Cutting-Edge Solar Array Attachment

Design & Engineering of RODEO

Group D03

Angélique Orban	5779545	Veronica Fossa	5779960
Aitor Bilbao	5792347	Giulio Ruiz	5698073
Mete Ozozgur	5694906	Lucas Huirne	5680778
Jun Qi	5680263	Miguel Murcia	5696380

Delft University of Technology

This page is intentionally left blank.

Cutting-Edge Solar Array Attachment

Design & Engineering of RODEO

by

Group D03

Angélique Orban	5779545	Veronica Fossa	5779960
Aitor Bilbao	5792347	Giulio Ruiz	5698073
Mete Ozozgur	5694906	Lucas Huirne	5680778
Jun Qi	5680263	Miguel Murcia	5696380

Delft, December 5, 2023
Faculty of Aerospace Engineering
AE1222-I Design & Construction
S. Watson

TU Delft

Preface

For tribal man space was the uncontrollable mystery. For technological man it is time that occupies the same role.

Marshall McLuhan

This report is the result of the collaborative efforts of eight second-year Aerospace Engineering students at Delft University of Technology. The aim of this report is to document and showcase the scientific rationale and design approach for an Earth-observing satellite under the name of RODEO. RODEO is short for Remote Observer of Data for an Earth Overview. This satellite will bridge a gap in the knowledge of mankind, in a world where precise modeling and prediction of natural phenomena remain elusive. The satellite, equipped with state-of-the-art instruments, will acquire data for weather models, climate models, and seismology.

RODEO will carry various instruments, that will perform the following functions:

- High-resolution imaging of the Earth's surface for various ends;
- Measuring of weather and climate variables, as well as volcanic activity with a multi-spectral scanner;
- Atmospheric sounding to measure weather and climate variables at multiple altitudes above land and water.
- Imaging of the earth surface with a radar, to observe the earth in high resolution in the night and during cloudy periods

Several constraints on the final design of RODEO have been set by the tender, such as mass, power, size, temperature, reliability, and cost constraints. The mission has to be operational in a Low Earth Orbit for a minimum of five years and needs to be launched in 2025.

The authors would like to thank their TA Calvin Grootenboer for his help during the design process.

Delft, December 5, 2023
Faculty of Aerospace Engineering | AE2111-I Systems Design (Q2)

Sincerely,
Miguel Murcia
Lucas Huirne
Veronica Fossa
Angelique Orban
Aitor Bilbao
Giulio Ruiz de Cardenas
Mete Özözgür
Jun Qi

Summary

This report analyses the attachment system of the solar panel for the RODEO spacecraft. It shows step by step the design of the lugs, the fasteners and flanges, together with the choice of materials. In the previous work package, the solar energy has been selected as the primary source of energy, with an area of 7.54 m^2 in order to meet the power requirements and a mass of 17.68 kg. From these values the operational loads together with the launch loads have been studied in chapter 2, caused from the acceleration of the rocket in order to achieve the required orbit. The calculated launch loads can be found in table 2.1.

In chapter 3, the lug configuration have been discussed between one or two lugs and the final decision have been found once the optimization results have been obtained. Moreover, the constrains of the design parameters and the loads that could cause the failure of the lug have been found and put into code using Python, in section 3.2.

In chapter 4, constrains have been added to the fastener for the backup plate and the bearing check have been performed, from identifying the center of mass, to deriving the moment arm, the moment and finally finding the resultant force. The maximum stress has been then compared with the yield strength of the selected material for the back-plate of the lug. Subsequently, the force acting on each fastener has been found, as well as the shear stress caused by it and then compared with the shear yield strength of the material. In section 4.4 the selection of fasteners has been done, observing that the exact dimensions of the fastener depend on the diameter of the hole and the length of the fastener depends on the thickness of the back-plate of the lug.

In chapter 6, the optimization process is explained, which depends on the material, its yield strength and density. Once the optimization is run for each material, a trade-off is established in section 7.1to select the most appropriate one. The final design, described in section 7.3. Two lugs, two flanges and four fasteners have been found to be the best configuration; the material selected for the lug is 2024-T4 and Titanium for fasteners. The total mass is 79.05 kg. More specific information has been summarized in **??**. It can be visualized in figure 6.1.

