfd = np.array([1, 0, 1, 0])
    rf   = np.array([0, 0, 1, 0])
    rd   = np.array([0, 1, 0, 0])
    for i in range(4):
        d_start = D[i]
        d_end = D[i + 1]
        d_prev = D[i - 1] if i > 0 else 0  # Avoid index error
        if i==2:
            theta=np.linspace(D[2],D[3],Tri)
            y=(TR/2)*(1+np.cos((np.pi*(theta-D[2]))/(D[3]-D[2]))) + BR
        else:
            theta=np.linspace(d_start, d_end, Tri)
            delta=d_end - d_start
            phase=(theta -d_prev*rf[i])/delta if delta != 0 else 0
            y = (TR/2)*(1-np.cos(np.pi*phase))*rdfd[i] + BR + TR*rd[i]
        angle.append(np.radians(theta))
        Y.append(y)
```

5.7.4 Cycloidal Profile

The final motion profile included in the calculator is the Cycloidal profile, often regarded as one of the smoothest and most dynamically efficient options for CAM mechanisms. The cycloidal law of motion eliminates abrupt changes in acceleration, resulting in continuous velocity and acceleration curves. This makes it ideal for high-speed applications where vibration, noise, and wear are critical concerns.

However, the cycloidal motion law produces higher peak acceleration values compared to the parabolic profile, and its geometry is more complex to design and machine. The equation defining the rise portion of a cycloidal motion is shown below, where L is the target travel radius, β is the target angle, θ is the variable angle and y is the height of the CAM profile given θ. The return/fall equation is pretty much the same, but y is replaced with y-L, θ is replaced with $\theta-\beta_1$ and β is replaced with $\beta_2-\beta_1$, where β_1 is the starting angle and β_2 is the final angle.

$$L\left(\frac{\theta}{\beta}-\frac{1}{2\pi}\sin\left(\frac{2\pi\theta}{\beta}\right)\right) \tag{4}$$

Like the other profiles, this equation was implemented in Python using the same modular structure. The function below loops through the four motion segments (rise, dwell, fall, dwell), using the rdfd, rf, and rd arrays as logical indicators for motion phase control.

During the rise segment, the function utilizes the cycloidal displacement equation and during the fall segment, the same logic is mirrored to reverse the motion smoothly back to the base radius. The function outputs two arrays: Y containing radial displacement and the corresponding angles which can be used later to generate 2D and 3D graphs and CAD models.

```
def Cycloidal(TR,D,BR,Tri):
    rdfd = np.array([1, 0, 1, 0])
    rf   = np.array([0, 0, 1, 0])
    rd   = np.array([0, 1, 0, 0])

    angle = []
    Y = []

    for i in range(4):
        d_start = D[i]
```

```
        d_end = D[i + 1]
        d_prev = D[i - 1] if i > 0 else 0  # Avoid index error
        theta = np.linspace(d_start, d_end, Tri)
        delta = d_end - d_start
        phase = (theta - d_prev * rf[i]) / delta if delta != 0 else 0
        if rf[i]==0:
           y = TR*(phase-
(1/(2*np.pi))*np.sin(2*np.pi*phase))*rdfd[i]+BR+TR*rd[i]
        else:
            beta = D[3] - D[2]
            theta_local = theta - D[2]
   y= TR*(1-(theta_local/beta-
(1/(2*np.pi))*np.sin(2*np.pi*theta_local/beta)))+BR
        angle.append(np.radians(theta))
        Y.append(y)
    return Y, angle
```

5.7.5 CAM Graphing

Afte the CAM profiles are created they can then be graphed on a 2D polar plot. This is completed using a function called CAMPlot. The first step in CAMPlot is to initiate the function, import libraries and then retrieve values from CAM GUI. The code snippet below does just that, collecting values from text and storing them in variables to be used later. Note how the degrees are stored in one array called Df. The values are also stored in the global variables by first using the ScanInputs function from the metadata system to grab the inputs and then store them in a global variable.

```
def CAMPlot(ui):
    from matplotlib.backends.backend_qtagg import FigureCanvasQTAgg as
FigureCanvas
    import matplotlib.pyplot as plt
    import numpy as np
    from PyQt6 import QtWidgets
    from .CAM_Func import Harmonic, Linear, Parabolic, Cycloidal
    from MetadataManager.MetaMan import ScanInputs, SaveData

    # === Get input from UI ===
    TR = float(ui.TravelRadius.text())
    D1 = float(ui.RiseDegrees.text())
    D2 = float(ui.HighDwell.text())
    D3 = float(ui.FallDegrees.text())
    D4 = float(ui.LowDwell.text())
    Df = [0, D1, D2, D3, D4]
```

```
Tri = int(float(ui.Triangles.text()))
BR = float(ui.BaseRadius.text())
Thickness = float(ui.Thickness.text())
RisePro = ui.RiseProfile.currentText()
FallPro = ui.FallProfile.currentText()
g.Metadata=ScanInputs(ui)
```

The next step is to then create the plot starting with the rise profile and dwell. The code below initializes the plot and then selects a rise profile based on the selected rise profile by the user. After which the code plots the first rise and dwell profiles before moving on to the fall and final dwell profile. Like the rise profile, the fall section uses the function corresponding to the users' fall profile choice. The figure is then converted to a canvas which can then be displayed in a widget on the PyQt GUI. Finally, an important part of the code below is that it creates a final array for both the angle and amplitude. This is so that a full profile of the CAM can be passed to the STL and STEP generating functions later.

```
# === Create plot ===
fig, ax = plt.subplots(figsize=(4, 4), subplot_kw={'projection': 'polar'})
plt.rcParams['font.size'] = '0.5'
anglefinal = []
Yfinal = []

# === Plot Rise ===
if RisePro == "Harmonic":
    Y_rise, angle_rise = Harmonic(TR, Df, BR, Tri)
elif RisePro == "Linear":
    Y_rise, angle_rise = Linear(TR, Df, BR, Tri)
elif RisePro == "Parabolic":
    Y_rise, angle_rise = Parabolic(TR, Df, BR, Tri)
elif RisePro == "Cycloidal":
    Y_rise, angle_rise = Cycloidal(TR, Df, BR, Tri)
ax.plot(angle_rise[0], Y_rise[0], label="Rise")
ax.plot(angle_rise[1], Y_rise[1], label="Dwell After Rise")
anglefinal.extend([angle_rise[0], angle_rise[1]])
Yfinal.extend([Y_rise[0], Y_rise[1]])

# === Plot Fall ===
if FallPro == "Harmonic":
    Y_fall, angle_fall = Harmonic(TR, Df, BR, Tri)
elif FallPro == "Linear":
    Y_fall, angle_fall = Linear(TR, Df, BR, Tri)
elif FallPro == "Parabolic":
```

```
        Y_fall, angle_fall = Parabolic(TR, Df, BR, Tri)
    elif FallPro == "Cycloidal":
        Y_fall, angle_fall = Cycloidal(TR, Df, BR, Tri)
    ax.plot(angle_fall[2], Y_fall[2], label="Fall")
    ax.plot(angle_fall[3], Y_fall[3], label="Dwell After Fall")
    anglefinal.extend([angle_fall[2], angle_fall[3]])
    Yfinal.extend([Y_fall[2], Y_fall[3]])
    canvas = FigureCanvas(fig)
    canvas.draw()
    ui.plot_layout.addWidget(canvas)
```

5.7.6 CAM Modeling

The final process of the CAM designer is to create a CAD model. This process uses the CADQuery [37] library to extrude the 2D profile by the thickness specified by the user. The first step in this process is to determine where the file should be saved. The code below does this by opening a file browser GUI to the CAD folder in the main project folder using the global variable g.ProjectDirectory. The user can then input a file name and then select to save the CAD file as either a STEP or STL file. The resulting directory is stored in a variable to be used later. This described process can be seen in the snippet of code below.

```
def CADFileName(ui, parent=None):
    from PyQt6.QtWidgets import QFileDialog
    from pathlib import Path
    from MetadataManager.MetaMan import ScanInputs, SaveData
    CADFolder = Path(str(g.ProjectDirectory)) / "CAD"
    filename, _ = QFileDialog.getSaveFileName(
        parent,
        "Save CAM File",
        str(CADFolder),
        "STEP File (*.step);;STL File (*.stl);;All Files (*)")
    print(filename)
    if filename:
        g.Metadata=ScanInputs(ui)
        outputs = {"CADFile": filename}
        SaveData(ui, ToolName="CAM Designer", outputs=outputs)
    return filename
```

Once the file name is determined the CAD model can be created. The first step is to convert the polar coordinates to cartesian. Once this is completed the coordinates can be sent to the CADQuery [37] library where it is extruded by thickness and saved

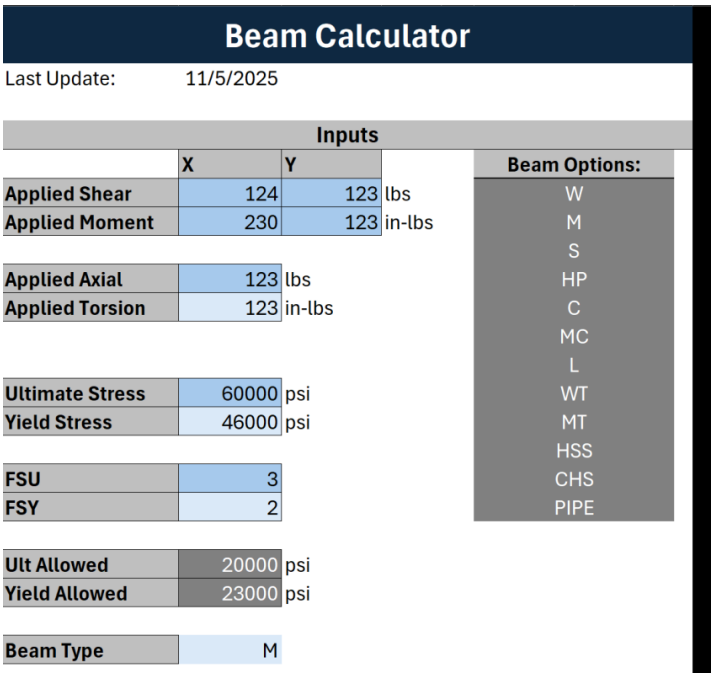
to the defined file that contains the STEP or STL extension. The code for this can be seen below.