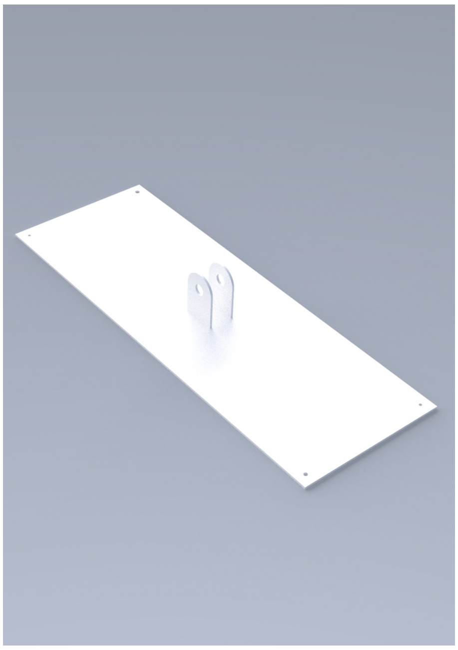


Figure 1: *Final attachment design*

Contents

List of Symbols

Symbols

W_{SA}	Weight solar array	N
R_a	Ratio of applied axial loads	-
R_{tr}	Ratio of applied transverse loads	-
$M.S.$	Margin of safety	W
K_t	Stress concentration factor	-
F_{tu}	Ultimate tensile force lug	N
A_t	Net tension area lug	mm^2
K_{bry}	shear-bearing efficiency factor	-
A_{br}	Bearing area	mm^2
F_{Ly}	Load along y-axis	N
x_{cg}	x-coordinate of center of gravity	mm
z_{cg}	z-coordinate of center of gravity	mm
A_i	Area of the i-th fastener hole	mm^2
x_i	x-coordinate of i-th fastener hole	mm
z_i	z-coordinate of i-th fastener hole	mm
n_f	Number of fasteners	-
F_{cgx}	Force acting at the center of gravity of the fasteners in the x-direction	N
F_{cgz}	Force acting at the center of gravity of the fasteners in the z-direction	N
r_i	distance of the i-th fastener hole to the center of gravity of the holes	mm
t_2	Thickness of back-plate	mm
t_3	thickness of spacecraft wall	mm
D_{fo}	Outer fastener diameter	mm
D_{fi}	Inner fastener diameter	mm
E	Youlgs Modulus	GPa

Greek Symbols

σ	Normal stress	MPa
σ_{yield}	Material yield stress	MPa
τ	Shear stress	MPa
π	Ratio of circumference and diameter	-
ϕ	Force ratio	-
δ	Compliance/material deflection	mm

Lug Abbreviations

h	Distance between flanges	mm
t_1	Thickness of flange	mm
w	Width of flange	mm
D_2	Diameter of fastener hole	mm
t_2	Thickness of the back-plate	mm
t_3	Thickness of the satellite skin	mm
D_1	Diameter of the flange hole	mm

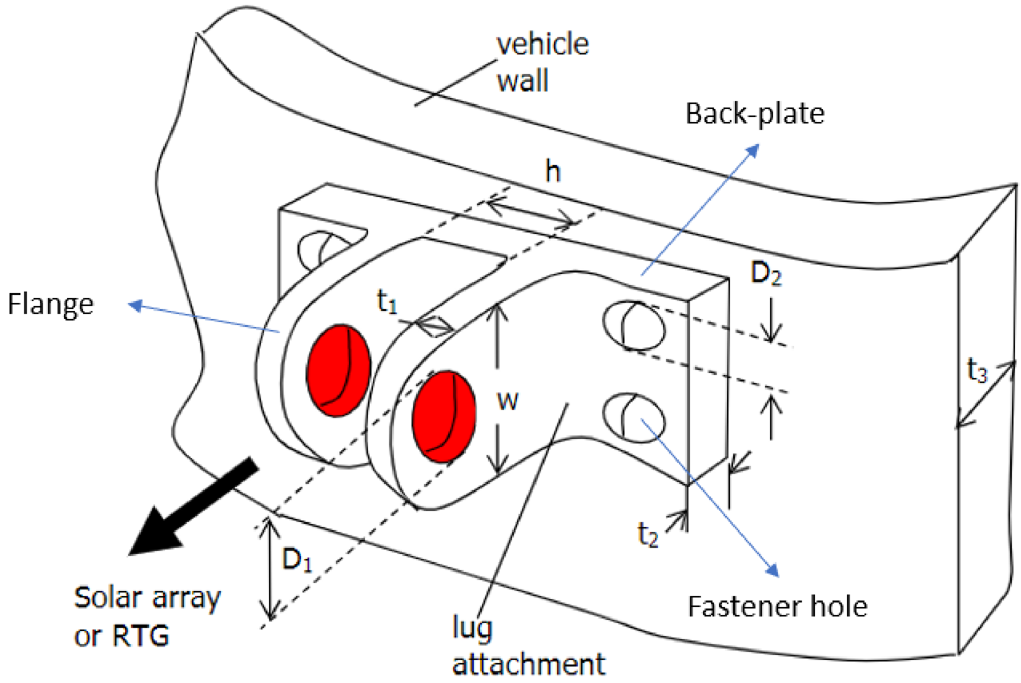


Figure 2: *Lug Abbreviation Modified [1]*

List of Tables

List of Figures

1 Introduction

The RODEO mission is an ambitious Earth Spectral Orbiter endeavor that revolves around a satellite located in a Sun-synchronous low Earth orbit, approximately 700 kilometers above Earth's surface. The main objective of the mission is to gather information about the Earth's surface for the purpose of mapping and studying the seismic activity of tectonic plates. Due to the orbit around Earth it is possible to use solar energy as the primary source of energy for the satellite. A fully operational and orientation of the solar array is of big importance in order to accomplish the mission. This includes a good attachment system of the solar array to the spacecraft, which is going to be designed in this report, including a fitting that connects the system to the vehicle and some fasteners.

The aim of this report is to design the attachment lug for the solar panel of the RODEO satellite. The detailed structural design of an attachment lug will initially be performed, by analysing the forces and stresses acting on it, and then the design will be iterated and a trade-off based on the outcomes is going to be necessary to identify the best configuration. Assumption will be done and explained throughout the report, and this will lead to a low-fidelity model.

The report will be presented in the following structure. In chapter 2 a detailed load analysis will be performed, including the operational loads and the launch loads. In chapter 3 different lug configurations will be investigated and successively, the flanges will be designed and analyzed in section 3.2. The fasteners are going to be discussed in chapter 4, with the final selection after checking the different forces acting on them, bearing and pull-through. The effect of thermal fluctuation will then be discussed in **??** and in chapter 6 the optimization process of all the previous analysis performed on Python will explained. The final material selection for the lugs and fasteners will be performed in section 7.1. Finally, insection 7.3 the final design of the attachment system will be presented and in chapter 8 visualisations and post processing will be discussed. The entire code in Python can be found in appendix A, together with the drawings of the final design, in appendix B.

During this project over almost 1750 lines of code were written. This code can be found in the repository "HuiLucas/LugDesign" on GitHub [1]. To run the code it is possible to clone the code from there. It is also possible to pull a whole container of code from Docker Hub, in the repository "lhuirne/lug_design3" [2]. The code was also uploaded to a Microsoft Azure Container Instance and to AWS EC2 using free computing/storage for students, but those are not publicly visible.

[1]See: https://github.com/HuiLucas/LugDesign.git
[2]see: https://hub.docker.com/r/lhuirne/lug_design3

2 Load Analysis

The design of a lug requires an exploration of the several loads and moments that the component will experience during its lifetime. This chapter contains a detailed load analysis describing the launch loads and the operational loads.

2.1. Launch Loads

Launch loads occur due to the acceleration of the rocket as it approaches the required orbit. The lug(s) must support the weight of the solar panel in the stowed configuration as it reaches these maximum loads. The stowed configuration assumes that panels are folded in the longitudinal direction of the spacecraft for it to fit inside the payload fairing.

The launch loads that will act on the spacecraft have been given by Space X when describing the Falcon 9 launch rocket. The axial load equals 6g, which is six times the gravitational constant, it is assumed that this constant takes into account 1g of the gravitational pull on the solar array. This acceleration has to be multiplied by the mass of the solar panel to find the force acting on the hinge, which will give a reaction force. The mass of the solar panel is 17.68 kg, thus the total axial force is 1040.64 N using equation 2.1. Knowing that the maximum lateral acceleration of the spacecraft during launch is 2g, the maximum lateral load required to be supported by the lug(s) can be calculated to be 346.88 N using equation 2.2. It can be assumed that this acceleration applies to both x- and y-direction, as the launch vehicle is free to rotate. These lateral accelerations during launch will also cause a moment about the y- and x-axes respectively. To calculate these moments, the distance between the centre of mass of the panel and the reference point of the lug, shown in figure 2.1, where all forces and moments are applied, must be multiplied by the force experienced laterally. Given that the panel has a length of 3.77 meters, and assuming an even distribution of mass along the panel, such a distance equals 1.85 meters, which leads to a maximal moment of 653 Nm about both the y- and x-axes. The moment around the z-axis would only be caused by a spinning of the rocket along its axis, which is negligible, as the Falcon 9 has a very slow spin rate. This moment direction will be non-negligible later on in the analysis, as the forces being introduced at the shaft location, will cause a moment in the z-direction for the back plate and fasteners, which will be discussed further on in the report.

$$F_A = m_{SA} a_A \tag{2.1}$$

$$F_L = m_{SA} a_L \tag{2.2}$$

Considering a safety factor of 1.25 [2], the loads summarised in table 2.1 can be obtained. The decomposition of forces and moments generated on the lug(s) due to the launch loads and their reaction can be observed in figure 2.1. The decomposition of forces and moments on the flange(s) and fasteners will be explained throughout the report.
Note that the forces and moments in blue are the reaction loads if the lug is set to be an isolated system.