```
def STPGen(angle, Y, Thickness, filename):
    x1, y1 = Y[0] * np.cos(angle[0]), Y[0] * np.sin(angle[0])
    x2, y2 = Y[1] * np.cos(angle[1]), Y[1] * np.sin(angle[1])
    x3, y3 = Y[2] * np.cos(angle[2]), Y[2] * np.sin(angle[2])
    x4, y4 = Y[3] * np.cos(angle[3]), Y[3] * np.sin(angle[3])
    try:
        wire = cq.Workplane("XY").polyline(cleaned_points).close()
        solid = wire.extrude(Thickness)
        cq.exporters.export(solid, filename)
        print(f"STEP file saved as '{filename}'")
    except Exception as e:
        print("Failed to generate STEP file:", e)
```

5.8 Beam Calculator

The beam calculator is meant to determine the optimal beam size and shape based on user input. The user can choose a style of beam based on the AISC beam database ranging from I beams to PIPE beams. The user can then input various forces and moments into the beam. If multiple forces are placed along multiple points on the beam the user will have to calculate the equivalent force and moment. The user will then enter ultimate and yield stresses of the chosen material as well as the factors of safety. The beam calculator will then calculate the safety margin for every beam in the ASIC database. This allows the user to determine an optimal preliminary beam design that can then be checked with FEA.

The beam calculator was designed in Excel around the V16.0 AISC beam database which contains approximately 2300 beams and their properties [55]. To use the calculator the user must input forces acting along a 2D cross section of a beam. On this cross section all 6 types of forces can be used: axial, shear in X, shear in Y, torsion and bending moment around the X and Y axis. This means that a free body diagram (FBD) must be created before use. If there are multiple forces acting in the same direction or outside of the 2D cross section plane the equivalent force plus moment must be calculated. The user can then decide on material ultimate and yield stress as well the safety factors which will determine the maximum allowable stresses. Once a beam type is selected the approximate stresses will be calculated. The beam calculator input GUI can be seen in the figure below.



Beam Calculator

Last Update: 11/5/2025

Inputs

	X	Y	
Applied Shear	124	123	lbs
Applied Moment	230	123	in-lbs
Applied Axial	123		lbs
Applied Torsion	123		in-lbs
Ultimate Stress	60000		psi
Yield Stress	46000		psi
FSU	3		
FSY	2		
Ult Allowed	20000		psi
Yield Allowed	23000		psi
Beam Type	M		

Beam Options:

W
M
S
HP
C
MC
L
WT
MT
HSS
CHS
PIPE

Figure 23 Beam calculator input user interface

The results are displayed next to the input user interface in a large table that a user can scroll through to find an optimal beam. The stresses that are calculated are the axial stress, bending stress in X & Y, shear stress in X & Y, torsional stress, and combined stress. In the example used in the figures above and below an I beam is used so for this beam there is flange torsional stress as well as web torsional stress. For the combined stresses 3 points were chosen on the flange web and intersection between the web and flange. The results user interface can be seen below. All stresses are listed in a row for each beam type. A safety margin (MS) is then displayed on the right where the user can then easily pick the beam that closely matches the loads and safety factors.

Results			Axial	Bending Stress		Shear Stress		Torsional Stress		Combined Stress			MS
			Stress	X-axis	Y-axis	X-axis	Y-axis	Flange	Webb	Flange	Web	Intersection	
Type	EDI_Std_Nomenclature	AISC_Manual_Label											
M	M12.5X12.4	M12.5X12.4	33.88	16.10	114.74	339.57	71.62	568.84	386.71	2506.32	682.04	164.43	6.98
M	M12.5X11.6	M12.5X11.6	36.18	17.90	142.55	422.27	72.73	626.88	460.51	2992.04	808.32	196.33	5.68
M	M12X11.8	M12X11.8	35.45	19.11	173.22	485.38	67.57	553.50	435.42	2550.09	764.02	227.42	6.84
M	M12X10.8	M12X10.8	38.68	20.69	186.94	526.17	74.69	657.25	500.76	3192.06	877.79	245.94	5.27
M	M12X10	M12X10	41.69	22.37	194.05	551.78	80.01	758.22	627.64	3779.92	1096.69	257.78	4.29
M	M10X9	M10X9	46.42	29.49	246.18	573.28	90.80	806.94	615.00	3956.63	1077.76	321.48	4.05
M	M10X8	M10X8	51.90	33.07	278.98	649.15	101.10	999.38	774.24	5478.81	1353.41	363.34	2.65
M	M10X7.5	M10X7.5	55.41	34.81	294.37	695.08	108.95	1137.91	855.08	5142.02	1494.04	383.98	2.89
M	M8X6.5	M8X6.5	64.06	49.73	372.93	689.24	131.00	1263.42	902.45	6639.21	1580.76	485.54	2.01

Figure 24 beam calculator results user interface

5.8.1 Fundamentals & Assumptions

The beam calculator utilizes a multitude of different equations but before those equations can be shown the fundamentals and assumptions made for each beam must be addressed. This section will outline the equations at the source of the beam calculator as well as the assumptions made for each beam.

1. Linear Elastic Behavior

All beams are assumed to behave according to linear elastic theory where maximum stress does not exceed the yield stress of the material.

2. Profile Based Stress

Only a scalar maximum stress will be calculated for each force case on each beam using basic algebraic functions. Strain will not be included at this time.

3. Slender Beam Assumption

Some of the governing equations utilize Euler-Bernoulli beam theory which requires beams to be tall and skinny. A slenderness ratio L/h>10 is assumed to ensure that classical beam theory remains valid where L is beam length and h is the width of the smallest side of the beam.

4. Static Loading and Negligible Deformation

All loads are assumed to be static or quasi-static. Dynamic loads are not included in the beam calculator currently. Deformations are also assumed to be small enough to make geometric nonlinearities negligible.

5. Use of AISC Geometric Properties

All beam properties used in the equations of the beam calculator utilize properties from the AISC beam database [55].

6. Axial Stress

Axial stress is computed using the classical uniform stress relation shown in the equation below where F_A is axial force and A is cross sectional area [56].

$$\sigma_A = \frac{F_A}{A} \tag{5}$$

The tool assumes that the load is applied at the centroid with perfect alignment with the beam's longitudinal axis. Off axis load must be broken into its sub-components (axial, shear, moment, etc.)

7. Zhuravskii shear stress

For shear in open sections (I-beams, C beams, L beams, and T beams) shear stress is computed using Zhuravskii shear stress formula shown below where V is shear force, Q is the first moment of area I is the moment of inertia and t is thickness [56].

$$\tau = \frac{VQ}{2I * t} \tag{6}$$

8. Thin Wall Shear

For thin-walled sections such as CHS and PIPE beams the thin-wall shear flow formula is used shown below where V is the shear force and A is the cross-sectional area of the beam [56].

$$\tau = 2\frac{V}{A} \tag{7}$$

9. Euler-Bernoulli Bending

Bending stress is calculated using the simplified Euler-Bernoulli algebraic equation shown below where M is the moment, Y is the distance to the neutral axis, and I is the moment of inertia [56].

$$\sigma_{bx} = \frac{M * Y}{I} \tag{8}$$

The AISC database provides S_x and S_y which is I/Y, so the equation above is simplified to the equation below.

$$\sigma_{bx} = \frac{M}{S} \tag{9}$$

10. St. Venant Torsion

Torsional shear stresses are computed using the St. Venant torsion theory shown in the equation below where T is the applied torque, t is thickness and J is the torsional constant [56].

$$\tau_{t,web} = \frac{T * t_w}{J} \tag{10}$$

11. Symmetry Considerations

For symmetrical beams, the symmetry is considered when using the above equations. For calculating max shear in the flange, the thickness of the flange is doubled because there are two flanges.

12. Von Mises Combined Stress

Stresses calculated by the above equations are combined using the von mises criterion. The max stress is calculated at two points for each beam (except PIPE and CHS beams) where stress is combined at the flange, web or individual legs or walls. The equation for von mises is shown below [56].

$$\sigma_{vm,flange} = \sqrt{(\sigma_{Axial} + \sigma_{bendX} + \sigma_{bendY})^2 + 3(\tau_{shear}^2 + \tau_{torsion}^2)} \qquad (11)$$

This assumes a ductile failure mode and a uniform stress at the evaluated point. This equation does not consider exact location of maximum stress so the final stress values should only be used as approximations and then be confirmed with FEA.

13. Design Intent

The beam calculator is intended to be used to find a preliminary design for beams and to give the user a better-informed initial decision on what beam to use. It is designed to be fast but not accurate, so users are expected to verify and validate their designs with proper FEA and testing analysis.

5.8.2 H & I Beams (W, M, S, HP)

The first and most popular beam that this calculator contains are the H and I beams. This includes wide flange (W) beams, American standard (S) beams, H-pile (HP) beams and miscellaneous I beams (M). To approximate the stress on an I beam two points were chosen to evaluate the stress at: the flange and the web. The figure below shows the I beam cross section. T is the length of the inner web, bf is the width of the beam, d is the height of the beam, tw is the thickness of the web and tf is the thickness of the flange. To calculate the stress of the beam the total stress of the flange and web will be calculated individually. But there are limitations to this method because it is assumed that force and stress are homogenous across a certain point, so the results of this beam calculator are only approximations to get the user a preliminary design.

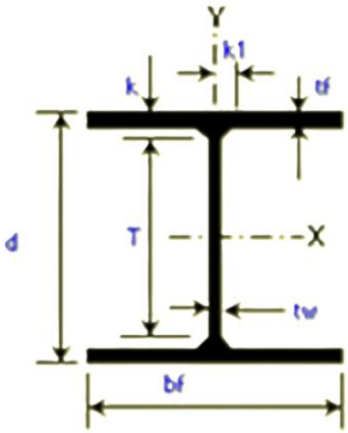


Figure 25 I beam profile [55]

The equations for calculating stresses on the different points on this beam follow the basic algebraic equations found in the mechanics of materials textbook written by James Gere and Berry Goodno [56]. Whenever applicable, geometric properties such as area, moment of inertia, and section modulus are taken from the AISC steel shape database [55] in applied to these equations. All units are imperial (inches, psi, etc.). The first equation below calculates the axial stresses where F_A is the axial force (lbs) and A is the cross sections area (in^2).

$$\sigma_A = \frac{F_A}{A} \tag{12}$$

The shear for both the X and Y direction are calculated using the first moment of area (Q) given by the AISC database [55]. The equation below is the shear calculation for the X direction in the flange where V_x is the shear force (lbs), Q_f is the first moment of area for one flange about the neutral axis (in^3), I_y is the moment of inertia bout the y axis (in^4) and t_f is thickness of the flange (in). Since I and H beams are symmetrical the total stress is then divided by 2 because there are two flanges.

$$\tau_{flange} = \frac{V_x Q_f}{2 I_y t_f} \tag{13}$$

The equation below is for calculating shear in the Y direction. Vy is the shear force (lbs), Qw is the first moment of area for a point in the center of the web (in^3), I_x is the moment of inertia about the x axis (in^4) and t_w is thickness of the web (in).

$$\tau_{web} = \frac{V_y Q_w}{I_x t_w} \tag{14}$$

To calculate stress due to bending the Euler Bernoulli beam equation is used [56]. The equation below is the calculation for stress created by a moment around the X axis. M is the applied moment (in-lbf) about the X axis and S_x (in^3) is the section modulus provided by the AISC database [55].

$$\sigma_{bx} = \frac{M_x}{S_x} \tag{15}$$

Similarly to the equation above, the stress created by a moment around the Y axis is also calculated using the Euler Bernoulli equation [56]. The equation below calculates the bending stress created by a moment where M is the applied moment (in-lbf) about the Y axis and S_y (in^3) is the section modulus provided by the AISC database [55].

$$\sigma_{by} = \frac{M_y}{S_y} \tag{16}$$

Since I beams are an open shape, torsion is calculated using the St. Venant expression [56] with the torsional constants provided by the AISC database [55]. It is assumed that warping effects are negligible. The equation below calculates torsion in the flange of the I beam where T is the applied Torque (in-lbf), t_f is the thickness of the flange (in) and J is the torsion constant (in^4) provided by the AISC database [55].

$$\tau_{t,flange} = \frac{T * t_f}{J} \tag{17}$$

The same equation can be applied to the web of the I beam using a similar equation based around the St. Venant equation [56]. T is the applied torque (in-lbf), t_w is the thickness of the web (in) and J is the torsion constant (in^4) again provided by the AISC database [55].

$$\tau_{t,web} = \frac{T * t_w}{J} \tag{18}$$

To combine the 6 stresses mentioned above the flange and the web will be considered. To combine stresses the von mises criteria will be used [56]. Starting with the outer flange, the equation is shown below where σ_A is axial stress (psi), σ_{bx} is bending stress about the x axis (psi), σ_{by} is bending stress about the y axis (psi), τ_{flange} is shear stress in the flange and $\tau_{t,flange}$ is torsional stress in the flange.

$$\sigma_{vm,flange} = \sqrt{(\sigma_A + \sigma_{bx} + \sigma_{by})^2 + 3(\tau_{flange}^2 + \tau_{t,flange}^2)} \quad (19)$$

The combined stresses at the center of the web are calculated using the equation below. σ_A is axial stress (psi), σ_{bx} is bending stress about the x axis (psi), σ_{by} is bending stress about the y axis (psi), τ_{web} is shear stress in the web and $\tau_{t,web}$ is torsional stress in the web.

$$\sigma_{vm,web} = \sqrt{(\sigma_A + \sigma_{bx} + \sigma_{by})^2 + 3(\tau_{web}^2 + \tau_{t,web}^2)} \quad (20)$$

Once all combined stresses are calculated the margin of safety can be calculated. To accomplish this the largest stress at one of the two points is plugged into the equation below where $\sigma_{allowable}$ is the maximum stress that the beam is allowed to experience. This is calculated by dividing the yield and ultimate stress of a chosen material by safety factors and then using the smallest number between yield and ultimate. $\sigma_{predicted}$ is the largest stress of the two points. A value zero means that the beam meets the criteria without any room to spare. A value less than zero means that the beam did not meet the criteria. Finally, a value larger than 0 means the beam met the criteria with room to spare.

$$MS = \frac{\sigma_{allowable}}{\sigma_{predicted}} - 1 \quad (21)$$

5.8.3 C Beams (C, MC)

The second beam offered in the beam calculator is the C beam which includes American standard C beams and miscellaneous MC beams. Similar to the I beam the calculator will calculate stress on the web and flange. The profile of the beam can be seen in the figure below where d is the overall height of beam, bf is the width of the flange, tf is the thickness of the flange, tw is the thickness of the web and x(bar) is the distance from the left edge to the centroid.

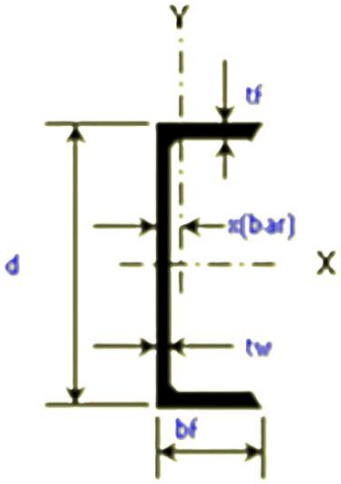


Figure 26 C beam profile [55]

The same equations are used on the C beam provided by James Gere and Berry Goodno [56]. The first equation below calculates the axial stresses where F_A is the axial force (lbs) and A is the cross sections area (in^2).

$$\sigma_A = \frac{F_A}{A} \tag{22}$$

The shear for both the X and Y direction are calculated using the first moment of area (Q) given by the AISC database [54]. The equation below is the shear calculation for the X direction in the flange where V_x is the shear force (lbs), Q_f is the first moment of area for the flange about the neutral axis (in^3), I_y is the moment of inertia about the x axis (in^4) and t_f is thickness of the flange (in). The flange shear stress is halved to approximate load sharing between the two flange plates.

$$\tau_x = \frac{V_x Q_f}{2 I_y t_f} \tag{23}$$

The equation below is for calculating shear in the Y direction. Vy is the shear force (lbs), Qw is the first moment of area for a point in the center of the web (in^3), I_x is the moment of inertia about the x axis (in^4) and t_w is thickness of the web (in).

$$\tau_y = \frac{V_y Q_w}{I_x t_w} \tag{24}$$

To calculate stress due to a bending moment the Euler Bernoulli beam equation is used [55]. The equation below is the calculation for stress created by a moment

around the X axis. M is the applied moment (in-lbf) about the X axis and S_x (in^3) is the section modulus provided by the AISC database [55].

$$\sigma_{bx} = \frac{M_x}{S_x} \tag{25}$$

Similarly to the equation above, the stress created by a moment around the Y axis is also calculated using the Euler Bernoulli equation [55]. The equation below calculates the bending stress created by a moment where M is the applied moment (in-lbf) about the Y axis and S_y (in^3) is the section modulus provided by the AISC database [55].

$$\sigma_{by} = \frac{M_y}{S_y} \tag{26}$$

Just like I beams, C beams are an open shape, so torsion is calculated using the St. Venant expression [55] with the torsional constants provided by the AISC database [54]. The equation below calculates torsion in the flange of the C beam where T is the applied Torque (in-lbf), t_f is the thickness of the flange (in) and J is the torsion constant (in^4) provided by the AISC database [54].

$$\tau_{flange} = \frac{T * t_f}{J} \tag{27}$$

The same equation can be applied to the web of the I beam using a similar equation based around the St. Venant equation [55]. T is the applied torque (in-lbf), t_w is the thickness of the web (in) and J is the torsion constant (in^4) again provided by the AISC database [54].

$$\tau_{web} = \frac{T * t_w}{J} \tag{28}$$

Like the I beams, to combine the 6 stresses mentioned above the flange and web will be used. To combine stresses the von mises criteria will be used [55]. Starting with the flange, the equation is shown below where σ_A is axial stress (psi), σ_{bx} is bending stress about the x axis (psi), σ_{by} is bending stress about the y axis (psi), τ_{flange} is shear stress in the flange and $\tau_{t,flange}$ is torsional stress in the flange.

$$\sigma_{vm,flange} = \sqrt{(\sigma_A + \sigma_{bx} + \sigma_{by})^2 + 3(\tau_{flange}^2 + \tau_{t,flange}^2)} \tag{29}$$

The combined stresses at the center of the web are calculated using the equation below. σ_A is the axial stress (psi), σ_{bx} is bending stress about the x axis (psi), σ_{by} is bending stress about the y axis (psi), τ_{web} is shear stress in the web and $\tau_{t,web}$ is torsional stress in the web.

$$\sigma_{vm,web} = \sqrt{(\sigma_A + \sigma_{bx} + \sigma_{by})^2 + 3(\tau_{web}^2 + \tau_{t,web}^2)} \quad (30)$$

Once all combined stresses are calculated the margin of safety can be calculated. To do this the largest stress at one of the two points is plugged into the equation below. $\sigma_{allowable}$ is the maximum stress that the beam is allowed to experience. This is calculated by dividing the yield and ultimate stress of a chosen material by safety factors and then using the smallest number between yield and ultimate. $\sigma_{predicted}$ is the largest stress of the two points. A value zero means that the beam meets the criteria without any room to spare. A value less than zero means that the beam did not meet the criteria. Finally, a value larger than 0 means the beam met the criteria with room to spare.

$$MS = \frac{\sigma_{allowable}}{\sigma_{predicted}} - 1 \quad (31)$$

5.8.4 L Beams

Another beam in the calculator is the L beam. Unlike the previous beams the stress will be measured at the 2 legs of the beam. The profile of the beam can be seen in the figure below where b is the length of the first leg, d is the length of the second leg, t is the thickness of the legs and x (bar) and y (bar) are the distances from the edge to the centroid in the x and y direction respectively.

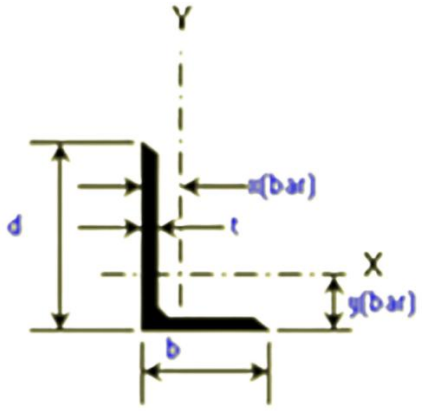


Figure 27 L beam profile [55]

The same equations as the previous beams are used on the L beam provided by James Gere and Berry Goodno [55]. The first equation below calculates the axial stresses where F_A is the axial force (lbs) and A is the cross sections area (in^2).

$$\sigma_A = \frac{F_A}{A} \tag{32}$$

For calculating shear in the L beams an interesting but conservative approach was taken. The equation below us used for calculating shear in thin rectangular members. Each leg is treated as a thin rectangular member where V_x is the shear force (lbf), b is the length of the first leg (in) and t is the thickness (in).

$$\tau_x = \frac{3}{2}\frac{V_x}{b * t} \tag{33}$$

The same approach is used for the other leg where it is treated like a thin rectangular member. The equation below calculates shear stress where V_y is the shear force from the y direction (lbf), d is the length of the second leg (in) and t is the thickness (in).

$$\tau_y = \frac{3}{2}\frac{V_y}{d * t} \tag{34}$$

To calculate stress due to bending the Euler Bernoulli beam equation is used [55]. The equation below is the calculation for stress created by a moment around the X axis. M is the applied moment (in-lbf) about the X axis and S_x (in^3) is the section modulus provided by the AISC database [54].

$$\sigma_{bx} = \frac{M_x}{S_x} \tag{35}$$

Similarly to the equation above, the stress created by a moment around the Y axis is also calculated using the Euler Bernoulli equation [55]. The equation below calculates the bending stress created by a moment where M is the applied moment (in-lbf) about the Y axis and S_y (in^3) is the section modulus provided by the AISC database [54].

$$\sigma_{by} = \frac{M_y}{S_y} \tag{36}$$

Just like the previous beams L beams are an open shape, so torsion is calculated using the St. Venant expression [55] with the torsional constants provided by the AISC

database [54]. The equation below calculates torsion in the flange of the L beam where T is the applied Torque (in-lbf), t is the thickness of the leg (in) and J is the torsion constant (in^4) provided by the AISC database [54].

$$\tau_t = \frac{T * t}{J} \tag{37}$$

Like the previous beams, to combine the 6 stresses mentioned above the von mises criteria will be used [55]. Starting with leg1, the equation is shown below where σ_A is axial stress (psi), σ_{bx} is bending stress about the x axis (psi), σ_{by} is bending stress about the y axis (psi), τ_x is shear stress in the x direction τ_t is torsional stress.

$$\sigma_{vm,leg1} = \sqrt{(\sigma_A + \sigma_{bx} + \sigma_{by})^2 + 3(\tau_x^2 + \tau_t^2)} \tag{38}$$

The combined stresses in the second leg are calculated using the equation below. σ_A is the axial stress (psi), σ_{bx} is bending stress about the x axis (psi), σ_{by} is bending stress about the y axis (psi), τ_y is shear stress from the y direction and τ_t is torsional stress.

$$\sigma_{vm,leg2} = \sqrt{(\sigma_A + \sigma_{bx} + \sigma_{by})^2 + 3(\tau_y^2 + \tau_t^2)} \tag{39}$$

Once all combined stresses are calculated the margin of safety can be calculated. To do this the largest stress at one of the two points is plugged into the equation below. $\sigma_{allowable}$ is the maximum stress that the beam is allowed to experience. This is calculated by dividing the yield and ultimate stress of a chosen material by safety factors and then using the smallest number between yield and ultimate. $\sigma_{predicted}$ is the largest stress of the two points. A value zero means that the beam meets the criteria without any room to spare. A value less than zero means that the beam did not meet the criteria. Finally, a value larger than 0 means the beam met the criteria with room to spare.

$$MS = \frac{\sigma_{allowable}}{\sigma_{predicted}} - 1 \tag{40}$$

5.8.5 T Beams (WT, MT)

The next beam available on the calculator is the T beam which includes the WT beam which is made from cutting a W I beam in half and a MT beam which are miscellaneous T beams. The profile of the beam can be seen in the figure below

where bf is the width of the beam, d is the height of the beam, tf is the thickness of the flange, tw is the thickness of the web y (bar) is the distance from the top edge of the flange to the centroid in the y direction.

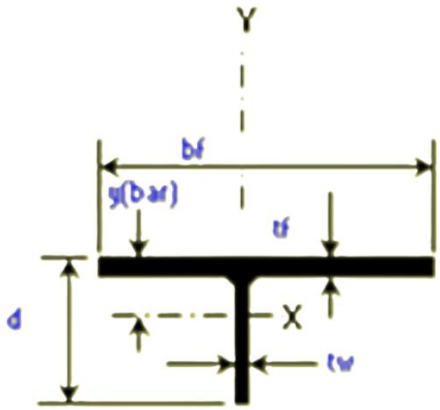


Figure 28 T beam profile [55]

The same equations as the previous beams are used on the T beam provided by James Gere and Berry Goodno [55]. The first equation below calculates the axial stresses where F_A is the axial force (lbs) and A is the cross sections area (in^2).

$$\sigma_A = \frac{F_A}{A} \tag{41}$$

Similar to the L beam a conservative approach was taken where the flange is approximated as a thin rectangle. The equation below us used for calculating shear in thin rectangular members where V_x is the shear force (lbf), b_f is the length of the flange and t_f is the flange thickness (in).

$$\tau_x = \frac{3}{2}\frac{V_x}{b_f t_f} \tag{42}$$

The same approach is taken for the web of the T beam where it is treated as a thin rectangle. The equation shows this below where V_y is the shear force in the y direction (lbf), t_w is the thickness of the web and then (d-t_f) is the length of the web where d is the height of the entire beam and t_f is the thickness of the flange.

$$\tau_y = \frac{3}{2}\frac{V_y}{(d - t_f)t_w} \tag{43}$$

To calculate stress due to bending the Euler Bernoulli beam equation is used [55]. The equation below is the calculation for stress created by a moment around the X axis. M is the applied moment (in-lbf) about the X axis and S_x (in^3) is the section modulus provided by the AISC database [54].

$$\sigma_{bx} = \frac{M_x}{S_x} \tag{44}$$

Similarly to the equation above, the stress created by a moment around the Y axis is also calculated using the Euler Bernoulli equation [55]. The equation below calculates the bending stress created by a moment where M is the applied moment (in-lbf) about the Y axis and S_y (in^3) is the section modulus provided by the AISC database [54].

$$\sigma_{by} = \frac{M_y}{S_y} \tag{45}$$

Since T beams are an open shape, torsion is calculated using the St. Venant expression [55] with the torsional constants provided by the AISC database [54]. It is assumed that warping effects are negligible. The equation below calculates torsion in the flange of the T beam where T is the applied Torque (in-lbf), t_f is the thickness of the flange (in) and J is the torsion constant (in^4) provided by the AISC database [54].

$$\tau_{t,flange} = \frac{T * t_f}{J} \tag{46}$$

The same equation can be applied to the web of the T beam using a similar equation based around the St. Venant equation [55]. T is the applied torque (in-lbf), t_w is the thickness of the web (in) and J is the torsion constant (in^4).

$$\tau_{t,web} = \frac{T * t_w}{J} \tag{47}$$

To combine the 6 stresses mentioned above the same approach is used as the I beam where stresses in the flange and the web will be considered. To combine stresses the von mises criteria will be used [55]. Starting with the outer flange, the equation is shown below where σ_A is axial stress (psi), σ_{bx} is bending stress about the x axis (psi), σ_{by} is bending stress about the y axis (psi), τ_{flange} is shear stress in the flange and $\tau_{t,flange}$ is torsional stress in the flange.

$$\sigma_{vm,flange} = \sqrt{(\sigma_A + \sigma_{bx} + \sigma_{by})^2 + 3(\tau_{flange}^2 + \tau_{t,flange}^2)} \quad (48)$$

The combined stresses at the center of the web are calculated using the equation below. σ_A is axial stress (psi), σ_{bx} is bending stress about the x axis (psi), σ_{by} is bending stress about the y axis (psi), τ_{web} is shear stress in the web and $\tau_{t,web}$ is torsional stress in the web.

$$\sigma_{vm,web} = \sqrt{(\sigma_A + \sigma_{bx} + \sigma_{by})^2 + 3(\tau_{web}^2 + \tau_{t,web}^2)} \quad (49)$$

Once all combined stresses are calculated the margin of safety can be calculated. To do this the largest stress at one of the two points is plugged into the equation below. $\sigma_{allowable}$ is the maximum stress that the beam is allowed to experience. This is calculated by dividing the yield and ultimate stress of a chosen material by safety factors and then using the smallest number between yield and ultimate. $\sigma_{predicted}$ is the largest stress of the two points. A value zero means that the beam meets the criteria without any room to spare. A value less than zero means that the beam did not meet the criteria. Finally, a value larger than 0 means the beam met the criteria with room to spare.

$$MS = \frac{\sigma_{allowable}}{\sigma_{predicted}} - 1 \quad (50)$$

5.8.6 Hollow Beams (HSS)

A very important beam of the calculator is the Hollow Structural Section (HSS) beam. The profile can be seen below where b is the width of the beam, h is the height of the beam and t_{des} is the thickness of the wall of the beam.

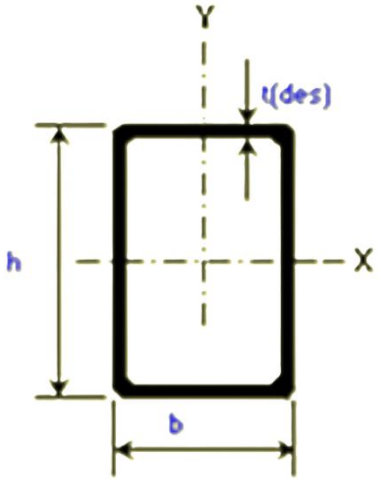


Figure 29 HSS beam profile [55]

The same equations as the previous beams are used on the T beam provided by James Gere and Berry Goodno [55]. The first equation below calculates the axial stresses where F_A is the axial force (lbs) and A is the cross sections area (in^2).

$$\sigma_A = \frac{F_A}{A} \tag{51}$$

To find shear in the HSS beam the average of the two walls shear is used [56]. Since the AISC database does not provide Q or other important information for calculating shear the area of the two walls must be approximated. The overall width of the wall is calculated by taking the total width of the beam and subtracting the thickness of the orthogonal walls. The remaining value is then multiplied by the thickness of the wall and then since there are two walls that value is multiplied by 2. The resulting equation is shown below where V_x is the shear force in the x direction (lbf), t_{des} is the thickness of the wall (in) and b is the overall width of the beam (in).

$$\tau_x = \frac{V_x}{2t_{des}(b - 2t_{des})} \tag{52}$$

The same idea is used for shear in the Y direction. The equation is shown below where V_y is the shear force in the Y direction (lbf), t_{des} is the thickness of the wall (in) and h is the overall height of the beam (in).

$$\tau_y = \frac{V_y}{2t_{des}(h - 2t_{des})} \tag{53}$$

To calculate stress due to bending the Euler Bernoulli beam equation is used [55]. The equation below is the calculation for stress created by a moment around the X axis. M is the applied moment (in-lbf) about the X axis and S_x (in^3) is the section modulus provided by the AISC database [54].

$$\sigma_{bx} = \frac{M_x}{S_x} \tag{54}$$

Similarly to the equation above, the stress created by a moment around the Y axis is also calculated using the Euler Bernoulli equation [55]. The equation below calculates the bending stress created by a moment where M is the applied moment (in-lbf) about the Y axis and S_y (in^3) is the section modulus provided by the AISC database [54].

$$\sigma_{by} = \frac{M_y}{S_y} \tag{55}$$

For torsional stress, the equation for torsion in a thin-walled tube is used. To do this the torque is divided by the mean area of the tube times two times the thickness [56]. The equation below shows this where T is the torque (in-lbf), h is the height of the beam (in), b is the base or width of the beam (in) and t is the thickness of the wall (in).

$$\tau_t = \frac{T}{2(h - t_{des})(b - t_{des})t_{des}} \tag{56}$$

To combine the 6 stresses mentioned above the von mises criteria will be used [55]. Starting with wall X, the equation is shown below where σ_A is axial stress (psi), σ_{bx} is bending stress about the x axis (psi), σ_{by} is bending stress about the y axis (psi), τ_x is shear stress in wall X and τ_t is the torsional stress.

$$\sigma_{vm,x} = \sqrt{(\sigma_A + \sigma_{bx} + \sigma_{by})^2 + 3(\tau_x^2 + \tau_t^2)} \tag{57}$$

The same idea is applied to the Y wall. The equation is shown below where σ_A is axial stress (psi), σ_{bx} is bending stress about the x axis (psi), σ_{by} is bending stress about the y axis (psi), τ_y is shear stress in wall Y and τ_t is the torsional stress.

$$\sigma_{vm,y} = \sqrt{(\sigma_A + \sigma_{bx} + \sigma_{by})^2 + 3(\tau_y^2 + \tau_t^2)} \tag{58}$$

Once all combined stresses are calculated the margin of safety can be calculated. To do this the largest stress at one of the two points is plugged into the equation below. $\sigma_{allowable}$ is the maximum stress that the beam is allowed to experience. This is calculated by dividing the yield and ultimate stress of a chosen material by safety factors and then using the smallest number between yield and ultimate. $\sigma_{predicted}$ is the largest stress of the two points. A value zero means that the beam meets the criteria without any room to spare. A value less than zero means that the beam did not meet the criteria. Finally, a value larger than 0 means the beam met the criteria with room to spare.

$$MS = \frac{\sigma_{allowable}}{\sigma_{predicted}} - 1 \tag{59}$$

5.8.7 Round Tubing (CHS, PIPE)

The final beam offered in the calculator is round tubing which consists of circular hollow tubing (CHS) and pipes. The profile for the beam is located in the figure below where OD is the outer diameter, ID is the inner diameter and t_{nom} is the thickness of the pipe wall.

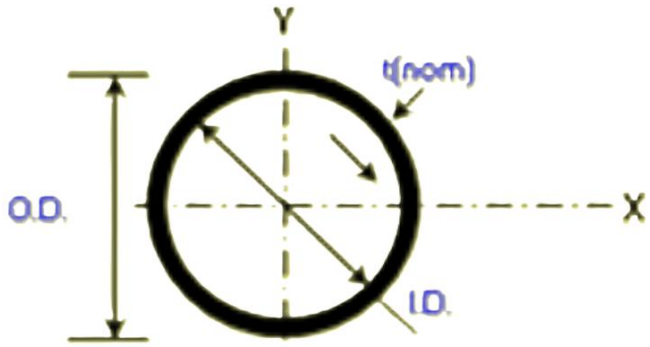


Figure 30 Round tubing profile [55]

To solve the stresses of a circular beam a different approach is taken when compared to the other beams. Instead of solving stress in each direction separately then

combining them with von mises the forces are combined into one equivalent force and then stress is calculated. For axial stress however, the same equation as the previous beams are used where F_A is the axial force (lbs) and A is the cross sections area (in^2).

$$\sigma_A = \frac{F_A}{A} \tag{60}$$

The equation below is used for calculating shear stress. Where V_x is shear in the x direction (lbf), V_y is shear force in the y direction and they are combined to create a resulting vector. This is then divided by A which is the area of the thin wall (in^2).

$$\tau_x = 2\frac{\sqrt{V_x^2 + V_y^2}}{A} \tag{61}$$

The moment is calculated in a comparable manner where an equivalent force vector is calculated using V_x and V_y which is force in the x and y direction respectively (lbf). This is then divided by S_x which is the section modulus provided by the AISC database [54].

$$\sigma_b = \frac{\sqrt{V_x^2 + V_y^2}}{S_x} \tag{62}$$

Torsional stress is calculated by multiplying the torque by the radius and then dividing by the torsional constant. This is shown in the equation below where T is torque (in-lbf) OD is the outer diameter (in) and J is the torsion constant (in^4)

$$\tau_t = \frac{T * {OD}/{2}}{J} \tag{63}$$

To combine the 4 stresses mentioned above the von mises criteria will be used [55]. The equation is shown below where σ_A is axial stress (psi), σ_{bx} is bending stress (psi), τ_x is shear stress (psi) and τ_t is the torsional stress (psi).

$$\sigma_{vm,x} = \sqrt{(\sigma_A + \sigma_b)^2 + 3(\tau_x^2 + \tau_t^2)} \tag{64}$$

Once the combined stress is calculated it is plugged into the equation below where $\sigma_{allowable}$ is the maximum stress that the beam is allowed to experience. This is calculated by dividing the yield and ultimate stress of a chosen material by safety factors and then using the smallest number between yield and ultimate. $\sigma_{predicted}$ is the largest stress of the two points. A value zero means that the beam meets the criteria without any room to spare. A value less than zero means that the beam did not meet the criteria. Finally, a value larger than 0 means the beam met the criteria with room to spare.

$$MS = \frac{\sigma_{allowable}}{\sigma_{predicted}} - 1 \tag{65}$$

5.9 ODrive Motor Controller

Another feature of the engineering tool is the ODrive motor controller. ODrive is a versatile brushless dc motor controller that is very intuitive and easy to use [53]. This controller can be used in many programming languages such as python and Ros. However, there is not a library or any other system that supports ODrive control in LabView. LabVIEW is not the first language that comes to mind when controlling a BLDC with an ODrive. But there are a lot of mechanical engineers that cannot use python but use LabView so a system that allows easy control of a BLDC motor through ODrive in LabVIEW is necessary.

The controller can send commands to an ODrive via USB to control a BLDC motor with either velocity, torque or position control. The controller is also able to record voltage, current, power, velocity, and position data at 70Hz which is then automatically saved to a CSV file for later examination. The tool also contains multiple safety features such as a STOP button which stops everything and acts as a virtual E-stop button. The tool also contains an initialize button which runs commands to connect to the ODRIVE over USB serial. The enable button then starts running commands and reading data. The motor on/off button enables the motor to run and will run based on the velocity, torque or position setpoints given. If any of these buttons are pressed again putting them in the off state the motor will stop running.

The main front panel of the ODrive controller is shown in the figure below.

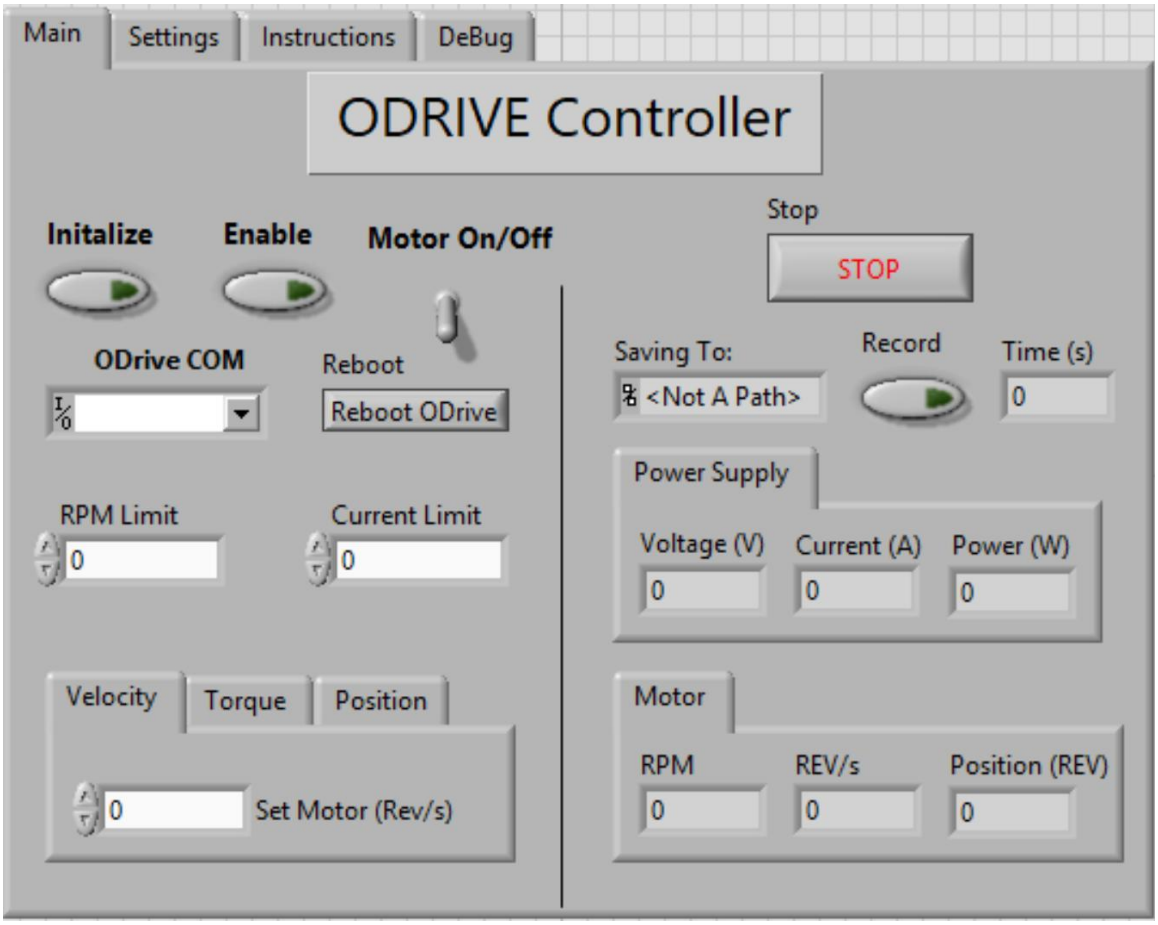


Figure 31 LabVIEW ODrive controller front panel

Since LabView is not an official language supported by ODrive unlike Python, C or ROS [53] commands must be sent to the ODRIVE using a serial connection. For this tool a simple USB cable is used. The architecture for sending commands over the USB is based on a VISA system which was inspired by a small project by Stijns [57] which controls an ODrive using LabVIEW but only works for one specific motor and is extremely limited in capability relying on potentiometers and loadcells. The other main issue with this code is that it does not record data and has an insanely high delay [57].

The controller presented in this thesis takes inspiration from the only other existing LabVIEW based ODrive controller and improves on it. The figure below shows the first loop of the LabVIEW controller which connects to the ODrive via USB and then sends setting commands such as PID values and maximum current and velocity values.

After one loop is completed the while loop enters a standby state until the enable button is pressed.

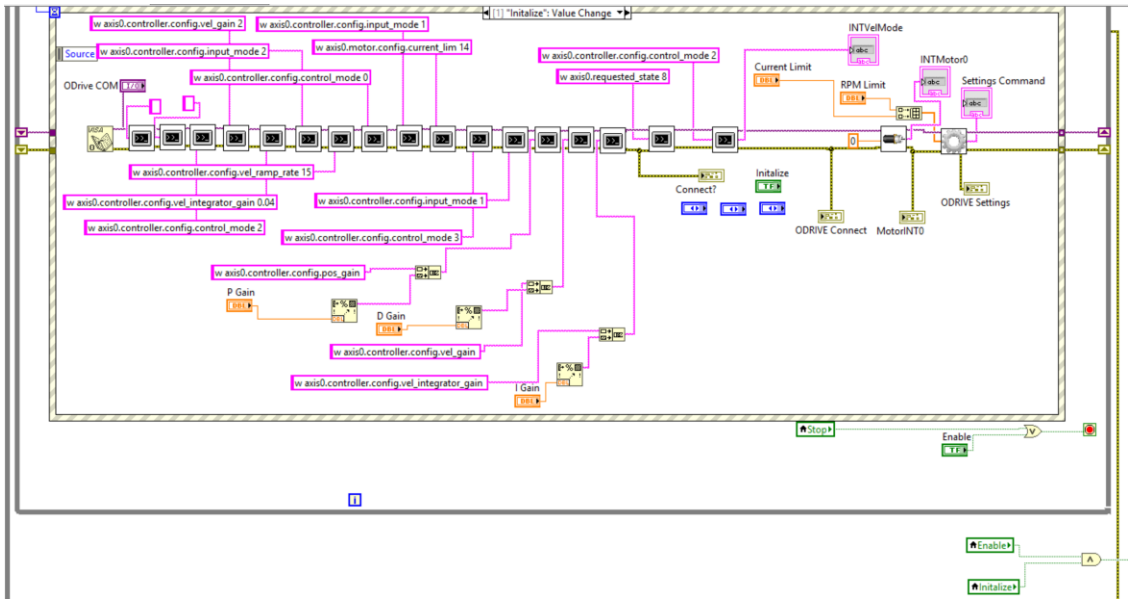


Figure 32 LabVIEW ODrive initialization loop

Then the data collection loop begins which can be seen in the figure below. Since LabVIEW is not a supported language the only way to get information from the ODrive such as voltage the program must send a command asking for it and then wait a couple of milliseconds before reading the response. There are four total values that are collected: voltage, current, velocity and position. These values are then used to calculate power and RPM which are displayed in real time on the front panel. Once the record button is pressed the data is saved to a CSV file. This loop will run until the program ends. Initialization

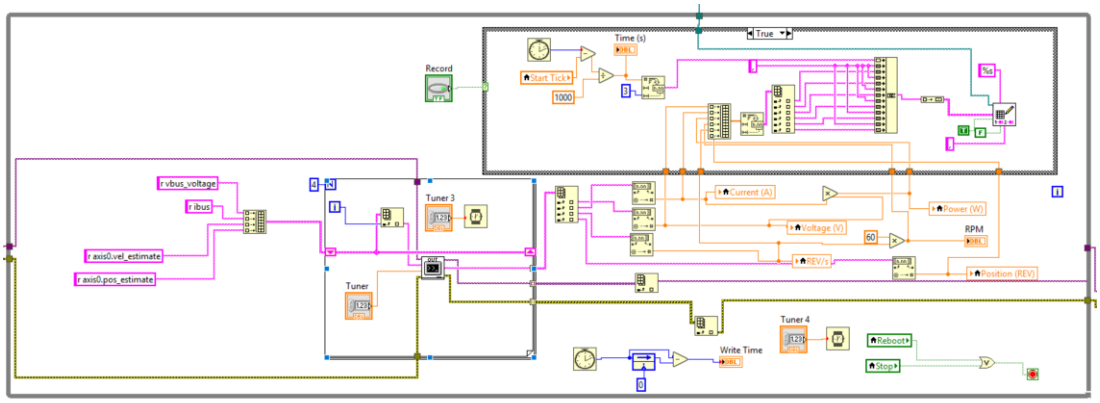


Figure 33 LabVIEW data collection and recording loop

Running in parallel to the data collection loop is the motor control loop. This loop contains three motor control options (velocity, torque, position). In the figure shown below is the velocity controller. This system works by setting the ODrive to an active state then setting it to a velocity control mode and then sending the velocity value. The control loop can be seen in the figure below.

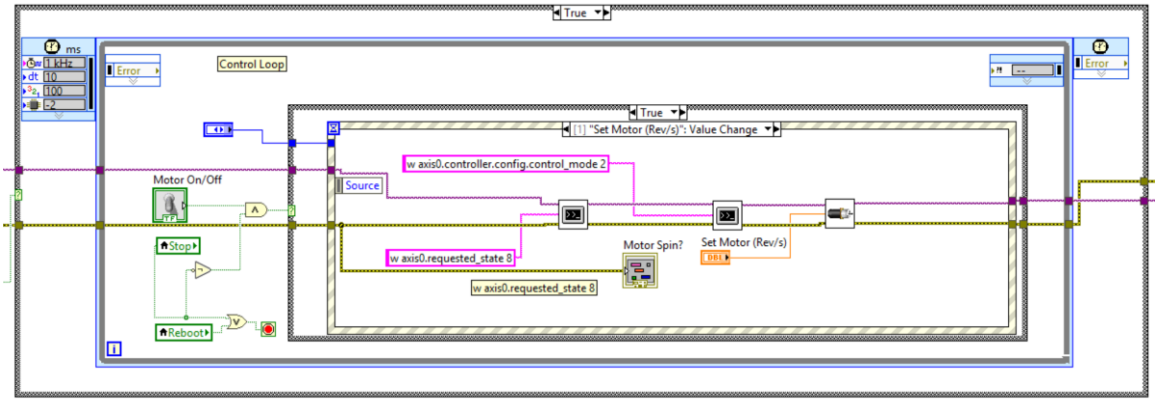


Figure 34 LabVIEW ODrive control loop

This system is dangerous because once a command is sent to the ODrive the BLDC will spin until a command to stop it is sent. This means that if connection to the ODrive is lost the motor will continue to spin until power is cut. This is why many safety features are added to the system including a watchdog which is a system built into ODrive where if a command is not received within a certain time the ODrive automatically cuts power. The LabVIEW script also contains multiple scripts that stop the motor if a fault happens. This system can be seen below which is placed at the end of every loop. The system sets the speed of the motor to zero and then disables the ODrive and then shuts down the VISA serial connection.

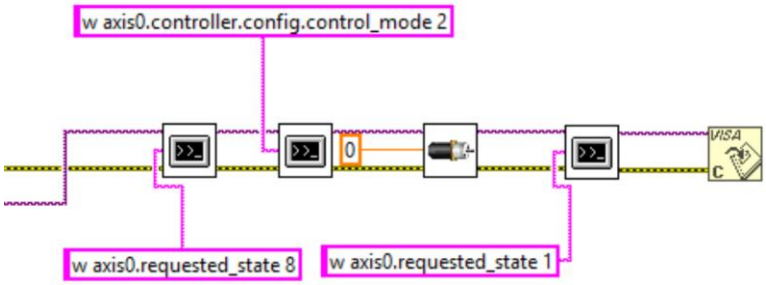


Figure 35 LabVIEW ODrive shut down script

6 System Verification

In engineering verification is a process that ensures a design solves a problem or model correctly [3]. The purpose of this section is to verify the beam calculator developed in this thesis. Verification ensures that the stress values produced by the calculator match the governing equations of mechanics of materials. The calculator is also compared with finite element analysis (FEA) to evaluate whether its predictions are conservative and to quantify differences between analytical and numerical results.

6.1 Beam Calculator Verification

To verify the beam calculator, examples of the equations used are worked by hand and then are compared to the beam calculator. This ensures that the equations are being solved correctly within the beam calculator.

6.1.1 I Beam Equation Verification

The first beam used to verify the equations is the W4X13 I beam shown in the figure below. The properties for this beam used in the equations are shown in the table below that. The loads used are shown in the table below.

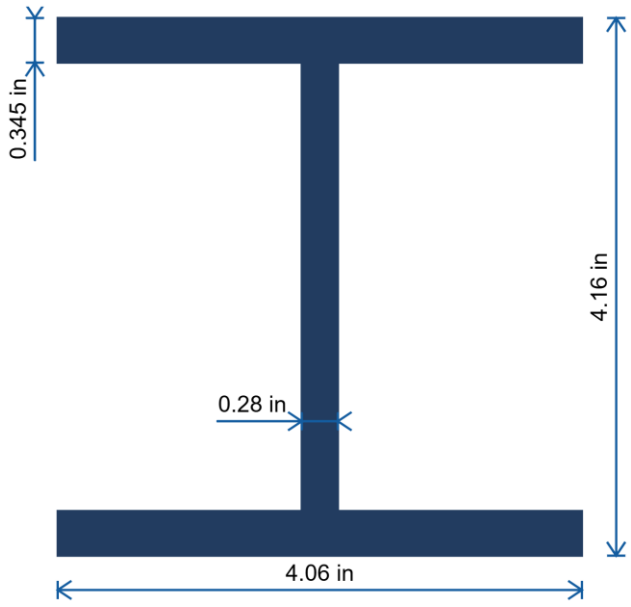


Figure 36 W4X13 I beam [55]

Table 4 W4X13 properties [55]

A (in^2)	Q_f (in^3)	I_y (in^4)	t_f (in)	Q_w (in^3)	I_x (in^4)	t_w (in)	S_x (in^3)	S_y (in^3)	J (in^4)
3.83	1.24	3.86	0.345	3.09	11.3	0.28	5.46	1.9	0.151

Table 5 Loads used

Axial (lbf)	X Shear (lbf)	Y Shear (lbf)	X moment (in-lbf)	Y moment (in-lbf)	Torsion (in-lbf)
11240	11240	11240	11240	11240	11240

The equation below is calculating axial stress which matches the calculator's calculated axial stress of 2934.73psi.

$$\sigma_A = \frac{11240}{3.83} = 2934.73\ psi \tag{66}$$

The equation below is calculating shear stress in the flange which matches the calculator's calculated shear stress of 5233.01psi.

$$\tau_{flange} = \frac{11240 * 1.24}{2 * 3.86 * 0.345} = 5233.01\ psi \tag{67}$$

The equation below is calculating shear stress on the web which matches the calculator's shear stress of 10977.1psi.

$$\tau_{web} = \frac{11240 * 3.09}{11.3 * 0.28} = 10977.1 psi \tag{68}$$

The equation below calculates bending stress about the x axis which matches the calculator's stress of 2058.61psi.

$$\sigma_{bx} = \frac{11240}{5.46} = 2058.61 psi \tag{69}$$

The equation below calculates bending stress about the y axis which matches the calculator stress of 5915.79psi.

$$\sigma_{by} = \frac{11240}{1.9} = 5915.79 psi \tag{70}$$

The equation below calculates the torsional stress in the flange which matches the calculator stress of 25680.8psi.

$$\tau_{t,flange} = \frac{11240 * 0.345}{0.151} = 25680.8 psi \tag{71}$$

The equation below calculates the torsional stress in the web which matches the calculator stress of 20842.4psi.

$$\tau_{t,web} = \frac{11240 * 0.28}{0.151} = 20842.4psi \tag{72}$$

The equation below calculates the combined stress inf the flange using the von mises criteria matching the calculator stress of 46687psi.

$$\sigma_{vm,flange} = \sqrt{(2934.7 + 2058.6 + 5915.79)^2 + 3(25680.8^2 + 5233^2)} = 46687psi \tag{73}$$

The equation below calculates the combined stress in the web using the con mises criteria matching the calculator stress of 42234psi.

$$\sigma_{vm,web} = \sqrt{(2934.7 + 2058.6 + 5915.79)^2 + 3(10977^2 + 5233^2)} = 42234psi \tag{74}$$

The inputs and results from the beam calculator can be seen in the figures below.

Beam Calculator

Last Update: 11/5/2025

Inputs

	X	Y	
Applied Shear	11240	11240	lbs
Applied Moment	11240	11240	in-lbs
Applied Axial	11240		lbs
Applied Torsion	11240		in-lbs
Ultimate Stress	60000		psi
Yield Stress	46000		psi
FSU	3		
FSY	2		
Ult Allowed	20000		psi
Yield Allowed	23000		psi
Beam Type	W		

Beam Options:
W
M
S
HP
C
MC
L
WT
MT
HSS
CHS
PIPE

Figure 37 Beam calculator inputs for I beam

Axial	Bending Stress		Shear Stress		Torsional Stress		Combined Stress	
Stress	X-axis	Y-axis	X-axis	Y-axis	Flange	Web	Flange	Web
2934.73	2058.61	5915.79	5233.01	10977.12	25680.79	20842.38	46686.96	42234.05

Figure 38 Beam calculator outputs for I beam

6.1.2 C Beam Equation Verification

The second beam used to verify the equations is the C3X3.5 C beam shown in the figure below. The properties for this beam used in the equations are shown in the table below that. Then the loads used are shown in the table below that.

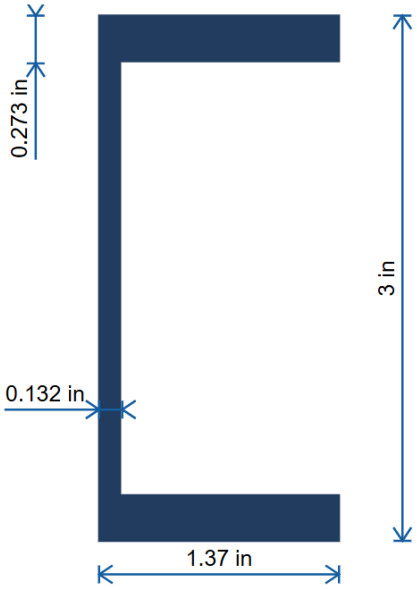


Figure 39 C3X3.5 C beam [55]

Table 6 C3X3.5 beam properties [55]

A (in^2)	Q_f (in^3)	I_y (in^4)	t_f (in)	Q_w (in^3)	I_x (in^4)	t_w (in)	S_x (in^3)	S_y (in^3)	J (in^4)
1.09	0.461	0.169	0.273	0.609	1.57	0.132	1.04	0.182	0.0226

Table 7 C3X3.5 beam loads

Axial (lbf)	X Shear (lbf)	Y Shear (lbf)	X moment (in-lbf)	Y moment (in-lbf)	Torsion (in-lbf)
11240	11240	11240	11240	11240	11240

The equation below calculates axial stress which matches the calculator's stress of 10311.9psi.

$$\sigma_A = \frac{11240}{1.09} = 10311.9psi \tag{75}$$

The equation below calculates shear stress in the x direction in the flange which matches the calculator's stress of 56154.9psi.

$$\tau_x = \frac{11240 * 0.461}{2 * 0.169 * 0.273} = 56154.9psi \quad (76)$$

The equation below calculates shear stress in the y direction in the web which matches the calculator's stress of 33030.1psi.

$$\tau_y = \frac{11240 * 0.609}{1.57 * 0.132} = 33030.1psi \quad (77)$$

The equation below calculates the stress due to bending about the x axis which matches the calculator's stress of 10807.7psi.

$$\sigma_{bx} = \frac{11240}{1.04} = 10807.7psi \quad (78)$$

The equation below calculates the stress due to bending about the y axis which matches the calculator's stress of 61758.2psi.

$$\sigma_{by} = \frac{11240}{0.182} = 61758.2psi \quad (79)$$

The equation below calculates the torsional stress in the flange which matches the calculator stress of 135775psi.

$$\tau_{t,flange} = \frac{11240 * 0.273}{0.0226} = 135775psi \quad (80)$$

The equation below calculates the torsional stress in the flange which matches the calculator stress of 65649.6psi.

$$\tau_{t,web} = \frac{11240 * 0.132}{0.0226} = 65649.6psi \quad (81)$$

The equation below calculates the combined stress in the flange using the von mises criteria matching the calculator stress of 267644.5psi.

$$\sigma_{vm,flange} = \sqrt{(10311.9 + 10807.7 + 61758)^2 + 3(56154^2 + 135775^2)} = 267644.5psi \quad (82)$$

The equation below calculates the combined stress in the web using the von mises criteria matching the calculator stress of 151892psi.

$$\sigma_{vm,web} = \sqrt{(10311.9 + 10807.7 + 61758)^2 + 3(33030^2 + 65649.6^2)} = 151892psi \quad (83)$$

The inputs and outputs to the beam calculator can be seen in the figures below and the outputs match the values calculated in the equations above.

Beam Calculator

Last Update: 11/5/2025

Inputs

	X	Y	
Applied Shear	11240	11240	lbs
Applied Moment	11240	11240	in-lbs
Applied Axial	11240		lbs
Applied Torsion	11240		in-lbs
Ultimate Stress	60000		psi
Yield Stress	46000		psi
FSU	3		
FSY	2		
Ult Allowed	20000		psi
Yield Allowed	23000		psi
Beam Type	C		

Beam Options:
W
M
S
HP
C
MC
L
WT
MT
HSS
CHS
PIPE

Figure 40 Beam calculator inputs for a C3X3.5 beam

Axial	Bending Stress		Shear Stress		Torsional Stress		Combined Stress	
Stress	X-axis	Y-axis	X-axis	Y-axis	Flange	Web	Flange	Web
10311.93	10807.69	61758.24	56154.93	33030.11	135775.22	65649.56	267644.54	151892.39

Figure 41 Beam calculator results for a C3X3.5 beam

6.1.3 L Beam Equation Verification

The L beam chosen to verify the calculator is the L2X2X1/8 beam shown in the figure below. The properties for this beam used in the equations are shown in the table below that. Then the loads used are shown in the table below that.

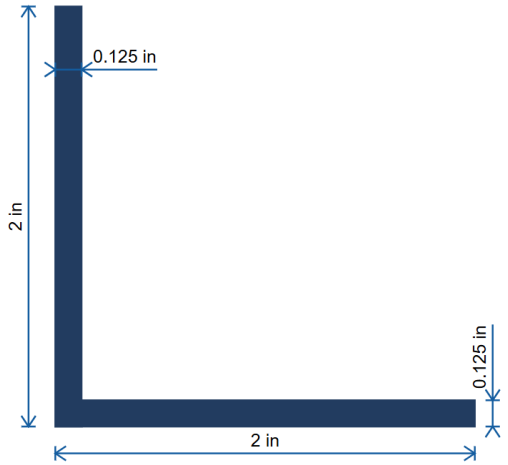


Figure 42 L2X2X1/8 I beam [55]

Table 8 L2X2X1/8 beam properties [55]

A (in^2)	b (in)	t (in)	d (in)	I_x (in^4)	S_x (in^3)	S_y (in^3)	J (in^4)
0.491	2	0.125	2	0.189	0.129	0.129	0.00293

Table 9 L2X2X1/8 beam loads

Axial (lbf)	X Shear (lbf)	Y Shear (lbf)	X moment (in-lbf)	Y moment (in-lbf)	Torsion (in-lbf)
11240	11240	11240	11240	11240	11240

The equation below calculates axial stress which matches the calculator's stress of 22892.1psi.

$$\sigma_A = \frac{11240}{0.491} = 22892.1psi \tag{84}$$

The equation below calculates shear stress in the X direction which matches the calculator's stress of 67440 psi.

$$\tau_x = \frac{3}{2}\frac{11240}{2 * 0.125} = 67440psi \tag{85}$$

The equation below calculates shear stress in the Y direction which matches the calculator's stress of 67440 psi.

$$\tau_y = \frac{3}{2}\frac{11240}{2 * 0.125} = 67440psi \tag{86}$$

The equation below calculates bending stress from a moment around the X axis which matches the calculator's stress of 87131.8psi

$$\sigma_{bx} = \frac{11240}{0.129} = 87131.8psi \tag{87}$$

The equation below calculates bending stress from a moment around the Y axis which matches the calculator's stress of 87131.8psi

$$\sigma_{by} = \frac{11240}{0.129} = 87131.8psi \tag{88}$$

The equation below calculates the torsional stress which matches the calculator's stress of 479522psi.

$$\tau_t = \frac{11240 * 0.125}{0.00293} = 479522psi \tag{89}$$

The equation below calculates the combined stress in the first leg using the von mises criteria which matches the calculator's stress of 861591psi.