2.2. Operational Loads

RODEO will also suffer external disturbances during its lifetime and will require internal torques to ensure its correct operation. Such internal torques will be generated by an electrical motor that will rotate the solar panel during operation. However, the structure containing the motor will not be load-carrying, instead, the lug will carry all loads and moments. As calculated in WP3, the internal torque required

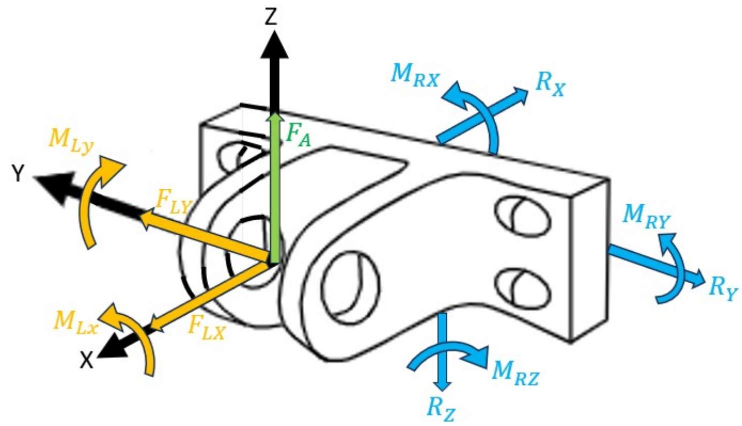


Figure 2.1: *Launch loads experienced by the lug*

Table 2.1: *Summary of launch loads acting on the lug*

Force	Value (Magnitude) [N]	Value (SF=1.25) [N]
F_A	1040.7	1300.8
F_{Lx}	346.9	433.6
F_{Ly}	346.9	433.6
Moment	Value (Magnitude) [Nm]	Value (SF=1.25) [Nm]
M_{Lx}	653.9	817.4
M_{Ly}	653.9	817.4

for changing the orientation of the spacecraft for *Night Mode Operation*, which is essentially rotating the spacecraft during the eclipse for imaging, equals 3.03 [Nm]. Also, during the eclipse, the spacecraft will suffer a sinusoidal torque of 9.38 [Nm] that will enable accurate imaging of the UMBRA-SAR. The first torque addressed will be applied twice in each orbit. It will be applied during 45 [s], which leads to a slow rotation. For the design of the lug, this torque can be considered negligible.

During *Night Mode Operation*, the sinusoidal torque will lead to fatigue. However, the value of such a torque leads to a smaller loading than the moments calculated in section 2.1, and will therefore not be considered during the design of the lug. It will be addressed in section 7.1 when selecting the final material.

The goal of determining the load is to obtain the limiting case. It can be observed that the resultant operational loads, derived from these torques, are negligible, or insignificant when compared to the launch loads. Thus, it can be assumed that the moments caused by the operation of the spacecraft won't limit the design, and only the launch loads should be taken into account.

3

Lug Configuration and Flange Design

In this chapter, the different lug configurations will be investigated and a choice will be made between the single or double lug setup based on the loading and solar panel attachment. The second part of this section consists of a description of the process behind the design of the flanges.

3.1. Selection of lug configuration

3.1.1. Comparison between single and double configuration

Two configurations that are investigated are single and double lug configurations. Such a design choice would influence the loading distribution described in figure 2.1, the forces would be halved making the overall dimensions smaller in a two-lugs configuration and the moment acting along the x-axis would be translated into two forces acting on each lug, and would be therefore dependent on the distance between the two lugs, limited to two meters, the width of the spacecraft. The choice of the configuration influences a lot of parameters besides the loading such as the attachment of the solar panel. In a single-lug configuration, the solar panel would have to be attached on the side of the lug resulting in a large moment along the x-axis whereas in a two-lug configuration the solar panel can be attached in between the two lugs, potentially reducing the amount of torque on each lug. One last parameter that the lug configuration would influence is the weight of each lug. Even though not limiting the design, the mass of the lugs should be taken into consideration since selecting a more lightweight design with comparable performances is preferred.
The choice of the final configuration has been taken once the optimization results were obtained in subsection 3.2.2.