$$\sigma_{vm,leg1} = \sqrt{(22892 + 87131.8 + 87131.8)^2 + 3(67440^2 + 479522^2)} = 861591psi \tag{90}$$

The equation below calculates the combined stress in the first leg using the von mises criteria which matches the calculator's stress of 861591psi.

$$\sigma_{vm,leg2} = \sqrt{(22892 + 87131.8 + 87131.8)^2 + 3(67440^2 + 479522^2)} = 861591psi \tag{91}$$

The inputs and outputs to the beam calculator can be seen in the figures below and it is clear that the outputs match the values calculated in the equations above.

Beam Calculator

Last Update: 11/5/2025

Inputs				Beam Options:
	X	Y		
Applied Shear	11240	11240	lbs	W
Applied Moment	11240	11240	in-lbs	M
				S
Applied Axial	11240		lbs	HP
Applied Torsion	11240		in-lbs	C
				MC
				L
Ultimate Stress	60000		psi	WT
Yield Stress	46000		psi	MT
				HSS
FSU	3			CHS
FSY	2			PIPE
Ult Allowed	20000		psi	
Yield Allowed	23000		psi	
Beam Type	L			

Figure 43 Beam calculator inputs for a L2X2X1/8 beam

Axial	Bending Stress		Shear Stress		Torsional Stress		Combined Stress	
Stress	X-axis	Y-axis	X-axis	Y-axis			Leg1	Leg2
22892.06	87131.78	87131.78	67440.00	67440.00	479522.18		861591.19	861591.19

Figure 44 Beam calculator results for a L2X2X1/8 beam

6.1.4 T Beam Equation Verification

The T beam chosen to verify the calculator is the WT22X204 beam shown in the figure below. The properties for this beam used in the equations are shown in the table below that. Then the loads used are shown in the table below that.

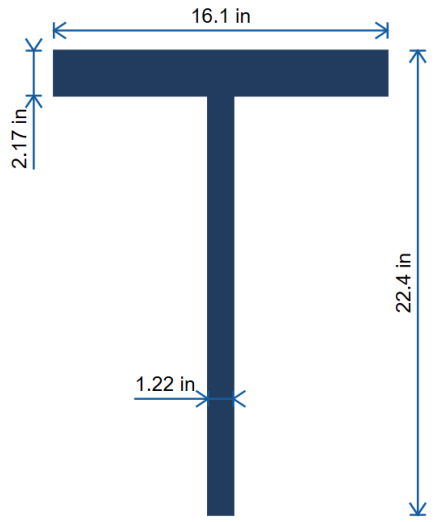


Figure 45 WT22X204 beam profile [55]

Table 10 WT22X204 beam properties [55]

A (in^2)	b_f (in)	t_f (in)	t_w(in)	d (in)	I_x (in^4)	S_x (in^3)	S_y (in^3)	J (in^4)
59.9	16.1	2.17	1.22	22.4	2670	160	94.2	66.9

Table 11 WT22X204 beam loads

Axial (lbf)	X Shear (lbf)	Y Shear (lbf)	X moment (in-lbf)	Y moment (in-lbf)	Torsion (in-lbf)
11240	11240	11240	11240	11240	11240

The equation below calculates axial stress which matches the calculator's stress of 187.65psi.

$$\sigma_A = \frac{11240}{59.9} = 187.65psi \tag{92}$$

The equation below calculates shear stress in the X direction which matches the calculator's stress of 482.6psi.

$$\tau_x = \frac{3}{2}\frac{11240}{16.1 * 2.17} = 482.6psi \tag{93}$$

The equation below calculates shear stress in the X direction which matches the calculator's stress of 683.13psi.

$$\tau_y = \frac{3}{2}\frac{11240}{(22.4 - 2.17)1.22} = 683.13psi \tag{94}$$

The equation below calculates bending stress from a moment around the X axis which matches the calculator's stress of 70.25psi.

$$\sigma_{bx} = \frac{11240}{160} = 70.25psi \tag{95}$$

The equation below calculates bending stress from a moment around the Y axis which matches the calculator's stress of 119.3psi.

$$\sigma_{by} = \frac{11240}{94.2} = 119.3psi \tag{96}$$

The equation below calculates torsional stress in the flange which matches the calculator's stress of 364.59psi.

$$\tau_{t,flange} = \frac{11240 * 2.17}{66.9} = 364.59psi \quad (97)$$

The equation below calculates torsional stress in the web which matches the calculator's stress of 204.98psi.

$$\tau_{t,web} = \frac{11240 * 1.22}{66.9} = 204.98psi \quad (98)$$

The equation below calculates the total stress in the flange using the von mises criteria which matches the calculator's stress of 1113.46psi.

$$\sigma_{vm,flange} = \sqrt{(187.65 + 70.25 + 119.3)^2 + 3(482.6^2 + 364.59^2)} = 1113.46psi \quad (99)$$

The equation below calculates the total stress in the web using the von mises criteria which matches the calculator's stress of 1291.64psi.

$$\sigma_{vm,web} = \sqrt{(187.65 + 70.25 + 119.3)^2 + 3(683.13^2 + 204.98^2)} = 1291.64psi \quad (100)$$

The figures below show the inputs and outputs of the calculator for this beam matching the calculated stresses from the equations above.

Beam Calculator

Last Update: 11/5/2025

Inputs

	X	Y	
Applied Shear	11240	11240	lbs
Applied Moment	11240	11240	in-lbs
Applied Axial	11240		lbs
Applied Torsion	11240		in-lbs
Ultimate Stress	60000		psi
Yield Stress	46000		psi
FSU	3		
FSY	2		
Ult Allowed	20000		psi
Yield Allowed	23000		psi
Beam Type	WT		

Beam Options:
W
M
S
HP
C
MC
L
WT
MT
HSS
CHS
PIPE

Figure 46 Beam calculator inputs for WT22X204

Axial	Bending Stress		Shear Stress		Torsional Stress		Combined Stress		
Stress	X-axis	Y-axis	X-axis	Y-axis	Flange	Web	Flange	Web	
187.65	70.25	119.32	482.58	683.13	364.59	204.97	1113.43	1291.64	

Figure 47 Beam calculator results for WT22X204

6.1.5 HSS Beam Equation Verification

The beam used to verify the HSS beams is the HSS34X10X1which is shown in the figure below. The properties for this beam used in the equations are shown in the table below that. Then the loads used are shown in the table below that.

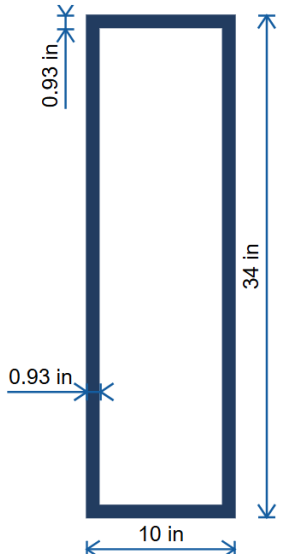


Figure 48 HSS34X10X1 profile [55]

Table 12 HSS34X10X1 beam properties [55]

A (in^2)	B (in)	t_{des} (in)	h (in)	S_x (in^3)	S_y (in^3)
76.2	10	0.93	31.2	565	268

Table 13 HSS34X10X1 beam loads

Axial (lbf)	X Shear (lbf)	Y Shear (lbf)	X moment (in-lbf)	Y moment (in-lbf)	Torsion (in-lbf)
11240	11240	11240	11240	11240	11240

The equation below calculates axial stress which matches the calculator's stress of 147.5psi.

$$\sigma_A = \frac{11240}{76.2} = 147.5psi \tag{101}$$

The equation below calculates the shear stress in the X direction which matches the calculator's stress of 742.4psi.

$$\tau_x = \frac{11240}{2 * 0.93(10 - 2 * 0.93)} = 742.4psi \quad (102)$$

The equation below calculates the shear stress in the Y direction which matches the calculator's stress of 205.97psi.

$$\tau_y = \frac{11240}{2 * 0.93(31.2 - 2 * 0.93)} = 205.97psi \quad (103)$$

The equation below calculates stress due to a bending moment around the X axis which matches the calculator's stress of 19.9psi.

$$\sigma_{bx} = \frac{11240}{565} = 19.9psi \quad (104)$$

The equation below calculates stress due to a bending moment around the Y axis which matches the calculator's stress of 41.94psi.

$$\sigma_{by} = \frac{11240}{268} = 41.94psi \quad (105)$$

The equation below calculates the torsional stress which matches the calculator's stress of 22.01psi.

$$\tau_t = \frac{11240}{2 * 0.93(31.2 - 0.93)(10 - 0.93)} = 22.01psi \quad (106)$$

The equation below calculates the total stress in the X wall using the von mises criteria which matches the calculator's stress of 1302.78psi.

$$\sigma_{vm,x} = \sqrt{(148 + 20 + 42)^2 + 3(742^2 + 22^2)} = 1302.78psi \quad (107)$$

The equation below calculates the total stress in the Y wall using the von mises criteria which matches the calculator's stress of 415.76psi.

$$\sigma_{vm,y} = \sqrt{(148 + 20 + 42)^2 + 3(206^2 + 22^2)} = 415.76psi \quad (108)$$

Beam Calculator

Last Update: 11/5/2025

Inputs				Beam Options:
	X	Y		W
Applied Shear	11240	11240	lbs	M
Applied Moment	11240	11240	in-lbs	S
				HP
Applied Axial	11240		lbs	C
Applied Torsion	11240		in-lbs	MC
				L
Ultimate Stress	60000		psi	WT
Yield Stress	46000		psi	MT
				HSS
FSU	3			CHS
FSY	2			PIPE
Ult Allowed	20000		psi	
Yield Allowed	23000		psi	
Beam Type	HSS			

Figure 49 Beam calculator inputs for HSS34X10X1

Axial	Bending Stress		Shear Stress		Torsional Stress		Combined Stress		
Stress	X-axis	Y-axis	X-axis	Y-axis		Torsional		Wall X	Wall Y
147.51	19.89	41.94	742.38	205.96		22.01		1303.33	415.38

Figure 50 Beam calculator results for HSS34X10X1

6.1.6 PIPE Beam Equation Verification

The beam used to verify the PIPE beams is the Pipe2XXS which is shown in the figure below. The properties for this beam used in the equations are shown in the table below that. Then the loads used are shown in the table below that.

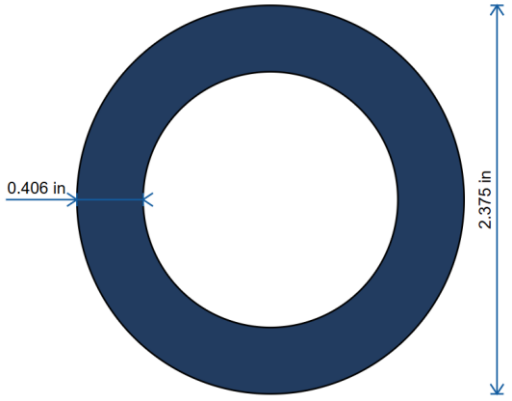


Figure 51 Pipe2XXS profile [55]

Table 14 Pipe2XXS beam properties [55]

A (in^2)	OD (in)	I_x (in^4)	J (in^4)
2.51	2.375	1.27	2.54

Table 15 Pipe2XXS beam loads

Axial (lbf)	X Shear (lbf)	Y Shear (lbf)	X moment (in-lbf)	Y moment (in-lbf)	Torsion (in-lbf)
11240	11240	11240	11240	11240	11240

The equation below calculates the axial stress of the beam which matches the calculator's stress of 4478.1psi.

$$\sigma_A = \frac{11240}{2.51} = 4478.1psi \tag{109}$$

The equation below calculates shear stress which matches the calculator's stress of 8443.96psi.

$$\tau = 2\frac{\sqrt{11240^2 + 11240^2}}{2.51} = 12666psi \tag{110}$$

The equation below calculates the bending stress of the beam which matches the calculator's stress of 14863.2psi.

$$\sigma_b = \frac{\sqrt{11240^2 + 11240^2}}{1.07} = 14856psi \tag{111}$$

The equation below calculates torsional stress which matches the calculator's stress of 5254.92psi.

$$\tau_t = \frac{11240 * {}^{2.375}/_{2}}{2.54} = 5254.92psi \tag{112}$$

The equation below calculates the total stress in the beam using the von mises criteria which matches the calculator's stress of 25900.3psi.

$$\sigma_{vm,x} = \sqrt{(4478 + 14856)^2 + 3(12666^2 + 5254.92^2)} = 30626psi \tag{113}$$

Beam Calculator

Last Update: 11/5/2025

Inputs				Beam Options:
	X	Y		W
Applied Shear	11240	11240	lbs	M
Applied Moment	11240	11240	in-lbs	S
				HP
Applied Axial	11240		lbs	C
Applied Torsion	11240		in-lbs	MC
				L
Ultimate Stress	60000		psi	WT
Yield Stress	46000		psi	MT
				HSS
FSU	3			CHS
FSY	2			PIPE
Ult Allowed	20000		psi	
Yield Allowed	23000		psi	
Beam Type	PIPE			

Figure 52 Beam calculator inputs for Pipe2XXS

Axial Stress	Bending Stress		Shear Stress		Torsional Stress		Combined Stress	
		Bending		Shear		Torsional		Combined
4478.09		14863.16		8443.96		5254.92		25900.35

Figure 53 Beam calculator results for Pipe2XXS

6.2 Beam Calculator FEA Comparison

The equations above verify that the beam calculator matches the principles laid out for basic mechanics of materials algorithms. But these calculations need to be compared to finite element methods to determine if the calculator over estimates or underestimates stress.

To do this, the W4X13 I beam was used with the same forces of 50,000 newtons (11,240.45 lbf) and 1,270 NM (11,240 in-lbf) applied in varying directions. The beam calculator was used and then compared to the identical I beam in an FEA software called FEMAP Nastran.

6.2.1 Setup

To ensure the FEA model produced accurate numbers the geometry of the W4X13 beam was refined to prevent as many analytical errors as possible. The properties for the I beam were taken from the AISC database [55] with a flange thickness of 0.345 inches, a web thickness of 0.56 inches and a radius of 0.25 inches connecting the flanges to the web. Although the AISC database does not specify the radius of any other edge, a radius of 0.02 inches was chosen because infinitely sharp corners are

bad for FEA and are impossible to manufacture. The entire profile of the I beam was extruded to 4mm thick. This was done to reduce moments when analyzing pear shear forces.

The geometry was then meshed in FEMAP Nastran with 10-noded tetrahedral (TET10) solid elements. Aluminum 6061 was chosen for the material because it is a common material used for this I beam. It was modeled as linear elastic with an elastic modulus of $6.8*10^{10}$ Pa, a shear modulus of $2.556*10^{10}$ and a Poisson's ratio of 0.33. Every node on the bottom surface of the mesh was fixed as shown in the right figure below. Every node on the top of the model was attached to an infinitely rigid body element (RBE2) spider node which converged to a central point where the loads were applied which is shown in the figure below on the left.

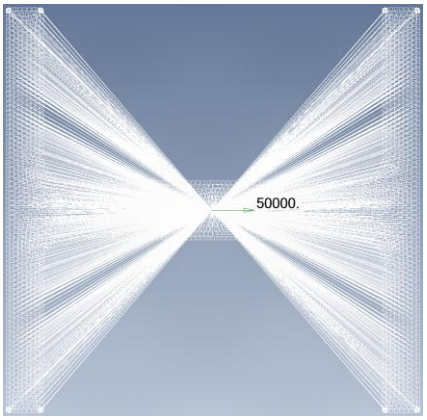


Figure 54 REB2 with load

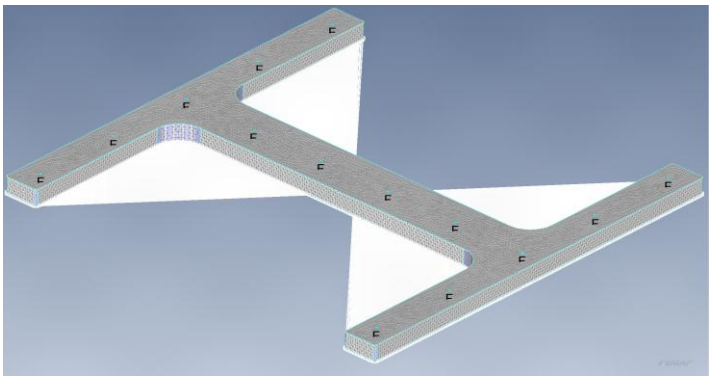


Figure 55 Fixed constraint

To further ensure that the stresses calculated by the model are accurate, a mesh convergence study was conducted using five different mesh sizes 5mm, 2.5mm, 1.25mm, 0.625mm, and 0.3125mm. These dimensions are the nominal length of the sides of the tetrahedron and allow stretch +/- 10% of the nominal length to accommodate complex geometry.

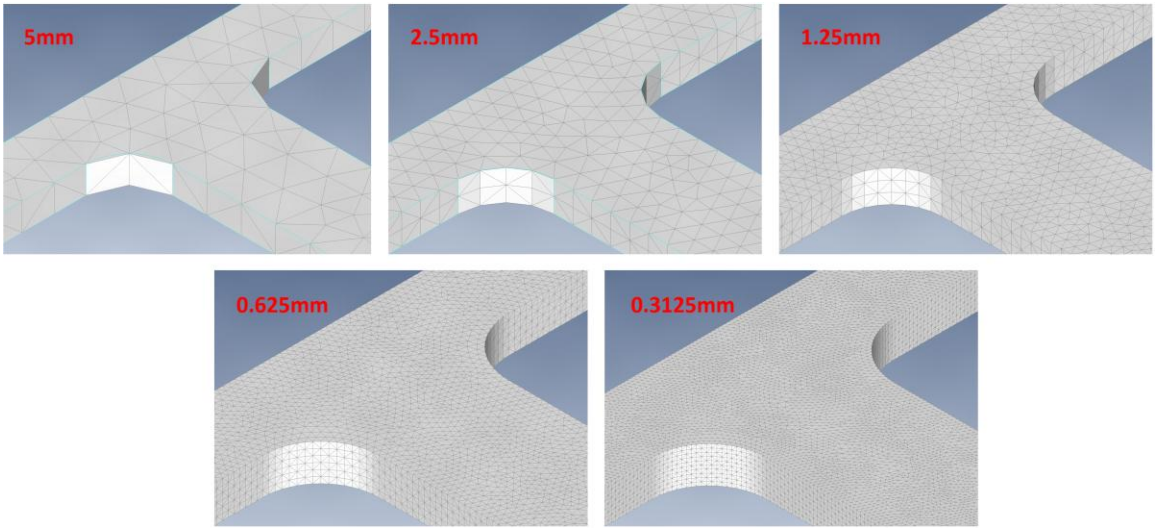


Figure 56 Mesh sizes

The model was then statically analyzed using Simcenter NASTRAN. The results are then analyzed using the solid Von Mises criteria. Because stress peaks at sharp corners and RBE attachment points the stress values are recorded at certain points that match where the max stress is expected to be for each force type. This point is kept constant between meshes. The points used for analyzing stress can be seen in the figures below.

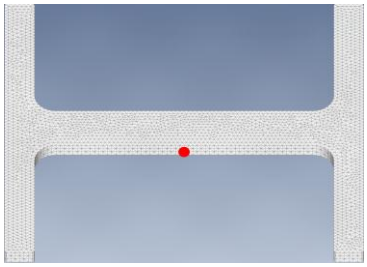


Figure 57 Axial location

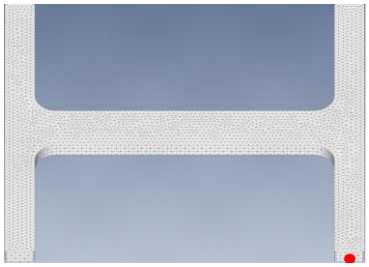


Figure 58 Shear X location

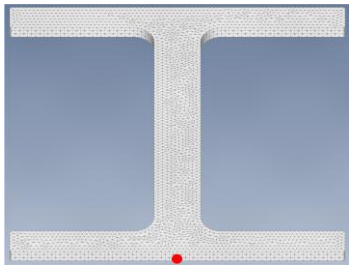


Figure 59 Shear Y location

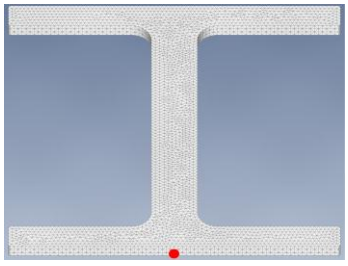


Figure 60 Bending X location

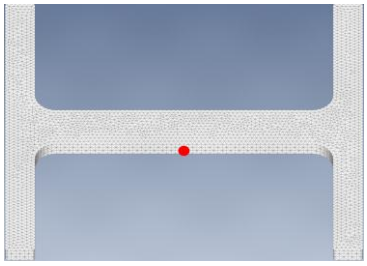


Figure 61 Bending Y location

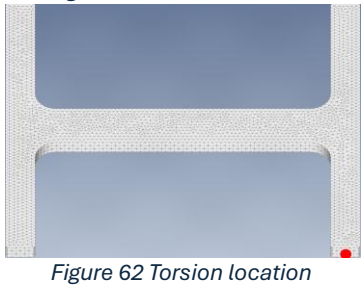


Figure 62 Torsion location

The stresses between the meshes will then be analyzed using the grid convergence index (GCI) methodology to quantify the uncertainty in the results [58]. GCI is performed to ensure that the results produced by the FEA model are independent of the mesh. This study is assuming a confidence of 95% meaning that any error bound larger than 5% is significantly affected by the mesh. The equation below is used to calculate the grid convergence index where F_s is the safety factor of 1.25 as recommended by ASME [59], σ_i is the stress of the current mesh and σ_{i-1} is the stress of the previous model, r is the mesh refinement rate of 2 and p is the apparent order of accuracy.

$$GCI = F_s \frac{|\sigma_{i-1} - \sigma_i|}{\sigma_i (r^p - 1)} \tag{114}$$

The apparent order of accuracy (p) is calculated using the equation below where three mesh stresses are required. In the equation below the finest most mesh stresses are shown. And then r is the mesh refinement rate which is 2 because the size of the mesh is decreased at a rate of 2.

$$p = \frac{\ln\left(\frac{\sigma_3 - \sigma_4}{\sigma_4 - \sigma_5}\right)}{\ln(r)} \tag{115}$$

6.2.2 Normal Force

The first force analyzed was the axial force where a force of 50,000 newtons (11240.45 lbf) was then applied in a pure axial direction in both FEMAP Nastran and the Beam Calculator. Shown in figure 63 FEMAP returned a value of 20.01MPa (2902.21 psi) with 4.15% uncertainty. The beam calculator returned a value of 2934.73 psi (20.23 MPa) which is shown in figure 65. There is a difference of 32.63 psi or .22MPa which means that the beam calculator slightly overestimates the total stress in the beam but overall is remarkably close.

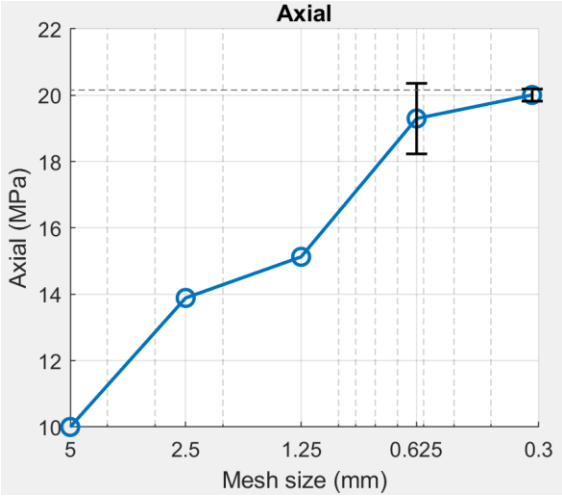


Figure 63 FEA GCI results for axial

The inputs and results to the beam calculator can be seen below where the applied axial force was applied at 11240.45 lbf.

Beam Calculator

Last Update: 3/21/2025

Inputs

	X	Y	
Applied Shear			lbs
Applied Moment			in-lbs
Applied Axial	11240.45		lbs
Applied Torsion			in-lbs

Beam Options:
W
M
S
HP
C
MC

Figure 64 Beam calculator inputs

Axial	Bending Stress		Shear Stress		Torsional Stress	
Stress	X-axis	Y-axis	X-axis	Y-axis	Flange	Web
2934.73	2058.61	5915.79	5233.01	10977.12	25680.79	20842.38

Figure 65 Beam calculator results

6.2.3 Shear Stress (X):

The second force analyzed is shear force in the X direction. 50,000 newtons (11240.45 lbf) were applied in a pure shear direction in both FEMAP Nastran and Beam Calculator. Shown in Figure 66 FEMAP Nastran calculates 43 MPa (6236.62psi) with an error of 2.03% while the beam calculator calculates 5233 psi (36.1MPa) which is shown in figure 68. This is a difference of 1,003.62 psi (6.9 MPa) where the beam calculator underestimates stress. This is due to the beam calculator dividing the calculated stress in half.

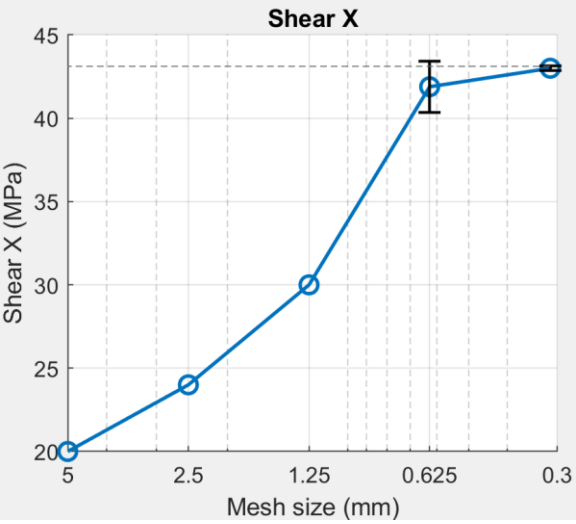


Figure 66 FEA GCI results for shear X

The inputs and results to the beam calculator can be seen below where the applied shear in the X direction was applied at 11240.45 lbf.

Beam Calculator

Last Update: 3/21/2025

Inputs

	X	Y		Beam Options:
Applied Shear	11240.45		lbs	W
Applied Moment			in-lbs	M
				S
Applied Axial			lbs	HP
Applied Torsion			in-lbs	C

Figure 67 Beam calculator inputs

Axial	Bending Stress		Shear Stress		Torsional Stress	
Stress	X-axis	Y-axis	X-axis	Y-axis	Flange	Web
2934.73	2058.61	5915.79	5233.01	10977.12	25680.79	20842.38

Figure 68 Beam calculator results

6.2.4 Shear Stress (Y)

The same process is applied to the analyses for shear force in the Y direction. A force of 50,000 newtons (11240.45 lbf) is applied in a purely shear direction in both FEMAP Nastran and Beam Calculator. Shown in figure 69 FEMAP Nastran calculates 40.1 MPa (5915.8psi) with an error bound of 2.28% while the beam calculator calculates 10977.12psi (75.7MPa) which is shown in figure 71. This creates a difference of 35.6 MPa (5163.3psi) where the beam calculator overestimates stress.

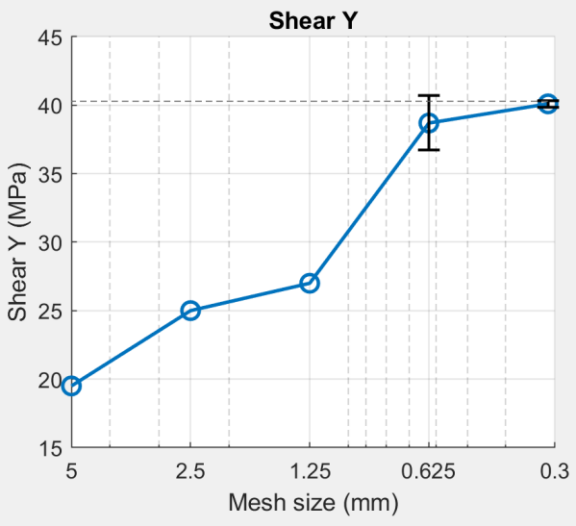


Figure 69 FEA GCI results for shear Y

The inputs and results to the beam calculator can be seen below where the applied shear in the Y direction was applied at 11240.45 lbf.

Beam Calculator

Last Update: 3/21/2025

Inputs

	X	Y		Beam Options:
Applied Shear		11240.45	lbs	W
Applied Moment			in-lbs	M
				S
Applied Axial			lbs	HP
Applied Torsion			in-lbs	C
				MC

Figure 70 Beam calculator inputs

Axial	Bending Stress		Shear Stress		Torsional Stress	
Stress	X-axis	Y-axis	X-axis	Y-axis	Flange	Web
2934.73	2058.61	5915.79	5233.01	10977.12	25680.79	20842.38

Figure 71 Beam calculator results

6.2.5 Bending Stress (X)

The next force analyzed is the bending about the X-axis where a moment of 1,270 newton-meters (11240.45 in-lbf) is applied around the X-axis direction in both FEMAP Nastran and Beam Calculator. Shown in Figure 72 FEMAP Nastran calculates 15.5 MPa (2248.1psi) of stress with an error of 2.57% while the beam calculator calculates 14.2 MPa (2058.6psi) which is shown in figure 74. This creates a difference of 1.3 MPa (178psi) where the beam calculator underestimates stress.

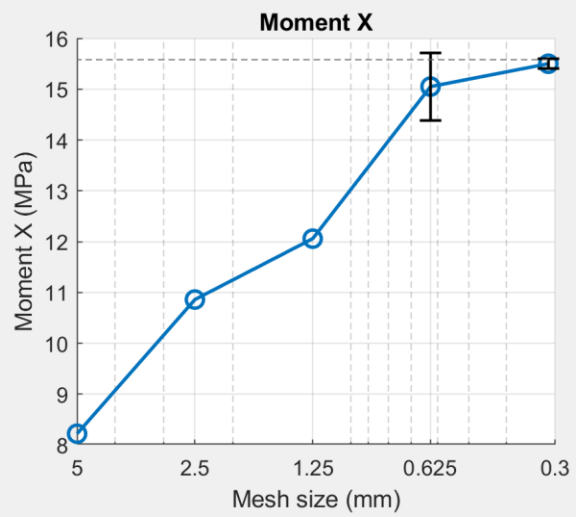


Figure 72 FEA GCI results for moment X

The inputs and results to the beam calculator can be seen in the figures below where the moment around the X axis was applied at 11240.45 in-lbf.

Beam Calculator

Last Update: 3/21/2025

Inputs

	X	Y		Beam Options:
Applied Shear			lbs	W
Applied Moment	11240.45		in-lbs	M
				S
Applied Axial			lbs	HP
Applied Torsion			in-lbs	C
				MC

Figure 73 Beam calculator inputs

Axial	Bending Stress		Shear Stress		Torsional Stress	
Stress	X-axis	Y-axis	X-axis	Y-axis	Flange	Web
2934.73	2058.61	5915.79	5233.01	10977.12	25680.79	20842.38

Figure 74 Beam calculator results

6.2.6 Bending Stress (Y)

The same moment of 1,270 newton-meters (11240.45 in-lbf) is applied around the Y-axis in both FEMAP Nastran and Beam Calculator. Shown in figure 75 FEMAP Nastran calculates 44.13MPa (6400.5psi) of stress with an error of 0.48% while the beam calculator calculates 40.79 MPa (5915.8psi) which is shown in figure 77. This creates a difference of 3.34MPa (484.4psi) where the beam calculator underestimates stress.

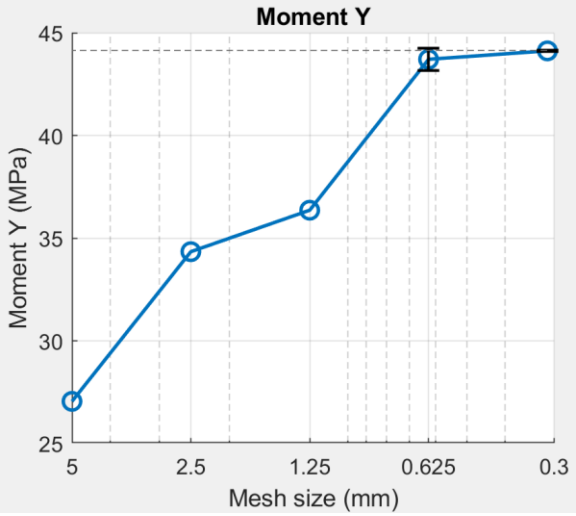


Figure 75 FEA GCI results for moment Y

The inputs and results to the beam calculator can be seen in the figures below where the moment around the Y axis was applied at 11240.45 in-lbf.

Beam Calculator

Last Update: 3/21/2025

Inputs				
	X	Y		Beam Options:
Applied Shear			lbs	W
Applied Moment		11240.45	in-lbs	M
				S
Applied Axial			lbs	HP
Applied Torsion			in-lbs	C
				MC

Figure 76 Beam calculator inputs

Axial	Bending Stress		Shear Stress		Torsional Stress	
Stress	X-axis	Y-axis	X-axis	Y-axis	Flange	Web
2934.73	2058.61	5915.79	5233.01	10977.12	25680.79	20842.38

Figure 77 Beam calculator results

6.2.7 Torsion

Torsion is where the beam calculator loses reliability as it is limited by only using basic algebraic formulas. When a 1,270 N-M (11240.45 in-lbf) is applied to the central axis of the beam FEMAP Nastran calculates a max stress of 26.86 MPa (3895.71psi) with an error of 1.73% which is shown in figure 78. When the same forces are applied to the beam calculator a max stress value of 25680.8psi (177.1 MPa) which is nowhere near the calculated value of 26.86 MPa given by FEMAP. This is an extreme overestimate of 150.24 MPa or 21.8 ksi.

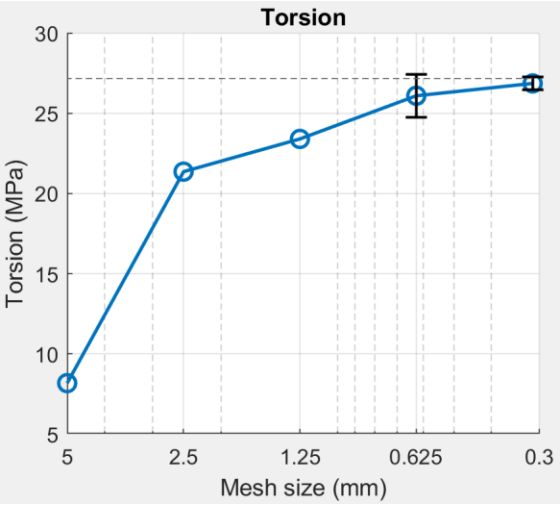


Figure 78 FEA GCI results for torsion

The inputs and results to the beam calculator can be seen in the figures below where the torsional load was applied at 11240.45 in-lbf.

Beam Calculator

Last Update: 3/21/2025

	X	Y		Beam Options:
Applied Shear			lbs	W
Applied Moment			in-lbs	M
				S
Applied Axial			lbs	HP
Applied Torsion	11240		in-lbs	C
				MC

Figure 79 Beam calculator inputs

Axial	Bending Stress		Shear Stress		Torsional Stress	
Stress	X-axis	Y-axis	X-axis	Y-axis	Flange	Web
2934.73	2058.61	5915.79	5233.01	10977.12	25680.79	20842.38

Figure 80 Beam calculator results

6.2.8 FEA Summary

Overall, the calculator overestimates for almost every single force except for bending in the X and Y direction and shear stress in the X direction but the bending was off by 1.3MPa and 3.34MPa respectively and the shear in the X direction is divided by two in the calculator so that could be made more conservative in the future. But given the scope of this calculator, which is to give an approximate ballpark answer for which beam to choose based on loads, materials, and safety factors, it accomplishes its job. The preliminary design should then be optimized using FEA. The results of all the FEA mesh convergence studies can be seen in the figure below.

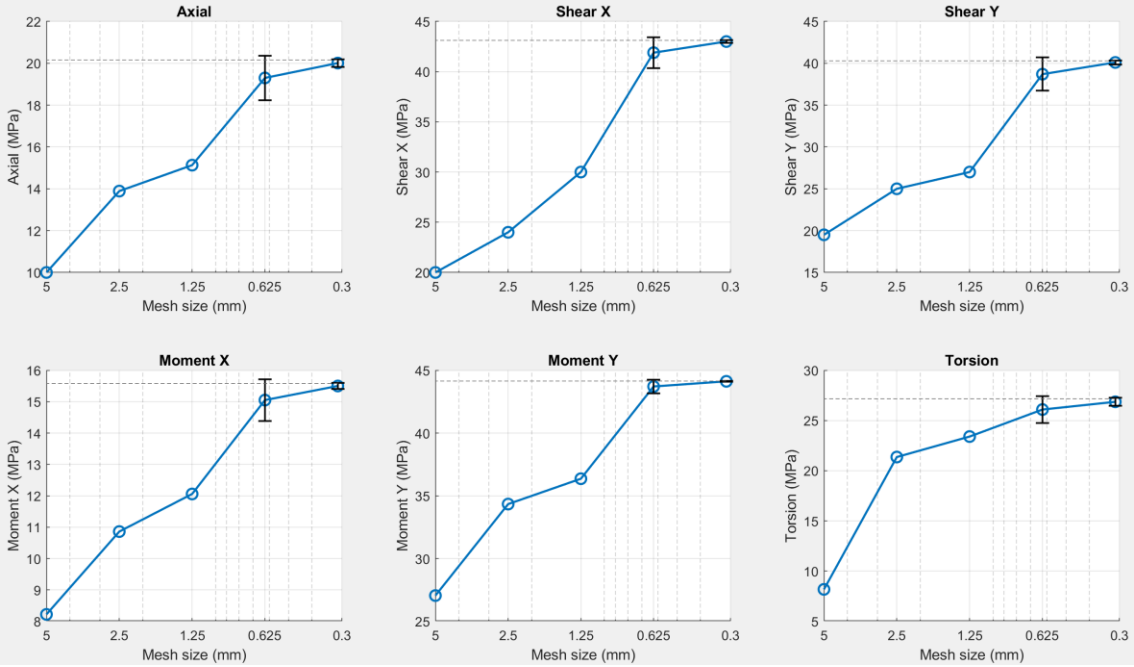


Figure 81 FEA GCI results for all 6 loads

Since the error bounds for the first 3 mesh sizes were so large they were omitted from the graphs above and can be seen in the table below

Table 16 GCI error bounds for all meshes and forces

Stress	GCI_1	GCI_2	GCI_3	GCI_4	GCI_5
Axial	65.10%	46.61%	36.11%	9.46%	4.15%
Shear X	63.00%	25.05%	18.96%	5.04%	2.03%
Shear Y	60.72%	24.35%	19.09%	6.37%	2.28%
Moment X	65.01%	42.55%	33.73%	6.14%	2.57%
Moment Y	51.74%	31.75%	22.44%	2.78%	0.48%
Torsion	62.23%	34.14%	27.86%	4.71%	1.73%

7 System Validation

In engineering, validation ensures that a tool fulfills its intended purpose by solving the right problem [3]. The aim of this tool is to increase engineering efficiency without sacrificing quality. So, if the tool successfully expedites project timelines without creating failures the tool will be considered validated. To accomplish this the tool was used in three different projects. Each project used at least one feature of the tool and the tool's contribution was assessed based on design time, design success, and feedback.

7.1 Lunar Terrain Vehicle Wheel Test Stand

The first project that this tool was used for was an advanced lunar wheel test stand. The aim of the project was to run longer duration tests on lunar terrain vehicle wheels in lunar conditions. The test stand had to be able to withstand forces over 200 pounds, cryogenic temperatures and vacuum conditions. The test stand also had to fit within certain physical constraints and be portable via jacks or crane. On top of all of that, the system also had extreme time and money constraints, meaning that a solution needed to be created as quickly as possible.

The design process started with the loads and constraints. Once they were determined, a free body diagram was used to solve the loads in each beam. Then the loads were applied to the beam calculator with safety factors of 3 and 4 to yield and ultimate respectively. The design in the figure below was eventually created in less than two days.

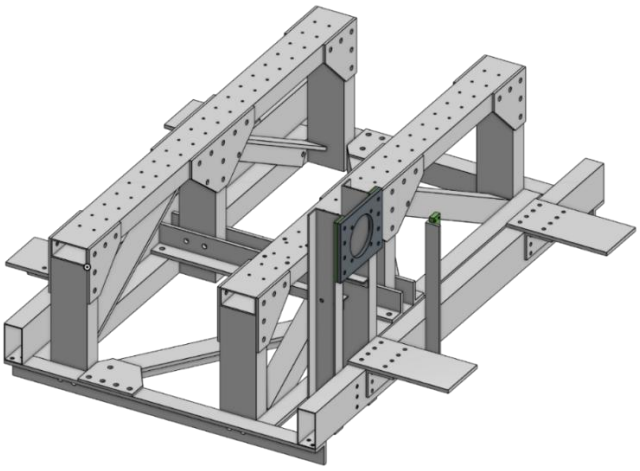


Figure 82 LTV wheel test stand

The design consisted of multiple beams that were determined to work by using the beam calculator. Without the beam calculator extra time would have been spent looking for the beams and the proper beam most likely would not have been chosen. A more detailed overview of the test stand can be seen in the figure below.

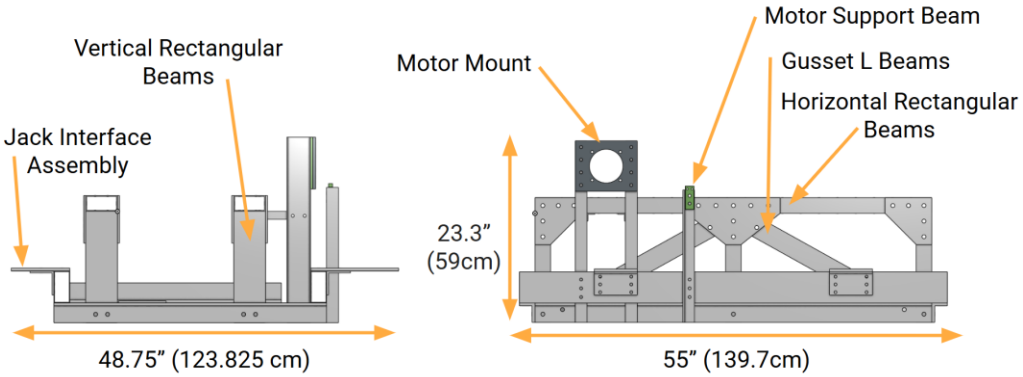


Figure 83 LTV wheel test stand dimensions

The tests conducted on the test stand worked flawlessly without the test stand failing or deflecting in a major way that affected the test results. This shows that the beam calculator decreased design time without sacrificing quality.

7.2 Regolith Dispensers

Another project that used the engineering tool was a regolith dispenser. The idea behind the dispenser was that it could deposit regolith onto lunar actuators in cryogenic and vacuum conditions. The figure below shows a regolith sifter where the regolith would sit in the main hopper and then a mesh would reciprocate under the regolith dispensing fine regolith onto a lunar actuator below.

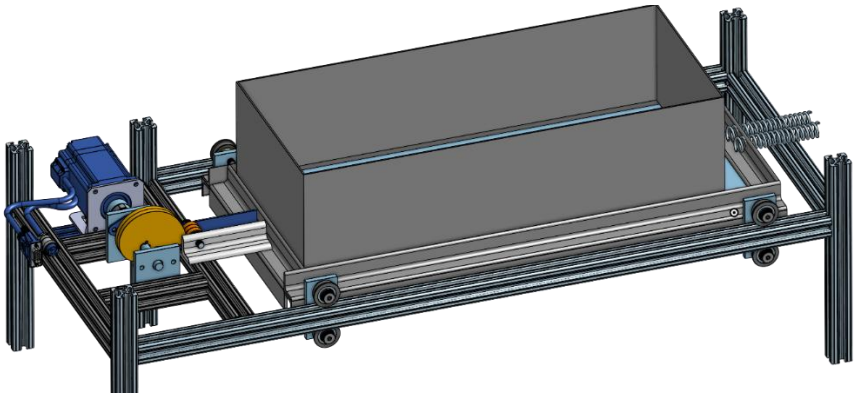


Figure 84 Regolith dispenser

To accomplish the linear reciprocating motion a spring-loaded CAM system was used. At first the CAM was designed using approximate sketching, but this was found to be difficult, timely and the final design was not continuous or seamless. So, the CAM calculator was used, and it turned a multi hour long task into a task that took a couple of seconds. The CAM CAD shown in the figure below is an autogenerated STEP file created by the calculator within seconds. This was uploaded to the main assembly shown in the figure above and then sent directly to a CAM software to create G-code for a CNC machine. Since the CAM calculator also creates STL files the design was sent directly to a 3D printing slicer for rapid prototyping.

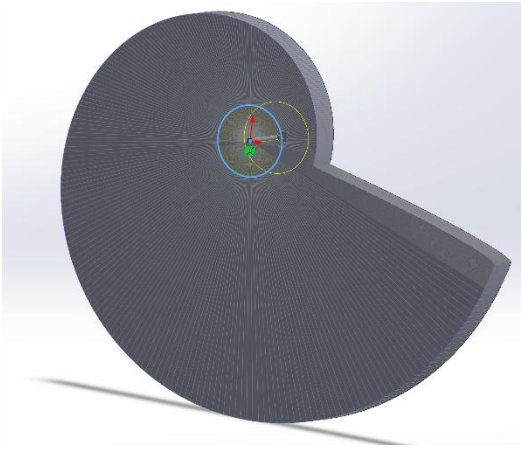


Figure 85 CAD STEP model of CAM designed by CAM calculator

Figure 86 CNC machined CAMs

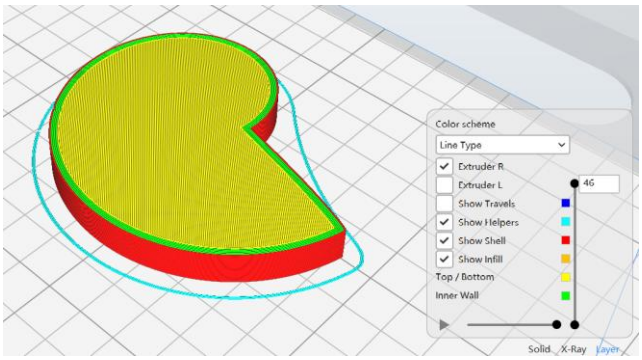


Figure 87 3D printer slice of CAM

The regolith dispenser that utilized the CAM designer tool from the engineering tool worked flawlessly and allowed the project to be completed ahead of schedule. Without the tool the extra time spent on creating the CAM would have failed to meet the deadline and would have resulted in a subpar design.

7.3 Lunar Anchor Test Stand

The final project that the engineering tool has been used on is the Automated Testbed for Lunar Anchoring Systems (ATLAS). The goal of this test stand is to test multiple different lunar anchors in both atmospheric and vacuum conditions. The system had to be able to rotate anchors into the ground, hammer anchors into the ground and hammer + rotate at the same time. The system also had to be able to handle forces up to 2KN and record data such as pull-out force, insertion power and displacement. The system shown in the figure below was designed to handle these tasks.

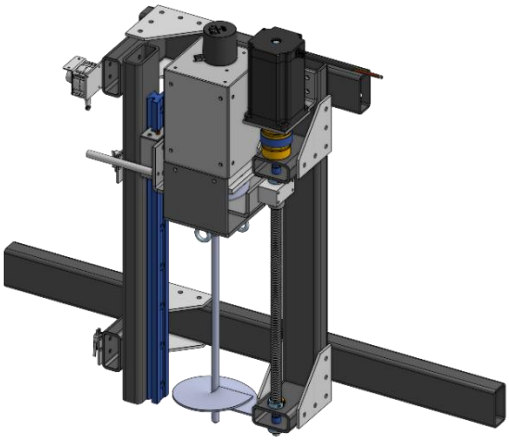


Figure 88 ATLAS

The main frame shown in the figure below was designed using the beam calculator which took less than an hour to have the entire system designed. Without the beam calculator it would have taken an entire day requiring either repetitive hand calculations iterating through different beams, or iterative CAD-FEA cycles.

Figure 89 ATLAS main frame

An important part of the ATLAS is to record force data which is best done using an NI data acquisition system which works best when combined with LabVIEW. LabVIEW is also great for logic control for mechanical engineers who do not know any other programming language very well. For these reasons LabVIEW was chosen as the main software for the project. This meant that to include the BLDC motor in the control diagram the LabVIEW ODrive controller from the engineering tool had to be used. Before the creation of the engineering tool, there was no easy way to control a BLDC motor using LabVIEW and ODrive but with the new tool the ODrive controller could be dropped into the pre-existing code and integrated with the rest of the control logic. The front panel of the ATLAS control code can be seen below where the drill motor section is the ODrive controller modified to accommodate the needs of the lunar anchor testing.

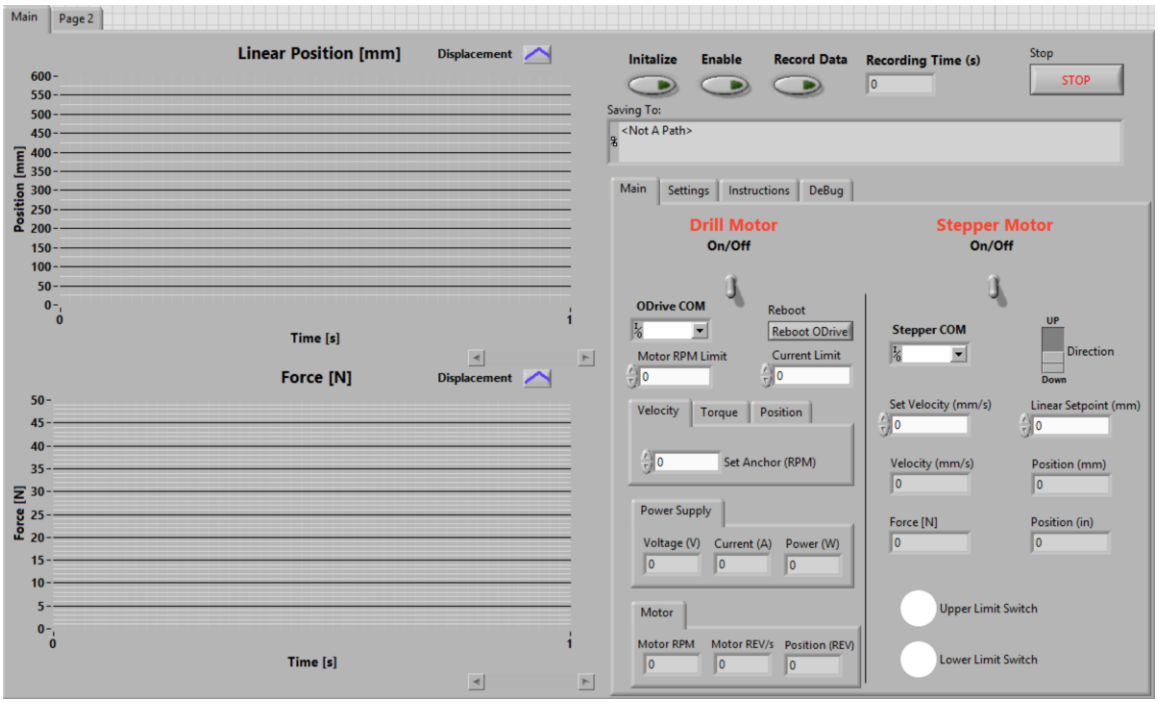


Figure 90 Integrated ODrive controller

The ATLAS project finished precisely on time with no second to spare. Without the use of the beam calculator or the BLDC ODrive motor controller from the engineering tool the project would have been pushed back a day or two. This means that without the engineering tool the test campaign would have been incomplete.

8 Discussion

The goal of this work was to review the existing engineering tools, identify gaps and then develop a new tool based on the existing ones and then verify and validate. The tool developed in this paper completes all the objectives for this paper. It is not intended to be a complete, polished commercial product; instead, it serves as a framework and proof-of-concept implementation that demonstrates the feasibility of a unified, extensible engineering environment capable of hosting calculators, databases, CAD automation features, and project-management utilities. Most importantly, the architecture of the tool allows for more features to be added easily, enabling long-term growth and customization.

From the literature review there are many engineering tools that address the problems individually of inefficient engineering practice, resources not centralized or project disorganization. But each of these tools does not address all of the problems at once. Furthermore, none of the tools are open source, allowing users to add or edit features. By far the engineering tool available was the StruCalc calculator which is an online pay to use tool. When compared to StruCalc the developed engineering tool shows great improvement in resource availability and knowledge-based engineering CAD model generation. The only place that it falls short is that it does not have tutorials or free body diagrams which are additions that can easily be added to the engineering tool due to its superior modularity. The comparison of the features of the engineering tool vs StruCalc can be seen in the table below.

Table 17 Table of features of Developed engineering tool (same setup as table 1)

	Features														
Tool	**FBD**	**Beams**	**CAM**	**CAD**	**Bolts**	**Mechanics**	**Centralized**	**Tutorials**	**Materials**	**FEA**	**Available**	**Simulation**	**CFD**	**Fast?**	**Traceable**
Engineering tool	no	yes	yes	yes	no	no	yes	no	yes	no	yes	no	no	yes	yes
StruCalc [18]	yes	yes	no	no	no	no	yes	yes	yes	yes	yes	no	no	yes	yes

The engineering tool aims to increase engineering efficiency, resource centralization, and project organization. It accomplishes this by containing calculators that utilize

knowledge-based engineering CAD modeling, databases and tools that centralize resources and project management tools to organize projects. The tool also recognizes the issues associated with calculators that automate process which can create black boxes. So, it addresses this by incorporating metadata knowledge retention systems that reduce the chances of black box syndrome. Finally, the engineering tool also recognizes that it is impossible to incorporate everything in one tool, so it is built on an extremely modular architecture that allows it to expand with ease.

The engineering tool is then verified by comparing its outputs to standard mechanics of material principles and finite element analysis. Since the engineering tool utilizes basic algebraic formulas for all its calculations the values calculated by the engineering tool are within rounding error of the results from the hand computations. The results are then compared to FEA results which show that the values are mostly conservative estimates showing that there is more work to be done to optimize the calculator. And that all beams designed with the beam calculated should be optimized with FEA. In addition, the engineering tool is only meant to aid the engineering process and is not meant to replace any steps such as FEA.

Finally, the engineering tool is validated by using it in critical engineering projects where the use of it clearly advanced the timeline and allowed the projects to be completed on time and most of them under budget. The frame consisting of beams for lunar terrain vehicle wheel testing was designed with the beam calculator within a couple of days, the CAM for the regolith dispenser was designed within a couple of seconds, and the frame and control system for the lunar anchor test stand was designed within a fraction of the time compared to if the engineering tool was not used. This shows that the engineering tool is needed and the engineering V diagram process could gain from its features.

9 Conclusion

The goal of this thesis was to evaluate the current engineering process, identify inefficiencies, review current solutions, and then develop a proof-of-concept framework tool. The three main issues identified in the current engineering process were inefficient design iteration, decentralized resources, and poor project organization. The tool presented in this thesis integrates databases, calculators,

external tools, and project management into one extensive, centralized, and expandable platform.

The system architecture emphasizes the idea that creating a perfect tool on the first try is impossible, so it instead focuses on expandability and modularity. This allows the tool to expand with a theoretically endless number of additional features. To achieve this a main central GUI to control and talk to individual modules such as the CAM designer and material database. These modules can operate independently but are supported and enhanced by the main GUI with its project management and metadata management.

The system's adaptability to external tools is a great feature as well. The source code of the tool is based on open-source software like Python and SQLite, but it is still able to link to other great software like Excel and LabView. Features like the excel beam calculator and BLDC ODrive controller. The Excel beam calculator is a great tool that takes advantage of basic mechanics of materials equations to expedite beam design. And the LabView BLDC ODrive controller is a novel controller that allows LabView users to easily control BLDC motors with a powerful ODrive without the need to switch languages away from LabView.

The system also addresses the issues associated with any tool which is the black box syndrome. It does not eliminate it, but it does reduce it by providing ample amounts of information in the form of metadata. This metadata is automatically generated anytime an output is produced by the tool. This metadata contains information such as tool used, date/time, inputs, and outputs. This makes it easier to retain knowledge and if any confusion is had about an output from the calculator the metadata will be able to clear it up. This combined with the note taking feature prevents most issues from black box syndrome.

Verification showed that the calculations are estimates and should only be used for preliminary work and then finalized with FEA. But the tools are still able to get the user to a ballpark estimate which is better than guessing. The tool also proved its worth through multiple real engineering projects. All projects that used the tool achieved intended purpose with no structural deformation or failure that compromised testing all while expediting timelines. This allowed for the creation of this tool whilst also working on engineering projects. Without the tool the projects would have fallen

behind schedule requiring extra time, meaning this thesis would have been impossible to complete in such a small timeframe.

These outcomes show that the tool was not only a proof of concept but also a critical enabler in the timely completion and success of this thesis. Its ability to reduce iteration inefficiencies, centralize information, and document results seamlessly demonstrates the potential for this engineering tool.

9.1 Future Work

Although the engineering tool developed in this thesis demonstrates a solid solution to the inefficient, decentralization and disorganization problems of the V diagram engineering process, it represents one of the first steps toward a much broader and more capable platform. The modular architecture was intentionally chosen to support continued expansion as additional engineering knowledge and project needs emerge. There are already several future developments that are planned to enhance the capability of the engineering tool.

The first improvement is to migrate the beam calculator from Excel to Python. The current beam calculator works as is but is not easily expandable and it cannot integrate with the rest of the Python system such as the metadata tracker or the material database. Once the beam calculator is in the Python ecosystem it can be hooked up to the material database to allow easier selection of materials. A CAD automation system can easily be added to create 3D CAD models based on the user's beam selection. The algorithm can also be expanded to include Timoshenko PDEs to increase optimization of initial beam design.

Future separate calculators could be added to the system as well including bolt calculators and weld calculators. The bolt calculator would be able to take user force inputs and determine the optimal bolt to use for their design. The weld calculator would be similar in determining the optimal weld type and size based on user force input.

Beyond individual calculators, the engineering tool could evolve into a more comprehensive engineering platform combining the existing convenience of storing the entire tool on a flash drive with an all-encompassing centralized server that would store larger amounts of data. The flash drive would store all active projects and information required for those projects and when the flash drive gained access to the

internet it would be able to wirelessly update, and access information stored on the server. Since the server would have a lot more storage capabilities a large database of engineering information could be stored on it. This would require an advancement of the current knowledge-based engineering where every design created with the system will undergo automatic review comparing the design to a multitude of databases and then returning warnings of violations.

To ensure future tools and calculators are not wrong and do not mislead future engineers they must contain proper documentation showing verification and validation. All tools and calculators should also include disclaimers telling users the limitations of the tool. This system will need to be documented and included within the documentation for this project to notify users that they are required to verify their additions. This is to prevent additions of false systems that may cause harm to other engineers and projects. Current tools, calculators and databases will also need to be updated with disclaimers letting users know that the beam calculator can only calculate stress for long slender beams. A system should also be added to the material database that automatically scans the excel file for origin information.

All together, these future expansions represent a clear and achievable path forward. The engineering tool established in this thesis lays the foundation for a modular and expandible design environment. Continued expansion will allow the system to grow in capability alongside the needs and expertise of its users, evolving into a comprehensive engineering ecosystem capable of supporting the full lifecycle of mechanical design. This engineering tool offers a promising platform that has the potential to expand into a groundbreaking engineering tool that could revolutionize the engineering industry.

10 References

[1] K. Forsberg and H. Mooz, “The Relationship of System Engineering to the Project Cycle,” *INCOSE Int. Symp.*, vol. 1, no. 1, pp. 57–65, Oct. 1991, doi: 10.1002/j.2334-5837.1991.tb01484.x.

[2] “System Engineering for Intelligent Transportation Systems.” Accessed: Oct. 30, 2025. [Online]. Available: https://rosap.ntl.bts.gov

[3] “Roache, P. J., "Validation: Definitions or Descriptions?" *Proceedings of the 3rd Workshop on CFD Uncertainty Analysis*, Lisbon, Portugal, October 2008.

[4] A. G. Lowe and N. W. Hartman, “A Case Study in CAD Design Automation,” *J. Technol. Stud.*, vol. 37, no. 1, Jan. 2011, doi: 10.21061/jots.v37i1.a.1.

[5] J. P. Elm and J. E. Robert, “Integration of Computer-Aided Design and Finite Element Analysis Tools in a Small Manufacturing Enterprise:,” Defense Technical Information Center, Fort Belvoir, VA, June 2003. doi: 10.21236/ADA416600.

[6] S. Zhang and A. Johnson, “The difficulties reported by engineers in searching information,” in *DS 87-6 Proceedings of the 21st International Conference on Engineering Design (ICED 17) Vol 6: Design Information and Knowledge, Vancouver, Canada, 21-25.08. 2017*, 2017, pp. 161–168. Accessed: Oct. 31, 2025. [Online]. Available: https://www.designsociety.org/publication/39779/The+difficulties+reported+by+engineers+in+searching+information

[7] anonuser9, “Haphazard document management and the lack of organization in construction - BETTER CONSTRUCT.” Accessed: Oct. 31, 2025. [Online]. Available: https://betterconstruct.com/haphazard-document-management-and-the-lack-of-organization-in-construction/

[8] D. T. Pham and Y. Yang, “A Genetic Algorithm Based Preliminary Design System,” *Proc. Inst. Mech. Eng. Part J. Automob. Eng.*, vol. 207, no. 2, pp. 127–133, Apr. 1993, doi: 10.1243/PIME_PROC_1993_207_170_02.

[9] G. Zhong, “A novel virtual product modelling framework for design automation in a knowledge-based engineering environment,” PhD Thesis, Birmingham City University, 2022. Accessed: Sept. 05, 2025. [Online]. Available: http://www.open-access.bcu.ac.uk/13992/

[10] “ClearCalcs.” Accessed: Oct. 26, 2025. [Online]. Available: https://calcs.com/

[11] “Structural Analysis and Design | SkyCiv Engineering.” Accessed: Oct. 27, 2025. [Online]. Available: https://skyciv.com/

[12] “WebStructural - Quick and Easy Structural Design.” Accessed: Oct. 27, 2025. [Online]. Available: https://webstructural.com/

[13] “The Engineering ToolBox.” Accessed: Oct. 27, 2025. [Online]. Available: https://www.engineeringtoolbox.com/

[14] “STRIAN—Free Online Structural analysis.” Accessed: Oct. 27, 2025. [Online]. Available: https://structural-analyser.com/

[15] “Calculators for Mechanical Engineers | MechaniCalc.” Accessed: Oct. 27, 2025. [Online]. Available: https://mechanicalc.com/calculators/

[16] “CalcForge.com.” Accessed: Oct. 28, 2025. [Online]. Available: https://CalcForge.com/

[17] “BEAMGURU.COM - Beam Calculator and Frame/Truss Beam Calculator Online (Draws Bending Moment, Shear Force, Axial Force).” Accessed: Oct. 28, 2025. [Online]. Available: https://beamguru.com/

[18] “The Structural Engineering Software Leader,” StruCalc. Accessed: Oct. 28, 2025. [Online]. Available: https://strucalc.com/

[19] “ITAR Compliance 101: Protecting Technical Data.” Accessed: Oct. 27, 2025. [Online]. Available: https://static.carahsoft.com/concrete/files/6817/1692/8829/ITAR_Compliance_Wrapped.pdf

[20] “Here’s what experts say the Amazon Web Services outage reveals about the fragility of the cloud - CBS News.” Accessed: Oct. 27, 2025. [Online]. Available: https://www.cbsnews.com/news/aws-amazon-web-services-outage-fragility-cloud-services/

[21] J. Valinsky, “The internet just had another global outage. Why does this keep happening? | CNN Business,” CNN. Accessed: Oct. 27, 2025. [Online]. Available: https://www.cnn.com/2025/10/20/tech/aws-why-internet-outages-keep-happening

[22] “Simcenter Femap and Nastran bundles comparison,” Siemens Digital Industries Software. Accessed: Oct. 27, 2025. [Online]. Available: https://plm.sw.siemens.com/en-US/simcenter/mechanical-simulation/femap/compare-simcenter-femap-nastran-bundles/

[23] Y. Lin, K. Shea, A. Johnson, J. Coultate, and J. Pears, “A method and software tool for automated gearbox synthesis,” in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2009, pp. 111–121. Accessed: Sept. 05, 2025. [Online]. Available: https://asmedigitalcollection.asme.org/IDETC-CIE/proceedings-abstract/IDETC-CIE2009/111/345304

[24] F. Berardinucci, G. Colombo, M. Lorusso, M. Manzini, W. Terkaj, and M. Urgo, “A learning workflow based on an integrated digital toolkit to support education in manufacturing system engineering,” *J. Manuf. Syst.*, vol. 63, pp. 411–423, Apr. 2022, doi: 10.1016/j.jmsy.2022.04.003.

[25] J. Lampinen, “Cam shape optimisation by genetic algorithm,” *Comput.-Aided Des.*, vol. 35, no. 8, pp. 727–737, 2003.

[26] M. Moret, “Automation of a turbine tip clearance preliminary calculation process,” PhD Thesis, École de technologie supérieure, 2018. Accessed: Sept. 05, 2025. [Online]. Available: https://core.ac.uk/download/pdf/162151648.pdf

[27] Ö. Babur, V. Smilauer, T. Verhoeff, and M. van den Brand, “A Survey of Open Source Multiphysics Frameworks in Engineering,” *Procedia Comput. Sci.*, vol. 51, pp. 1088–1097, Jan. 2015, doi: 10.1016/j.procs.2015.05.273.

[28] J. C. H. Chung *et al.*, “Framework for integrated mechanical design automation,” *Comput.-Aided Des.*, vol. 32, no. 5, pp. 355–365, May 2000, doi: 10.1016/S0010-4485(00)00017-8.

[29] K. Sofias, Z. Kanetaki, C. Stergiou, A. Kantaros, S. Jacques, and T. Ganetsos, “Implementing CAD API Automated Processes in Engineering Design: A Case Study Approach,” *Appl. Sci.*, vol. 15, no. 14, p. 7692, July 2025, doi: 10.3390/app15147692.

[30] N. Batini, “Leveraging artificial neural networks for design automation in the engineering-to-order sector,” 2023, Accessed: Sept. 05, 2025. [Online]. Available: https://www.politesi.polimi.it/handle/10589/225733

[31] S. M. Gumbo, “DEVELOPMENT OF A COMPREHEN- SIVE DESIGN HANDBOOK FOR EN- HANCING MECHANICAL ENGINEER- ING PRACTICES FOR A COMPANY”.

[32] “Free Online Spreadsheet Software: Excel | Microsoft 365.” Accessed: Nov. 04, 2025. [Online]. Available: https://www.microsoft.com/en-us/microsoft-365/excel

[33] “MATLAB.” Accessed: Nov. 04, 2025. [Online]. Available: https://www.mathworks.com/products/matlab.html

[34] “Welcome to Python.org,” Python.org. Accessed: Nov. 04, 2025. [Online]. Available: https://www.python.org/

[35] “TIOBE Index,” TIOBE. Accessed: Nov. 03, 2025. [Online]. Available: https://www.tiobe.com/tiobe-index/

[36] *PyQt6: Python bindings for the Qt cross platform application toolkit*.

[37] “CadQuery 2 Documentation — CadQuery Documentation.” Accessed: Nov. 03, 2025. [Online]. Available: https://cadquery.readthedocs.io/en/latest/#

[38] “sqlite3 — DB-API 2.0 interface for SQLite databases,” Python documentation. Accessed: Nov. 03, 2025. [Online]. Available: https://docs.python.org/3/library/sqlite3.html

[39] “Qt Designer Download for Windows and Mac.” Accessed: Nov. 03, 2025. [Online]. Available: https://build-system.fman.io/qt-designer-download

[40] “Welcome to Python.org,” Python.org. Accessed: Nov. 03, 2025. [Online]. Available: https://www.python.org/

[41] “NumPy.” Accessed: Nov. 03, 2025. [Online]. Available: https://numpy.org/

[42] “pandas - Python Data Analysis Library.” Accessed: Nov. 03, 2025. [Online]. Available: https://pandas.pydata.org/

[43] “Overview - NLopt Documentation.” Accessed: Nov. 03, 2025. [Online]. Available: https://nlopt.readthedocs.io/en/latest/

[44] *casadi: CasADi -- framework for algorithmic differentiation and numeric optimization*. C++, Python. Accessed: Nov. 03, 2025. [MacOS, Microsoft :: Windows, POSIX, Unix]. Available: http://casadi.org

[45] *multimethod: Multiple argument dispatching*. Python. Accessed: Nov. 03, 2025. [OS Independent]. Available: https://github.com/coady/multimethod

[46] *meshlib: 3d processing library*. Python. Accessed: Nov. 03, 2025. [MacOS :: MacOS X, Microsoft :: Windows, POSIX :: Linux]. Available: https://meshlib.io/

[47] *numpy-stl: Library to make reading, writing and modifying both binary and ascii STL files easy.* Python. Accessed: Nov. 03, 2025. [OS Independent]. Available: https://github.com/WoLpH/numpy-stl/

[48] *ezdxf: A Python package to create/manipulate DXF drawings*. Python.

[49] "Matplotlib — Visualization with Python." Accessed: Nov. 03, 2025. [Online]. Available: https://matplotlib.org/

[50] "PyQtGraph - Scientific Graphics and GUI Library for Python." Accessed: Nov. 03, 2025. [Online]. Available: https://www.pyqtgraph.org/

[51] "DUAL Interface Type C and USB A solid state U disk-SD301-SSK." Accessed: Nov. 06, 2025. [Online]. Available: https://www.ssk.cn/Product_Center_details/60.html

[52] "WinPython." Accessed: Nov. 06, 2025. [Online]. Available: https://winpython.github.io/

[53] "ODrive," ODrive. Accessed: Nov. 07, 2025. [Online]. Available: https://odriverobotics.com

[54] R. L. Norton, *Cam Design and Manufacturing Handbook*. Industrial Press Inc., 2009.

[55] "AISC Shapes Database v16.0 | American Institute of Steel Construction." Accessed: Nov. 05, 2025. [Online]. Available: https://www.aisc.org/publications/steel-construction-manual-resources/16th-ed-steel-construction-manual/aisc-shapes-database-v16.0/

[56] J. M. Gere and B. J. Goodno, *Mechanics of materials*, 7. ed., [Nachdr.]. Toronto: Cengage Learning, 2010.

[57] S. Projects, *stijnsprojects/ODrive-LabVIEW*. (Jan. 19, 2024). C++. Accessed: Nov. 13, 2025. [Online]. Available: https://github.com/stijnsprojects/ODrive-LabVIEW

[58] P. J. Roache, *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers, 1998.

[59] “Standard for Verification and Validation in Computational Fluid Dynamics and Heat Transfer - ASME.” Accessed: Nov. 29, 2025. [Online]. Available: https://www.asme.org/codes-standards/find-codes-standards/standard-for-verification-and-validation-in-computational-fluid-dynamics-and-heat-transfer

A. Appendix

A.1 Main Runner Code

```
2
3   import sys
4   import os
5   from pathlib import Path
6   import datetime as dt
7
8   ### Import PyQt Libraries ###
9   from PyQt6.QtWidgets import QWidget
10  from PyQt6 import QtWidgets, uic, QtGui
11  from PyQt6.uic.load_ui import loadUi
12  from PyQt6.QtWidgets import QPushButton, QLineEdit, QLabel, QCompleter
13  from PyQt6.QtCore import Qt, QStringListModel, QTimer
14
15  ### Add the project root to Python path ###
16  project_root = Path(__file__).resolve().parents[1]  # this goes up one
    folder to 'Tool'
17  sys.path.append(str(project_root))
18
19  ### Import custom sub functions, classes and widgets ###
20  from CAMs.CAM_SubClass import CAMWidget
21  from Materials.MatRun import MaterialWidget
22  from ProjectManager.ProMan import CreateProjectFolder,
    LoadProjectFolder, SaveNotesAs, SaveNotes, LoadNotes, ClearNotes
23  from MetadataManager.MetaMan import LoadData, RunData, ClearData
24  import Globals as g
25
26  MainPath=Path(r"E:\Py Engineering Tool\Engineering-Tool\Tool\Main
    GUI\Main.ui") # Path of the MAINGUI ui file
27
28  logo=Path(r"E:\Py Engineering Tool\Engineering-Tool\Tool\Main
    GUI\Media\SierraTechOG.png")
29
30  class MainWindow(QtWidgets.QMainWindow):
31      def __init__(self):
32          super().__init__()
33          loadUi(MainPath, self)
34
35          ### Logo SetUp ###
```