3.1.2. Locking mechanism for the shaft

Regardless of the configuration chosen, the solar panel shall be attached to the spacecraft by the means of one/two shaft(s) going though the lug(s). The locking mechanism opted for that would stabilize or rotate the solar panel when deploying consists of one or two motors depending on the configuration. In the single-lug configuration, a single shaft connected to one motor would lock the solar panel in place. Whereas in a double lug configuration, given the distance between the lugs came out to be 1.4 meters as a result of the optimization in subsection 3.2.2, the solar panel would be attached with two shafts, each going through one lug. Therefore, each one of the two shafts would be connected to their own motor.

3.2. Design of the flanges

3.2.1. Constraints on the design parameters

The design of the lug is based on three considerations:

- The lug shall withstand the limiting loads and moments applied to it, and described in figure 2.1;
- The mass of the lug shall be minimised.

The design of the flange will consider a list of parameters containing the width and thickness of the flange; the diameter of the hole in the flange; and the distance between the two flanges of each of the lugs. These will be determined based on the loading case described under chapter 2.
The calculations made to design the lugs take the limiting stress to be the yield stress with a safety factor of 1.1 [2], in order to prevent catastrophic events due to yielding. Efficiency factors for yield will indeed be used rather than for ultimate tension or stresses. No design constraints were imposed on the maximum deflection of the flange or other elastic behaviour.

According to the method described in section D1 of the book ”the Analysis and Design of Flight Vehicle Structures” by Bruhn [3] the lug can fail due to different type of loading. The loading types investigated and analysed for the sizing of the flange are: failure due to tension across the net section, failure due to transverse loads and failure due to shear. To check that whether the lugs will break under the loads applied equation 3.1.

$$R_a^{1.6} + R_{tr}^{1.6} = 1 \tag{3.1}$$

In equation 3.1 the term R_a stands for the axial component of the applied ultimate load divided by the smaller of the two values obtained by calculating the tension across the net section and the shear-out bearing strength [3]. $R_t r$ is the transverse component of the applied ultimate load divided by the yield strength [3].

However not considering a margin of safety would be equivalent to design for the lug to yield during launch, therefore equation 3.1 must be formulated to account for that margin of safety.

$$M.S. = \frac{1}{(R_a^{1.3} + R_{tr}^{1.6})^{0.625}} - 1 \tag{3.2}$$

The margin of safety given by equation 3.2 can be estimated to be 0.375 and stems from the product of the safety factor of 1.1 on the material properties by the safety factor of 1.25 on the loads and moments.

Then, the allowable axial and transverse forces can be estimated by equation 3.3 and equation 3.4 from the same section D1 mentioned earlier [3].

$$P_u = K_t \cdot F_{tu} \cdot A_t \tag{3.3}$$

The axial load P_u is therefore dependent on the ultimate tensile force F_{tu} or in this case the force F_A as can be seen in figure 2.1, the net tension area A_t, which is dependent on the dimensions designed for, and the stress concentration factor K_t. The latter can be determined depending on the material in figure 3.1 with the curves corresponding to the material given in figure 3.2.

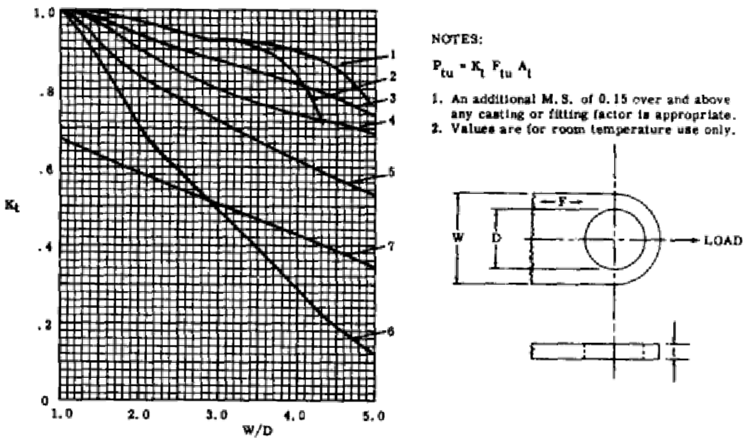


Figure 3.1: *Stress Concentration factor for axially loaded lug [3]*

For the transverse loads, they consists of the forces along the y- and z-axis in figure 2.1 and they will cause the lug to shear if not accounted for. The transverse loads can be computed thanks to equation 3.4. They are therefore dependent on the bearing area, applied forces and shear-bearing yield efficiency factor K_{bry}, dependent on the thickness-to-diameter ratio (diameter or the hole) of the flange as can be seen in figure 3.3.

$$P_{br} = K_{bry} \cdot F_{tu} \cdot A_{br} \tag{3.4}$$

In order to make use of Python to iterate through the different possible materials and dimensions, the graphs in figure 3.1 and figure 3.3 were digitized to extract a polynomial relation thanks to the online