```
36          self.SierraTech: QLabel #label that displays Sierra Tech logo
37          self.SierraTech.setPixmap(QtGui.QPixmap(str(logo)).scaled(225,
    90, Qt.AspectRatioMode.KeepAspectRatio,
    Qt.TransformationMode.SmoothTransformation))
38
39          ### Button Set Up ###
40          self.BeamCall: QPushButton #button that opens Beam GUI
41          self.CAMCall: QPushButton #button that opens CAM GUI
42          self.MaterialCall: QPushButton #button that opens Material GUI
43          self.OpenProjectBot: QPushButton #button that opens project
    folder
44          self.NewProjectBot: QPushButton #button that creates new
    project folder
45          self.SaveNotesAs: QPushButton #button that saves notes into a
    new directory
46          self.LoadNotes: QPushButton #button that loads notes
47          self.SaveNotes: QPushButton #button that saves notes into
    existing directory
48          self.ClearNotes: QPushButton #button that clears the notes
    textbox
49          self.ImportData: QPushButton #button that imports metadata
50          self.ClearData: QPushButton #button that clears data from
    metadata textbox
51          self.RunData: QPushButton #button that calls metadata and then
    opens correct tool and then inputs data
52


53          ### Label Set Up ###
54          self.ActiveFile: QLabel #Label that displays active project
    file directory
55          self.ProjectTitle: QLabel #Label that displays the active
    project title
56          self.Metadata: QLabel #Label that displays the Metadata
57
58          ### Center Widget Setup ###
59          self.MainStack: QtWidgets.QStackedWidget #stacked widget that
    holds different GUIs
60
61          self.cam_widget=CAMWidget() #define CAM widget
62          self.mat_widget=MaterialWidget() #define Material widget
63
```