Table D1. 3

Curve Nomenclature for Axial Loading for Fig. D1. 12

L, LT and ST Indicate Grain in Direction F in Sketch
L - Longitudinal
LT - Long Transverse
ST - Short Transverse (Normal)

MATERIALS	
Curve 1 -	2014-T6 and 7075-T6 Die Forging (L) 4130 and 8630 Steel 2014-T6 and 7075-T6 Plate ≤ 0. 5 (L, LT) 7075-T6 Bar and Extrusion (L) 2014-T6 Hand Forged Billet ≤ 144 in.² (L)
Curve 2 -	2014-T6 and 7075-T6 Plate > 0. 5 in. ≤ 1. 0 in. (L, LT) 7075-T6 Extrusion (LT, ST) 2014-T6 Hand Forged Billet > 144 in.² (L) 2014-T6 Hand Forged Billet ≤ 36 in.² (LT) 2014-T6 and 7075-T6 Die Forgings (LT)
Curve 3 -	2024-T4, 2024-T2 Extrusion (L, LT, ST)
Curve 4 -	2014-T6 and 7075-T6 Plate > 1 in. (L, LT) 2024-T4 Bar (L, LT) 2024-T3, 2024-T4 Plate (L, LT)
Curve 5 -	2014-T6 Hand Forged Billet > 36 in.² (LT)
Curve 6 -	Aluminum Alloy Plate, Bar, Hand Forged Billet and Die Forging (ST). NOTE: For Die Forgings ST Direction Exists Only at Parting Plane. 7075-T6 Bar (T)
Curve 7 -	AZ91C-T6 Mag. Alloy Sand Casting 356-T6 Aluminum Alloy Casting

Figure 3.2: *Materials corresponding to the graphs in figure 3.1 [3]*

software automeris.io [1].

Beside the launch loads, moments are also acting on the lug in the x- and y-directions. As detailed in section 3.1, within a single lug configuration, the moment along the x-axis can be converted into two forces constituting a couple moment that acts on the flanges. Additionally, this force would be combined with the force along the z-axis. When opting for a two lug configuration, the moment along the x-axis creates two equal but opposite in direction forces (couple moment) on the two lugs and, considering the inter-flange distance negligible, the additional force acting on one flange due to this moment would be half the one acting on the lug.

The equation 3.2 can now be adapted given the loading and returns equation 3.5, with d being the distance between the lugs.

$$M.S. = ((\frac{F_{Lx}}{K_t \cdot \sigma \cdot 1.1 \cdot A_t})^{1.6} + (\frac{F_A + \frac{M_y}{d \cdot N_{Flanges}}}{K_{ty} \cdot A_{br} \cdot \sigma \cdot 1.1})^{1.6})^{-0.625} - 1 \tag{3.5}$$

Lastly, because of the offset of the load acting along the y-axis, F_{Ly}, an extra moment is applied and should also be taken into consideration. Such a moment causes pure bending of the lug and restrict the possible heights of the flange, denoted h. By solving equation 3.6 for the height of the flange, taking the moment of inertia along the z-axis, as it is the axis with smallest moment of inertia, it ensures that the lug will withstand the load.

$$\sigma_{yield} \cdot 1.1 = \frac{F_{Ly} \cdot height_{flange}}{I} \tag{3.6}$$

3.2.2. Optimization of the design of the flange

Now that the design constraints regarding the loading have been established, the design process can start. All the lines of code regarding the flange design optimization, digitization of the graphs, etc. can be found in the appendices under Listing A.4.

The function used to generate an optimized lug design is part of the module scipy.optimize, and is called minimize. This function provides a common interface to unconstrained and constrained minimization algorithms for multi-variable functions. In this case, the objective function that is minimized for each material is the mass of a specific part of the flange while constraining the design variables to optimize: the outer radius of the flange e, the thickness t, the diameter of the inner hole to fit in the shaft D and

[1] https://apps.automeris.io/wpd/

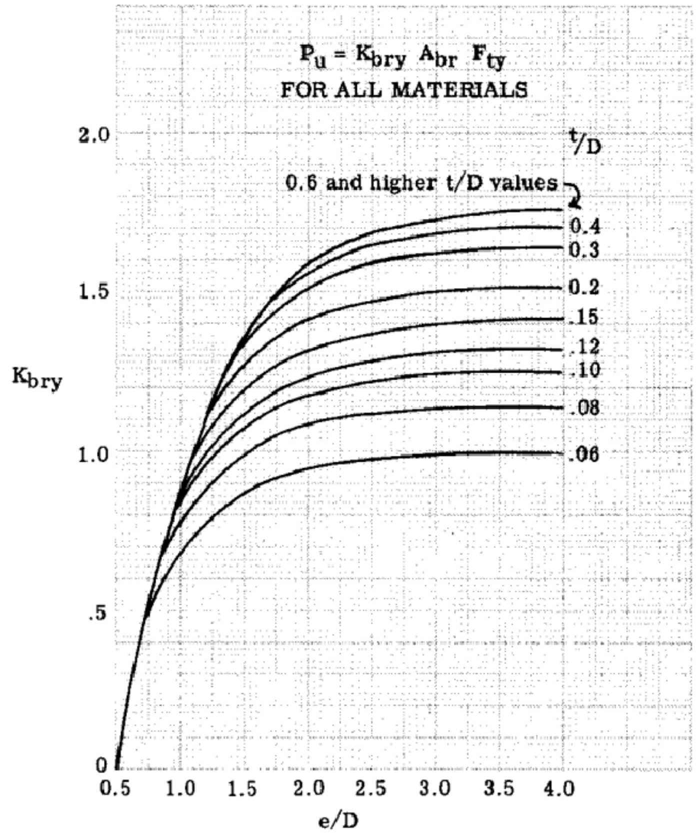


Figure 3.3: *Values of shear-bearing factors for lugs [3]*

the inter-flange distance h.

As mentioned above, the function aims to minimized the the mass of a certain part of the flange, that is the hollow cylinder delimited by the hole and the outer-diameter of the flange. To do so, the algorithm chosen is SLSQP, it minimize a function of one or more variables using Sequential Least Squares Programming. That method was opted for as it allows to constraint the variables given a set of inequalities or equalities. For example, in order to not fail under the launch loads, the design variables should satisfy equation 3.5. Another relevant constraint that was added is that the volume, the design variables and the mass shall be positive. Sometimes to further limit the outcomes to plausible design, constraints such as the inner radius shall be at least 10 times bigger than the thickness are added. All of those constraints are listed in a dictionary, and passed as an argument in the minimized function.

The subtlety lies in the fact that the optimization takes as input initial guesses and, given their accuracy, the function might converge or not. However, from experience, the optimization is very sensible, therefore intricated for-loops are used to iterate through every possible initial e, t, D and h configuration and append every plausible outcomes in an array from which the best configuration was selected. But a major problem arises from this approach: the computation time. At first to overcome this problem, tolerance was introduced. Indeed by adding a tolerance, the optimization converges more easily and less initial guesses have to be provided, implying therefore to find a fine balance between a tolerance low enough to give accurate results and an acceptable amount of iterations. But since that was not enough to speeding up the computation time, Numba is used. It consists of a Just-In-Time (JIT) compiler for Python code which is meant to have it automatically optimized for execution speed.

As investigated in section 3.1, several lug configurations can be selected: a single or double lug configuration. The optimization was made with consideration for both scenarios, allowing for the evaluation of which configuration to choose. When comparing the 3D models automatically generated with numpy.stl, it appeared that when using a single lug, the thickness of the flanges is quite large and the inter-flange distance quite small whereas when opting for a two lugs at a distance from 50cm to 1m, the thickness of the flanges reduces drastically. Therefore, the two lugs configuration was selected not only for the reason mentioned above but also since it would allow the solar panel to be placed in between the lugs, reducing the moment acting along the x-axis and reducing the magnitude of the loads on the lugs by two.

As mentioned earlier, the optimization is dependent on the material, its yield strength and density. The list of materials considered was taken from the different materials referenced for the graphs under figure 3.2: 2014-T6(DF-L), 2014-T6(DF-LT), 2014-T6(P), 7075-T6(P), 7075-T6(DF-L), 7075-T6(DF-LT), 4130 Steel, 8630 Steel, 2024-T4, 356-T6 Aluminium and 2024-T3. Subsequently, once the optimization is run for each material, a trade-off is established in section 7.2 to select the most appropriate one.

4

Fasteners Configuration

This chapter discusses the fastener configuration with a detail focus on bearing check and pull through check.

4.1. Select fastener pattern for the backup plate

For a given configuration of fastener positions and hole diameters, two constraints are given by the reader. Namely, the distance from the hole position should be at least 1.5 times the diameter of the hole itself. This constraint is chosen to ensure that the stress can distribute itself evenly around the hole preventing stress concentrations that could lead to premature failure. For the same reason, a lower limit is placed on the center-to-center distance between the holes. The magnitude of this distance depends on the material selection for the back plate. In case it was a composite the magnitude of a distance of 4 times the diameter of the hole was chosen.On the other hand it were a metal a distance of 3 times the diameter of the holes was chosen.

To implement these constraints in Python a function was created that takes a list of coordinates of the hole positions and a list of hole diameters to return True if the design complies with requirements. If the design does not comply the function returns False and specifies the index of which hole is causing the issue. This will be useful for the final optimization as it allows us to identify exactly what is wrong in the design such that it can be changed. All the code is present in Listing A.8.