```
64          self.MainStack.addWidget(self.cam_widget) #add CAM widget to
    stack
65          self.MainStack.addWidget(self.mat_widget) #add Material widget
    to stack
66
67          ### Call Buttons ###
68
69          self.CAMCall.clicked.connect(lambda:
    self.MainStack.setCurrentWidget(self.cam_widget))
70          self.MaterialCall.clicked.connect(lambda:
    self.MainStack.setCurrentWidget(self.mat_widget))
71
72          self.NewProjectBot.clicked.connect(lambda: CreateDisplay(self))
73          self.OpenProjectBot.clicked.connect(lambda: LoadDisplay(self))
74
75          self.SaveNotesAs.clicked.connect(lambda: SaveNotesAs(self))
76          self.LoadNotes.clicked.connect(lambda: LoadNotes(self))
77          self.SaveNotes.clicked.connect(lambda: SaveNotes(self))
78          self.ClearNotes.clicked.connect(lambda: ClearNotes(self))
79
80          self.ClearData.clicked.connect(lambda: ClearData(self))
81          self.ImportData.clicked.connect(lambda: LoadData(self))
82          self.RunData.clicked.connect(lambda: RunData(self))
83
84
85  ### Functions ###
86
87  # Function to display active project directory on new project creation
88  def CreateDisplay(self):
89      ProjectPath,g.ProjectName=CreateProjectFolder(self)
90
91      if ProjectPath:
92          self.ActiveFile.setText(f"Active Project Directory:
    {str(ProjectPath)} ")
93          self.ProjectTitle.setText(f"Project: {g.ProjectName}")
94
95  # Function to display active project directory on project open
96  def LoadDisplay(self):
97      result = LoadProjectFolder(self)
98      if result:
99          # LoadProjectFolder currently returns (path, msg)
100         ProjectPath, msg = result
```

```
101         if msg:
102             QtWidgets.QMessageBox.warning(self, "Invalid Project", msg)
103         else:
104             self.ActiveFile.setText(f"Active Project Directory:
    {str(ProjectPath)} ")
105             g.ProjectName = Path(ProjectPath).name
106 
107             self.ProjectTitle.setText(f"Project: {g.ProjectName}")
108 

109 ### Run and Show the GUI ###
110 if __name__ == "__main__":
111     import sys
112     app = QtWidgets.QApplication(sys.argv)
113     Widget = MainWindow()
114     Widget.show()
115     sys.exit(app.exec())
116 
```

A.2 Project Manager Code

```
from pathlib import Path
from PyQt6.QtWidgets import QFileDialog, QInputDialog
from PyQt6 import QtWidgets
import os
import datetime as dt

import sys

project_root = Path(__file__).resolve().parents[1]
sys.path.append(str(project_root))

import Globals as g

def ChooseFolder(parent=None, title="Select Project Folder"):

    folder = QFileDialog.getExistingDirectory(parent, title)

    return folder


def CreateProjectFolder(parent=None):
    folder = ChooseFolder(parent, "Select Location for New Project")
    if not folder:
        return None, None

    ### Create the Standard Folder Layout ###

    ProjectName, ok = QtWidgets.QInputDialog.getText(parent, "Project Name", "Enter Project Name:")
    if not ok or not ProjectName.strip():
        return None, None

    ProjectPath = Path(folder) / ProjectName.strip()
    ProjectPath.mkdir(parents=True, exist_ok=True)

    g.ProjectDirectory=str(ProjectPath) # update project directory file global variable

    folders= ["Presentations",
        "Order Forms",
```

```
        "Documentation/Templates",
        "Calculations",
        "CAD/Layouts",
        "CAD/Assemblies",
        "CAD/Parts",
        "CAD/Final Design",
        "CAD/Drawings",
        "Media",
        "Tests",
        "Electrical + DAQ",
        "Notes",
        "Metadata",
        "Archive",
        "Analysis/Scripts",
        "Analysis/Figures",
        "Software"
    ]
    for f in folders:
        (ProjectPath / f).mkdir(parents=True, exist_ok=True)

    ### Create the ProjectConfig File ###
    configfile = ProjectPath / "ProjectConfig.txt"
    with open(configfile, "w") as f:
        f.write(f"Project Name = {ProjectName}\n")
        f.write(f"Created = {dt.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")

    return ProjectPath, ProjectName



def LoadProjectFolder(parent=None):
    folder = ChooseFolder(parent, "Select Project Folder to Load")
    if not folder:
        return None

    ProjectPath=Path(folder)
    configfile = ProjectPath/ "ProjectConfig.txt"

    g.ProjectDirectory=str(ProjectPath) # update project directory file global variable

```

```
192     msg=None
193     if not configfile.exists():
194         msg="This is not a valid project folder"
195 
196     return ProjectPath, msg
197 

198 ### Note Creation and Management ###
199 
200 def SaveNotesAs(self,parent=None):
201 
202     if not g.ProjectDirectory:
203         QtWidgets.QMessageBox.warning(self, "No Project Selected",
    "Please select or create a project first.")
204         return # If a project folder is not selected or created prevent
    user from useing this feature
205 
206 
207     NotesName, ok = QtWidgets.QInputDialog.getText(parent, "Notes File
    Name", "Enter Notes File Name:")
208     if not ok or not NotesName.strip():
209         return  # user cancelled or gave empty name
210 
211     NotesPath=Path(str(g.ProjectDirectory)) / "Notes"
212     NotesPath.mkdir(exist_ok=True)
213     NotesFile = NotesPath / f"{NotesName}.txt"
214 
215     if NotesFile.exists(): # Check to see if the file name already
    exists
216         reply = QtWidgets.QMessageBox.question(
217             parent,
218             "Overwrite File?",
219             f"A note named '{NotesName}.txt' already exists.\nDo you
    want to overwrite it?",
220             QtWidgets.QMessageBox.StandardButton.Yes |
    QtWidgets.QMessageBox.StandardButton.No
221         )
222         if reply == QtWidgets.QMessageBox.StandardButton.No:
223             return
224 
225     text= self.Notes.toPlainText()
226 
```

```
227     with open(NotesFile, "w", encoding="utf-8") as f:
228         f.write(f"Project Name = {g.ProjectName}\n")
229         f.write(f"Notes Created = {dt.datetime.now().strftime('%Y-%m-%d
    %H:%M:%S')}\n")
230         f.write("\n")
231         f.write(text)
232 
233     g.CurrentNotePath=str(NotesFile)
234     QtWidgets.QMessageBox.information(self, "Notes Saved", f"Notes
    saved to:\n{NotesFile}")
235 
236 def SaveNotes(self,parent=None):
237     if not g.ProjectDirectory:
238         QtWidgets.QMessageBox.warning(self, "No Project Selected",
    "Please select or create a project first.")
239         return
240 
241     if not g.CurrentNotePath:
242         QtWidgets.QMessageBox.warning(self, "No Active Note", "Please
    use 'Save Notes As' to create or open a note first.")
243         return
244 
245 
246     text = self.Notes.toPlainText()
247     with open(g.CurrentNotePath, "w", encoding="utf-8") as f:
248         #f.write(f"Project Name = {g.ProjectName}\n")
249         #f.write(f"Modified = {dt.datetime.now().strftime('%m-%d-%Y
    %H:%M:%S')}\n\n")
250         f.write(text)
251 
252     QtWidgets.QMessageBox.information(self, "Notes Saved", f"Changes
    saved to:\n{g.CurrentNotePath}")
253 
254 
255 def ClearNotes(self):
256     self.Notes.clear()
257     return
258 
259 def LoadNotes(self):
260 
261     if not g.ProjectDirectory:
```

```
262         QtWidgets.QMessageBox.warning(self, "No Project Selected",
    "Please select or create a project first.")
263         return # If a project folder is not selected or created prevent
    user from useing this feature
264 
265     NotesFolder = Path(str(g.ProjectDirectory)) / "Notes"
266     FilePath, _ = QtWidgets.QFileDialog.getOpenFileName(self, "Open
    Notes", str(NotesFolder), "Text Files (*.txt)")
267 
268     if FilePath:
269         with open(FilePath, "r", encoding="utf-8") as f:
270             content = f.read()
271         self.Notes.setPlainText(content)
272 
273     g.CurrentNotePath = FilePath
274 
275 
```

A.3 Metadata Code

```
276 from pathlib import Path
277 from PyQt6.QtWidgets import QLineEdit, QComboBox, QMessageBox
278 from PyQt6 import QtWidgets
279 import Globals as g
280 import json, datetime as dt
281
282 def ScanInputs(ui, ToolName=None):
283
284     inputs={}
285     for widget in ui.findChildren(QLineEdit):
286         name=widget.objectName()
287         value=widget.text().strip()
288         if value:
289             inputs[name] = value
290
291     for widget in ui.findChildren(QComboBox):
292         name=widget.objectName()
293         value=widget.currentText().strip()
294         if value:
295             inputs[name] = value
296
297     return {
298         "Tool": ToolName or ui.objectName(),
299         "WidgetObjectName": ui.objectName(),
300         "Inputs": inputs
301     }
302
303 def SaveData(ui, ToolName = None, outputs=None):
304
305     if not g.ProjectDirectory:
306         QtWidgets.QMessageBox.warning(ui, "No Project Selected",
    "Please select or create a project first.")
307         return # If a project folder is not selected or created prevent
    user from useing this feature
308
309     MetaFolder = Path(g.ProjectDirectory) / "Metadata"
310     MetaFolder.mkdir(exist_ok=True)
311
312     MetaPath=MetaFolder/f"{ToolName}_{dt.datetime.now().strftime('%m-
    %d-%Y_%H%M%S')}.json"
```

```
313
314     if not ToolName:
315         ToolName = ui.objectName()
316
317     data={
318         "Tool": ToolName,
319         "WidgetObjectName": ui.objectName(),
320         "Timestamp": dt.datetime.now().strftime("%m-%d-%Y %H:%M:%S"),
321         "Inputs": g.Metadata.get("Inputs",{}),
322         "Outputs": outputs or {}
323     }
324
325     with open(MetaPath, "w", encoding="utf-8") as f:
326         json.dump(data, f, indent=4)
327
328     msgBox = QMessageBox()
329     msgBox.setText(f"Metadata saved to{MetaPath}") # want to put
    message box that says where the metadata was saved and that it was
    indeed saved
330     msgBox.exec()
331
332     return MetaPath
333
334 def LoadData(self):
335
336     if not g.ProjectDirectory:
337         QtWidgets.QMessageBox.warning(self, "No Project Selected",
    "Please select or create a project first.")
338         return # If a project folder is not selected or created prevent
    user from useing this feature
339
340     DataFolder = Path(str(g.ProjectDirectory)) / "Metadata"
341     FilePath, _ = QtWidgets.QFileDialog.getOpenFileName(self, "Open
    Data", str(DataFolder), "Json Files (*.json)")
342
343     if not FilePath:
344         return
345
346     with open(FilePath, "r", encoding="utf-8") as f:
347         data = json.load(f)
348
349     self.Metadata.setWordWrap(True)
```

```
350     self.Metadata.setText(json.dumps(data,indent=4))
351     g.MetadataPath = FilePath
352 
353 def RunData(mainui):
354 
355     FilePath = g.MetadataPath
356     if not FilePath:
357         return
358 
359 
360 
361     with open(FilePath, "r", encoding="utf-8") as f:
362         data = json.load(f)
363 
364     ToolName = data["Tool"]
365     WidgetName = data["WidgetObjectName"]
366 
367     TargetWidget = mainui.findChild(QtWidgets.QWidget, WidgetName)
368 
369     if not TargetWidget:
370         QtWidgets.QMessageBox.warning(mainui, "Metadata Load Error",
371                                       f"Could not find widget
    '{WidgetName}' for tool '{ToolName}'.")
372         return
373 
374 
375     mainui.MainStack.setCurrentWidget(TargetWidget)
376 
377     for name, value in data["Inputs"].items():
378         textbox = TargetWidget.findChild(QtWidgets.QLineEdit, name)
379         if textbox:
380             textbox.setText(value)
381         combo = TargetWidget.findChild(QtWidgets.QComboBox, name)
382         if combo:
383             index = combo.findText(value)
384             if index >= 0:
385                 combo.setCurrentIndex(index)
386 
387 def ClearData(self):
388     self.Metadata.clear()
389     return
```

A.4 Material Database Runner Code

```
import sys
import os
from PyQt6.QtWidgets import QWidget
from PyQt6 import QtWidgets, uic
from PyQt6.uic.load_ui import loadUi
from pathlib import Path
from PyQt6.QtWidgets import QPushButton, QLineEdit, QLabel, QCompleter
from PyQt6.QtCore import Qt, QStringListModel, QTimer
from .MatMan import matsearch, MatWrite, ExcelFileName, readmat

MatPath = Path(__file__).parent / "MaterialWidget.ui"

#MatPath=Path(r"E:\Py Engineering Tool\Engineering-Tool\(1) Tool\Materials\MaterialWidget.ui")

class MaterialWidget(QtWidgets.QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        loadUi(MatPath, self)

        self.Search: QPushButton #button that searches database for material based on search bar text

        self.MaterialAdd: QPushButton #button that adds material to database based on excel file path from filesearch button
        self.FileSearch: QPushButton #button that finds excel file

        self.SearchBar: QLineEdit #text input for material search
        self.FileBar: QLineEdit #line to display file path

        self.Results: QLabel #label that displays search results

        # Hook up buttons
        self.Search.clicked.connect(self.search_material)
        self.MaterialAdd.clicked.connect(self.add_material)
        self.FileSearch.clicked.connect(self.ExcelName)

        # Hook up completer for search bar
        self.model = QStringListModel()
        self.completer = QCompleter(self.model, self)
```

```
427         self.completer.setCaseSensitivity(Qt.CaseSensitivity.CaseInsens
    itive)
428         self.completer.setCompletionMode(QCompleter.CompletionMode.Popu
    pCompletion)
429         self.completer.setFilterMode(Qt.MatchFlag.MatchContains)

430         self.SearchBar.setCompleter(self.completer)
431 
432         # Timer to delay search updates
433         self._search_timer = QTimer()
434         self._search_timer.setSingleShot(True)
435         self._search_timer.timeout.connect(lambda:
    self.update_completer_matches(self.SearchBar.text()))
436 
437         self.SearchBar.textChanged.connect(lambda _:
    self._search_timer.start(200))
438 
439 # When the user picks something from the completer
440         self.completer.activated.connect(self.on_completion_selected)
441 
442         # text change triggers search update
443         #self.SearchBar.textChanged.connect(self.update_completer_match
    es)
444 
445     def update_completer_matches(self, text):
446         text=text.strip()
447         if len(text) < 2:
448             self.model.setStringList([])
449             return
450 
451         matches = matsearch(text)
452 
453         # prevent recursive textChanged calls while updating
454         self.SearchBar.blockSignals(True)
455         self.model.setStringList(matches)
456         self.SearchBar.blockSignals(False)
457 
458     def on_completion_selected(self, text):
459         # Optional: do something when a completion is chosen
460         # e.g. auto-run the search or just leave it
461         self.SearchBar.setText(text)
462 
```

```
    def GhostCompleter(self): # No Longer Used but was useful at some point is kept around for reference
        text = self.SearchBar.text()
        if len(text)<2: #will not start looking for matches until 2 characters are typed
            return

        matches = matsearch(text)

        completer = QCompleter(matches)
        completer.setCaseSensitivity(Qt.CaseSensitivity.CaseInsensitive)
        self.SearchBar.setCompleter(completer)


    def search_material(self): # Needs to search for material and display results in results label
        query = self.SearchBar.text().strip() # get text from search bar

        if not query:
            return

        results = readmat(query)
        self.Results.clear() # clear the Results label
        if not results:
            self.Results.setText(f"No results found for '{query}'.")
            return

        try:
            formatted = "\n".join([f"{col}: {val}" for col, val in zip(
                ["ID", "Name", "Density (kg/m³)", "Ultimate Strength (Pa)", "Yield Strength (Pa)", "Young's Modulus (Pa)",
                 "Poisson's Ratio", "Shear Modulus (Pa)", "Shear Strength (Pa)", "Electrical Resistivity (Ohm·m)",
                 "Coefficient of Thermal Expansion (1/K)", "Specific Heat (J/(kg·K))", "Thermal Conductivity (W/(m·K))"],
                results[0]
            )])
            self.Results.setText(formatted)

        except Exception as e:
```

```
497             print("Error formatting results:", e)
498             self.Results.setText(f"Error displaying results: {e}")
499 
500     def add_material(self):
501         path = self.FileBar.text().strip()
502         self.FileBar.setText("Successfully added material!")
503         if not path:
504             self.FileBar.setText("no file selected.")
505             return
506         MatWrite(path)
507 
508     def ExcelName(self):
509         file=ExcelFileName(self)
510         if file:
511             self.FileBar.setText(file)
512         else:
513             self.FileBar.setText("no file selected.")
514 
515 if __name__ == "__main__":
516     import sys
517     app = QtWidgets.QApplication(sys.argv)
518     Widget = MaterialWidget()
519     Widget.show()
520     sys.exit(app.exec())
```

A.5 Material Database Manager Code

```
import sqlite3
import os
import pandas as pd
from pathlib import Path
from difflib import get_close_matches
from typing import List, Optional, Dict, Any

# Directory path of the database
db_path=Path(__file__).parent / "MaterialDatabase.db"

#db_path = Path(r"E:\Py Engineering Tool\Engineering-Tool\(1) Tool\Materials\MaterialDatabase.db")  #Old Path


### List of functions in MatMan.py: ###
# initialize_database(db_path: Path = db_path) # initializes the database (NOT FOR EXTERNAL USE)

# readmat        # Reads material properties from database based on material name input

# matexists      # checks to see if the material name exists in the database

# allmats        # Returns a list of all materials (NOT FOR EXTERNAL USE)

# matsearch      # Returns a list of close matches based on text input

# strstrip       # Simplifies strings (NOT FOR EXTERNAL USE)

# excelSearch    # Searches Excel file for properties and returns a dictionary (NOT FOR EXTERNAL USE)

# MatWrite       # Writes properties from Excel file to database

# ExcelFileName  # Opens file dialog to select Excel File

def initialize_database(db_path: Path = db_path): # Initializes the database and creates the materials table if it does not exist
```

```
    with sqlite3.connect(str(db_path)) as con:
        cur = con.cursor()

        cur.execute("""
        CREATE TABLE IF NOT EXISTS materials (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT UNIQUE,
            density REAL,
            ultimate_strength REAL,
            yield_strength REAL,
            youngs_modulus REAL,
            poisson_ratio REAL,
            shear_modulus REAL,
            shear_strength REAL,
            electrical_resistivity REAL,
            CTE REAL,
            specific_heat REAL,
            thermal_conductivity REAL
        )
        """)

def readmat(name, db_path: Path = db_path): # Reads material properties from database and returns them as a list of tuples
    if not db_path.exists():
        initialize_database()

    with sqlite3.connect(str(db_path)) as con:
        cur = con.cursor()
        cur.execute("SELECT * FROM materials WHERE name = ?", (name,))
        return cur.fetchall()

def matexists(name, db_path: Path = db_path): # Checks to see if material exists in database
    if not db_path.exists():
        initialize_database()
    with sqlite3.connect(str(db_path)) as con:
        cur = con.cursor()
        cur.execute("SELECT 1 FROM materials WHERE name = ? LIMIT 1", (name,))
        return cur.fetchone() is not None # Returns True if material exists, False otherwise

```

```
591 def allmats(db_path: Path = db_path): # Returns a list of all materials
    in the database
592     if not db_path.exists():
593         initialize_database()
594     with sqlite3.connect(str(db_path)) as con:
595         cur = con.cursor()
596         cur.execute("SELECT name FROM materials")
597         return [row[0] for row in cur.fetchall()]
598 
599 def matsearch(Search, db_path: Path = db_path): # Searches for material
    in database and returns a list of close matches
600     if not db_path.exists():
601         initialize_database()
602     limit=10
603     names=allmats(db_path)
604     searchLower = Search.lower()
605 
606     #substring matches anywhere in the name
607     substringMatches = [n for n in names if searchLower in n.lower()]
608 
609     #fuzzy matches (get_close_matches) if needed
610     if len(substringMatches) < limit:
611         fuzzy_matches = get_close_matches(Search, names, n=limit,
    cutoff=0.5)
612         # merge lists without duplicates
613         substringMatches.extend(m for m in fuzzy_matches if m not in
    substringMatches)
614 
615     return substringMatches[:limit]
616 
617 def strstrip(s: str):
618     return s.strip().lower()
619 
620 def excelSearch(excel_path): # Searches an excel file for material
    properties and returns them as a dictionary
621     data=pd.read_excel(excel_path, header=None)
622 
623     words = {
624         'density': ['density'],
```

```
625         'ultimate_strength': ['tensile strength, ultimate', 'ultimate
    tensile strength'],
626         'yield_strength': ['tensile strength, yield', 'yield
    strength'],
627         'youngs_modulus': ['modulus of elasticity', 'youngs modulus'],
628         'poisson_ratio': ['poissons ratio', "poisson's ratio"],
629         'shear_modulus': ['shear modulus'],
630         'shear_strength': ['shear strength'],
631         'electrical_resistivity': ['electrical resistivity'],
632         'CTE': ['cte', 'coefficient of thermal expansion', 'cte,
    linear'],
633         'specific_heat': ['specific heat capacity', 'specific heat'],
634         'thermal_conductivity': ['thermal conductivity']
635     }
636 
637     results: Dict[str, Any] = {}
638 
639     results.setdefault('name', data.iat[0,0])  # Assuming the title is
    in the first cell
640 
641     # the idea behind this for loop is to go through each row of excel
    and at each row go through each property in the words disctionary and
    see if any of the variants are in the first column of excel
642     for _, row in data.iterrows(): # Iterate through each row in the
    DataFrame
643         a=row[0] # First column of excel
644         b=row[1] # Second column of excel
645 
646         # Check to see if first coumn contains labels and then match to
    words dictionary and write results to results
647         if isinstance(a, str): # Ensure 'a' is a string to avoid errors
648             a_stripped = strstrip(a) # get rid of spaces and cases
649             for key, variants in words.items(): # Loop through each
    property and its variants, uses the keys in the word dictionary to
    cycle through material properties to return values
650                 if any(strstrip(variant) in a_stripped for variant in
    variants): # Check if any variant matches the stripped string
651                     results.setdefault(key, b) # Store the value in
    results dictionary
652 
653     return results
654 
```

```
655 def MatWrite(excel_path, db_path: Path = db_path): # Writes material
    properties from excel file to database
656         if not db_path.exists():
657             initialize_database()
658         epath=Path(excel_path)
659         if not epath.exists():
660             raise FileNotFoundError(f"Excel file not found: {epath}")
661         properties=excelSearch(epath)
662         name=properties.get('name')
663         SqlInsert=("INSERT INTO materials (name, density,
    ultimate_strength, yield_strength, youngs_modulus, poisson_ratio,
    shear_modulus, shear_strength, electrical_resistivity, CTE,
    specific_heat, thermal_conductivity) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?,
    ?, ?, ?)")
664         values=(
665             name,
666             properties.get('density'),
667             properties.get('ultimate_strength'),
668             properties.get('yield_strength'),
669             properties.get('youngs_modulus'),
670             properties.get('poisson_ratio'),
671             properties.get('shear_modulus'),
672             properties.get('shear_strength'),
673             properties.get('electrical_resistivity'),
674             properties.get('CTE'),
675             properties.get('specific_heat'),
676             properties.get('thermal_conductivity')
677         )
678         if matexists(name, db_path):
679             return None
680         else:
681             with sqlite3.connect(str(db_path)) as con:
682                 cur = con.cursor()
683                 cur.execute(SqlInsert, values)
684                 return cur.lastrowid
685
686 def ExcelFileName(parent=None):
687     from PyQt6.QtWidgets import QFileDialog
688
689     filename, _ = QFileDialog.getOpenFileName(
690         parent,
```

```
691         "Select Excel Material File",
692         "",
693         "Excel Files (*.xlsx *.xls);;All Files (*)"
694     )
695     return filename
696 
```

A.6 CAM Designer Runner Code

```
import sys
import os
from PyQt6 import QtWidgets, uic
from PyQt6.uic.load_ui import loadUi
from pathlib import Path
from PyQt6.QtWidgets import QPushButton
import Globals as g


#print("Current working directory:", os.getcwd())
CamPath=Path(r"E:\Py Engineering Tool\Engineering-Tool\Tool\CAMs\QT\CAM_Desinger_Widget.ui")

class CAMWidget(QtWidgets.QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)

        #CamPath2=Path(__file__).parent/"CAM_Designer_GUI.ui"
        loadUi(CamPath, self)

        self.GraphBot: QPushButton
        self.STEPBot: QPushButton
        self.CADBot: QPushButton

        ### Action Items ###

        # Graphing button
        from .CAM_Func import CAMPlot
        self.GraphBot.clicked.connect(lambda: CAMPlot(self)) # graph button must be called GraphBot
        self.CADBot.clicked.connect(self.STLSave) # STL button must be called STLBot
        #self.STEPBot.clicked.connect(self.STPSave) # STEP Button must be called STEPBot

    def STLSave(self):
        from .CAM_Func import STLFileName, CADFileName

        if not g.ProjectDirectory:
```

```
731             QtWidgets.QMessageBox.warning(self, "No Project Selected",
    "Please select or create a project first.")
732             return # If a project folder is not selected or created
    prevent user from useing this feature
733 
734         filename = CADFileName(self)
735         if filename:
736             print("Save to:", filename)
737             from .CAM_Func import STLGen, CAMPlot, STPGen
738             angle,Y,Thickness=CAMPlot(self)
739             STPGen(angle, Y, Thickness,filename)
740 
741         # Do something with filename, like saving a file
742         else:
743             print("No file selected.")
744 
745     def STPSave(self):
746         from .CAM_Func import STPFileName
747 
748         filename = STPFileName(self)
749         if filename:
750             print("Save to:", filename)
751             from .CAM_Func import STPGen, CAMPlot
752             angle,Y,Thickness=CAMPlot(self)
753             STPGen(angle, Y, Thickness,filename)
754 
755         # Do something with filename, like saving a file
756         else:
757             print("No file selected.")
758 
759 
760 
761 
762 if __name__ == "__main__":
763     import sys
764     app = QtWidgets.QApplication(sys.argv)
765     Widget = CAMWidget()
766     Widget.show()
767     sys.exit(app.exec())
768 
769 
```

A.7 CAM Designer Manager Code

```
770 import numpy as np
771 import matplotlib.pyplot as plt
772 from stl import mesh
773 import Globals as g
774 

775 ### List of functions in CAM_Func.py: ###
776 # from CAM_Func import
    Harmonic,Linear,Parabolic,Cycloidal,CAMPolPlot,STLGen
777 
778 # Harmonic      input:   (TR, D, BR, Tri)        return: (angle , Y)
779 # Linear        input:   (TR, D, BR, Tri)        return: (angle , Y)
780 # Parabolic     input:   (TR, D, BR, Tri)        return: (angle , Y)
781 # Cycloidal     input:   (TR, D, BR, Tri)        return: (angle , Y)
782 # CAMPolPlot    input:   (angle, Y)
783 # STLGen        input:   (angle, Y, Thickness)
784 
785 ## Variables
786 # TR = Travel Radius
787 # BR = Base Radius which is the smallest radius of the CAM
788 # D  = Array containing angles of travel for D1, D2, D3, and D4
789 # D1 = Angle for rising to travel radius plus the base radius (Starts
    at 0 degrees)
790 # D2 = Angle for dwelling at the top
791 # D3 = angle for falling back down to base radius
792 # D4 = Angle for dwelling at the basradius (Should always end at 360)
793 # Tri= Number of triangles generated in STL file
794 # Thickness = thickness of the CAM
795 
796 ### Example CAM PARAMETERS ###
797 #BR = 1.3             # Base Radius
798 #TR = 3.0             # Top Radius
799 #Thickness = 0.35     # Thickness of CAM
800 #D1, D2, D3, D4 = 87, 100, 200, 360  # Angular Segments (Degrees)
801 #Tri = 250           # Number of points per segment
802 #Df=[0,D1,D2,D3,D4]
803 
804 ### Functions ###
805 
806 def Harmonic(TR, D, BR, Tri):
```

```
    rdfd = np.array([1, 0, 1, 0])
    rf   = np.array([0, 0, 1, 0])
    rd   = np.array([0, 1, 0, 0])

    angle = []
    Y = []

    for i in range(4):
        d_start = D[i]
        d_end = D[i + 1]
        d_prev = D[i - 1] if i > 0 else 0  # Avoid index error

        if i==2:
            theta=np.linspace(D[2],D[3],Tri)
            y=(TR/2)*(1+np.cos((np.pi*(theta-D[2]))/(D[3]-D[2]))) + BR
            #theta=theta/180*np.pi

        else:
            theta = np.linspace(d_start, d_end, Tri)
            delta = d_end - d_start
            phase = (theta - d_prev * rf[i]) / delta if delta != 0 else
0
            y = (TR / 2) * (1 - np.cos(np.pi * phase)) * rdfd[i] + BR +
TR * rd[i]

        angle.append(np.radians(theta))
        Y.append(y)

    return Y, angle

def Linear(TR, D, BR, Tri):
    # TR: Travel Radius, D: Degree vector containing D1,D2,D3,D4,
    # BR: Base Radius, Tri: number of triangles
    rdfd = np.array([1, 0, 1, 0])  # indicators (Rise/fall/dwell/dwell)
    rf   = np.array([0, 0, 1, 0])  # Fall indicator
    rd   = np.array([0, 1, 0, 0])  # Previous dwell (adds BR shift)

    angle = []
    Y = []

    for i in range(4):
        d_start = D[i]
```

```
        d_end = D[i + 1]
        delta = d_end - d_start

        theta = np.linspace(d_start, d_end, Tri)
        phi = (theta - d_start) / delta  # Normalized range [0,1]

        if rf[i] == 0:
            # Rise: y = h * phi
            y = TR * phi * rdfd[i] + BR + TR * rd[i]
        else:
            # Fall: y = h * (1 - phi)
            y = TR * (1 - phi) + BR

        angle.append(np.radians(theta))
        Y.append(y)

    return Y, angle

def Parabolic(TR, D, BR, Tri):
    # TR: Travel Radius, D: Degree vector containing D1,D2,D3,D4,
    # BR: Base Radius, Tri: number of triangles

    #Indicators (Rise, Dwell, Fall, Dwell)
    rdfd = np.array([1, 0, 1, 0])  # Rise/Fall/Dwell flags
    rf   = np.array([0, 0, 1, 0])  # Fall indicator
    rd   = np.array([0, 1, 0, 0])  # Previous dwell

    angle = []
    Y = []

    for i in range(4):
        d_start = D[i]
        d_end = D[i + 1]
        delta = d_end - d_start

        theta = np.linspace(d_start, d_end, Tri)
        beta = delta
        phi = (theta - d_start) / beta  # Normalize to [0,1]

        if rf[i] == 0:
            # Rise: y = h * (2*phi - phi^2)
            y = TR * (2 * phi - phi ** 2) * rdfd[i] + BR + TR * rd[i]
```

```
        else:
            # Fall: y = h * (1 - 2*phi + phi^2)
            y = TR * (1 - 2 * phi + phi ** 2) + BR

        angle.append(np.radians(theta))
        Y.append(y)

    return Y, angle

def Cycloidal(TR,D,BR,Tri):
    rdfd = np.array([1, 0, 1, 0])
    rf   = np.array([0, 0, 1, 0])
    rd   = np.array([0, 1, 0, 0])

    angle = []
    Y = []

    for i in range(4):
        d_start = D[i]
        d_end = D[i + 1]
        d_prev = D[i - 1] if i > 0 else 0  # Avoid index error

        theta = np.linspace(d_start, d_end, Tri)
        delta = d_end - d_start
        phase = (theta - d_prev * rf[i]) / delta if delta != 0 else 0
        if rf[i]==0:

            y = TR*(phase-(1/(2*np.pi))*np.sin(2*np.pi*phase)) *
rdfd[i] + BR + TR * rd[i]
        else:
            beta = D[3] - D[2]
            theta_local = theta - D[2]
            y= TR * (1 - (theta_local / beta - (1 / (2 * np.pi)) *
np.sin(2 * np.pi * theta_local / beta)))+BR
        angle.append(np.radians(theta))
        Y.append(y)

    return Y, angle

#angle=Harmonic(TR,Df,BR,Tri)[1]
#Y=Harmonic(TR,Df,BR,Tri)[0]

```

```
929 def CAMPolPlot(angle, Y):
930     fig2, plot = plt.subplots(subplot_kw={'projection': 'polar'})
931     plot.plot(angle[0], Y[0])
932     plot.plot(angle[1], Y[1])
933     plot.plot(angle[2], Y[2])
934     plot.plot(angle[3], Y[3])
935     plot.set_rticks([0.5, 1, 1.5, 2])  # type: ignore # Less radial
    ticks
936     plt.show()
937 
938 def CAMPolPlotRise(angle, Y):
939     fig2, plot = plt.subplots(subplot_kw={'projection': 'polar'})
940     plot.plot(angle[0], Y[0])
941     plot.plot(angle[1], Y[1])
942     #plot.plot(angle[2], Y[2])
943     #plot.plot(angle[3], Y[3])
944     plot.set_rticks([0.5, 1, 1.5, 2])  # type: ignore # Less radial
    ticks
945     #plt.show()
946 
947 def CAMPolPlotFall(angle, Y):
948     fig2, plot = plt.subplots(subplot_kw={'projection': 'polar'})
949     #plot.plot(angle[0], Y[0])
950     #plot.plot(angle[1], Y[1])
951     plot.plot(angle[2], Y[2])
952     plot.plot(angle[3], Y[3])
953     plot.set_rticks([0.5, 1, 1.5, 2])  # type: ignore # Less radial
    ticks
954     #plt.show()
955 #CAMPolPlot(angle,Y)
956 
957 def STLGen(angle, Y, Thickness,filename):
958     # Convert Polar to Cartesian
959     x1, y1 = Y[0] * np.cos(angle[0]), Y[0] * np.sin(angle[0])
960     x2, y2 = Y[1] * np.cos(angle[1]), Y[1] * np.sin(angle[1])
961     x3, y3 = Y[2] * np.cos(angle[2]), Y[2] * np.sin(angle[2])
962     x4, y4 = Y[3] * np.cos(angle[3]), Y[3] * np.sin(angle[3])
963 
964     # Combine all profile points (excluding center)
965     profile_x = np.concatenate((x1, x2, x3, x4))
966     profile_y = np.concatenate((y1, y2, y3, y4))
967     num_outer = len(profile_x)
```

```
968
969     # Create Z values
970     z_bot = np.zeros(num_outer)
971     z_top = np.ones(num_outer) * Thickness
972
973     # Add center points at beginning of each layer
974     points_bottom = np.vstack((
975         [[0, 0, 0]],
976         np.stack((profile_x, profile_y, z_bot), axis=-1)
977     ))
978     points_top = np.vstack((
979         [[0, 0, Thickness]],
980         np.stack((profile_x, profile_y, z_top), axis=-1)
981     ))
982
983     # Full vertex list
984     vertices = np.vstack((points_bottom, points_top))
985
986     # === Triangle Faces ===
987     faces = []
988
989     # Bottom face (fan from center index 0)
990     for i in range(1, num_outer):
991         faces.append([0, i, i + 1])
992     faces.append([0, num_outer, 1])  # Close loop
993
994     # Top face (fan from center index num_outer + 1)
995     top_center = num_outer + 1
996     for i in range(1, num_outer):
997         faces.append([top_center, top_center + i + 1, top_center + i])
998     faces.append([top_center, top_center + 1, top_center +
    num_outer])  # Close loop
999
1000      # Side walls (between bottom and top outer rings)
1001      for i in range(1, num_outer):
1002          b1 = i
1003          b2 = i + 1
1004          t1 = b1 + num_outer + 1
1005          t2 = b2 + num_outer + 1
1006          faces.append([b1, b2, t2])
1007          faces.append([b1, t2, t1])
1008
```

```
1009        # Close side wall loop
1010        b1 = num_outer
1011        b2 = 1
1012        t1 = b1 + num_outer + 1
1013        t2 = b2 + num_outer + 1
1014        faces.append([b1, b2, t2])
1015        faces.append([b1, t2, t1])
1016
1017        faces = np.array(faces)
1018
1019        # === Create STL Mesh ===
1020        cam_mesh = mesh.Mesh(np.zeros(faces.shape[0],
    dtype=mesh.Mesh.dtype))
1021        for i, f in enumerate(faces):
1022            for j in range(3):
1023                cam_mesh.vectors[i][j] = vertices[int(f[j]), :] # type:
    ignore
1024
1025        # === Export STL ===
1026        cam_mesh.save(filename) # type: ignore
1027        print("STL file saved as '{filename}'")
1028
1029    def STPGen(angle, Y, Thickness, filename):
1030        import cadquery as cq
1031        import numpy as np
1032
1033        ### Convert polar to cartesian ###
1034        x1, y1 = Y[0] * np.cos(angle[0]), Y[0] * np.sin(angle[0])
1035        x2, y2 = Y[1] * np.cos(angle[1]), Y[1] * np.sin(angle[1])
1036        x3, y3 = Y[2] * np.cos(angle[2]), Y[2] * np.sin(angle[2])
1037        x4, y4 = Y[3] * np.cos(angle[3]), Y[3] * np.sin(angle[3])
1038
1039        profile_x = np.concatenate((x1, x2, x3, x4))
1040        profile_y = np.concatenate((y1, y2, y3, y4))
1041
1042        ### Create list of (x, y) points as native Python floats ##3
1043        points_2d = [(float(x), float(y)) for x, y in zip(profile_x,
    profile_y)]
1044
1045        ### Remove duplicates or tiny distances (optional but helpful)
    ###
1046        cleaned_points = []
```

```
1047        for pt in points_2d:
1048            if len(cleaned_points) == 0 or np.linalg.norm(np.array(pt) -
    np.array(cleaned_points[-1])) > 1e-6:
1049                cleaned_points.append(pt)
1050
1051        ### Build and extrude using CadQuery ###
1052        try:
1053            wire = cq.Workplane("XY").polyline(cleaned_points).close()
1054            solid = wire.extrude(Thickness)
1055            cq.exporters.export(solid, filename)
1056            print(f"STEP file saved as '{filename}'")
1057        except Exception as e:
1058            print("Failed to generate STEP file:", e)
1059
1060    def CAMPlot(ui):
1061        from matplotlib.backends.backend_qtagg import FigureCanvasQTAgg
    as FigureCanvas
1062        import matplotlib.pyplot as plt
1063        import numpy as np
1064        from PyQt6 import QtWidgets
1065        from .CAM_Func import Harmonic, Linear, Parabolic, Cycloidal
1066        from MetadataManager.MetaMan import ScanInputs, SaveData
1067
1068        ### Get input from UI ###
1069        TR = float(ui.TravelRadius.text())
1070        D1 = float(ui.RiseDegrees.text())
1071        D2 = float(ui.HighDwell.text())
1072        D3 = float(ui.FallDegrees.text())
1073        D4 = float(ui.LowDwell.text())
1074        Df = [0, D1, D2, D3, D4]
1075        Tri = int(float(ui.Triangles.text()))
1076        BR = float(ui.BaseRadius.text())
1077        Thickness = float(ui.Thickness.text())
1078        RisePro = ui.RiseProfile.currentText()
1079        FallPro = ui.FallProfile.currentText()
1080
1081        g.Metadata=ScanInputs(ui)
1082
1083        ### Create plot ###
1084        fig, ax = plt.subplots(figsize=(4, 4), subplot_kw={'projection':
    'polar'})
1085        plt.rcParams['font.size'] = '0.5'
```

```
1086
1087        anglefinal = []
1088        Yfinal = []
1089
1090        ### Plot Rise ###
1091        if RisePro == "Harmonic":
1092            Y_rise, angle_rise = Harmonic(TR, Df, BR, Tri)
1093        elif RisePro == "Linear":
1094            Y_rise, angle_rise = Linear(TR, Df, BR, Tri)
1095        elif RisePro == "Parabolic":
1096            Y_rise, angle_rise = Parabolic(TR, Df, BR, Tri)
1097        elif RisePro == "Cycloidal":
1098            Y_rise, angle_rise = Cycloidal(TR, Df, BR, Tri)
1099
1100        ax.plot(angle_rise[0], Y_rise[0], label="Rise")
1101        ax.plot(angle_rise[1], Y_rise[1], label="Dwell After Rise")
1102        anglefinal.extend([angle_rise[0], angle_rise[1]])
1103        Yfinal.extend([Y_rise[0], Y_rise[1]])
1104
1105        ### Plot Fall ###
1106        if FallPro == "Harmonic":
1107            Y_fall, angle_fall = Harmonic(TR, Df, BR, Tri)
1108        elif FallPro == "Linear":
1109            Y_fall, angle_fall = Linear(TR, Df, BR, Tri)
1110        elif FallPro == "Parabolic":
1111            Y_fall, angle_fall = Parabolic(TR, Df, BR, Tri)
1112        elif FallPro == "Cycloidal":
1113            Y_fall, angle_fall = Cycloidal(TR, Df, BR, Tri)
1114
1115        ax.plot(angle_fall[2], Y_fall[2], label="Fall")
1116        ax.plot(angle_fall[3], Y_fall[3], label="Dwell After Fall")
1117        anglefinal.extend([angle_fall[2], angle_fall[3]])
1118        Yfinal.extend([Y_fall[2], Y_fall[3]])
1119
1120        #### Font styling ###
1121        ax.tick_params(labelsize=6)
1122        ax.legend(loc='best', prop={'size': 5})
1123
1124        ### Create and embed canvas ###
1125        canvas = FigureCanvas(fig)
1126        canvas.draw()
1127
```

```
1128        # Set up layout once if not done already
1129        if not hasattr(ui, 'plot_layout'):
1130            ui.plot_layout = QtWidgets.QVBoxLayout(ui.CAMPlot)
1131            ui.CAMPlot.setLayout(ui.plot_layout)
1132
1133        # Remove any existing plot widgets
1134        while ui.plot_layout.count():
1135            old_widget = ui.plot_layout.takeAt(0).widget()  # type:
    ignore
1136            if old_widget:
1137                old_widget.setParent(None)
1138
1139        ui.plot_layout.addWidget(canvas)
1140
1141        ### Return values for STL ###
1142        return anglefinal, Yfinal, Thickness
1143
1144    from PyQt6.QtWidgets import QFileDialog
1145
1146    def STLFileName(parent=None):
1147        filename, _ = QFileDialog.getSaveFileName(
1148            parent,
1149            "Save File",
1150            "",
1151            "STL Files (*.stl);;All Files (*)"
1152        )
1153        return filename if filename else None
1154
1155    def STPFileName(parent=None):
1156        from PyQt6.QtWidgets import QFileDialog
1157        import os
1158
1159        filename, _ = QFileDialog.getSaveFileName(
1160            parent,
1161            "Save STEP File",
1162            "",
1163            "STEP File (*.step);;All Files (*)"
1164        )
1165
1166        if filename:
1167            # Ensure .stp extension is present
1168            if not filename.lower().endswith(".step"):
```

```
            filename += ".step"
        return filename

    return None

def CADFileName(ui, parent=None):
    from PyQt6.QtWidgets import QFileDialog
    from pathlib import Path
    from MetadataManager.MetaMan import ScanInputs, SaveData

    CADFolder = Path(str(g.ProjectDirectory)) / "CAD"

    filename, _ = QFileDialog.getSaveFileName(
        parent,
        "Save CAM File",
        str(CADFolder),
        "STEP File (*.step);;STL File (*.stl);;All Files (*)"
    )
    print(filename)

    if filename:
        g.Metadata=ScanInputs(ui)
        outputs = {"CADFile": filename}

        SaveData(ui, ToolName="CAM Designer", outputs=outputs)

    #if filename:
        # Ensure .stp or .stl extension is present
        #if not filename.lower().endswith(".step"):
            #filename += ".step"
        #return filename

    return filename
```

A.8 GitHub – EngineeringTool-Public

The entire engineering tool developed in this thesis is in a GitHub repository as a public open-source project. It serves as a place for engineers and developers to download the tool to use and edit and it also serves as a place for new ideas and versions of the tool to be uploaded to. The goal of this repository is to provide open access to the modular engineering tool so that other engineers, students, teachers and researchers may use, modify or expand the tool.

The project is distributed under the MIT license, which permits free use, copying, modification, and distribution of the software, provided that the license and copyright notice are retained in all copies or substantial portions of the software. This ensures both openness and clarity about permitted use.

Public repository link:

https://github.com/EliSierra/EngineeringTool-Public