4.2. Bearing check for fasteners in back-up plate

To inspect the bearings of the fasteners on the back-plate, start by identifying the center of mass for the fasteners within the back-plate. Utilize the coordinate system outlined in figure 4.1, where the x and z prime axes align with the forces projected onto the plate. This alignment proves particularly beneficial for subsequent analysis of in-plane forces. The centroid's coordinates can be computed using equation 4.1 and equation 4.2, where A_i represents the area of the i-th fastener hole. It is noteworthy that the chosen coordinate system ensures that both x_i and z_i are consistently positive.

$$x_{cg} = \frac{\sum A_i x_i}{\sum A_i} \tag{4.1}$$

$$z_{cg} = \frac{\sum A_i z_i}{\sum A_i} \tag{4.2}$$

Proceeding further, the in-plane forces can be calculated employing equation 4.3, equation 4.4, and equation 4.5. In these equations, n_f denotes the total count of fasteners, A_i signifies the cross-sectional area of the i-th fastener, and r_i represents the radial distance from the center of the i-th fastener to the previously determined center of gravity of the fasteners.

$$F_{in-plane-x} = \frac{F_{cgx}}{n_f} \tag{4.3}$$

$$F_{in-plane-z} = \frac{F_{cgz}}{n_f} \tag{4.4}$$

$$F_{in-plane-M_y} = \frac{M_y A_i r_i}{\sum A_i r_i^2} \tag{4.5}$$

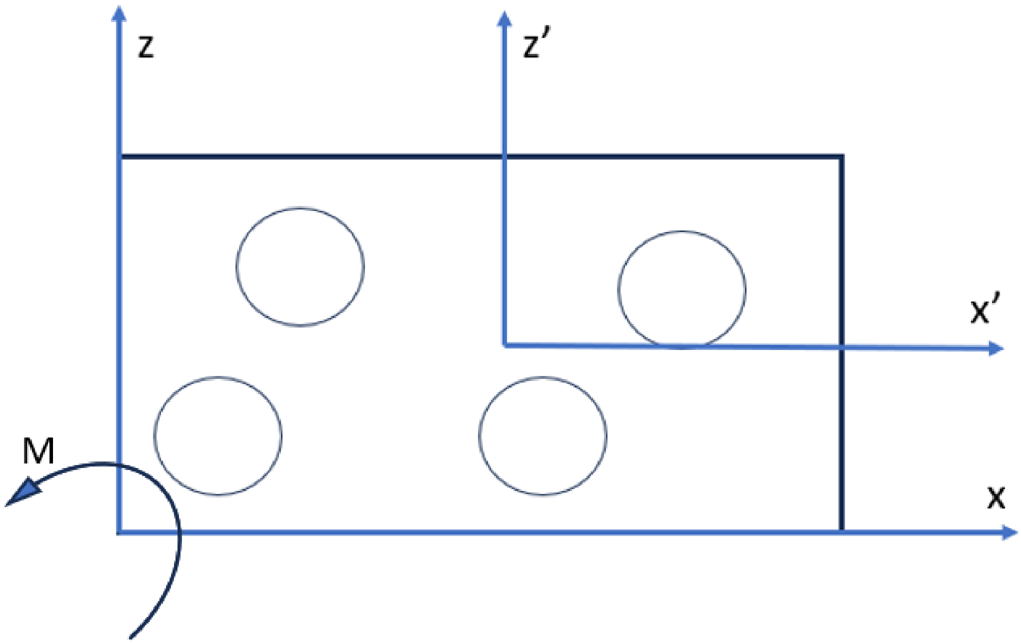


Figure 4.1: *Coordinate system to determine the centroid*

Following that, it is crucial to shift the loads onto the back-plate of the lug. The forces involved are F_L and $F_A - W_{SA}$, as indicated in figure 2.1 during the launch phase, acknowledged as the more demanding aspect of lug design. However, these forces will give rise to a moment on the back-plate, contingent upon the predetermined location of the centroid of the holes. In total, there are nine situations concerning the x'-z' coordinate system, as illustrated in figure 4.1.

1. The centroid lies on the origin, then the resultant moment is zero.
2. The centroid lies on the positive x' axis, then there is a negative resultant moment caused by F_z.
3. The centroid lies on the negative x' axis, then there is a positive resultant moment caused by F_z.
4. The centroid lies on the negative z' axis, then there is a positive resultant moment caused by F_x.
5. The centroid lies on the positive z' axis, then there is a negative resultant moment caused by F_x.
6. The centroid lies on the first quadrant, then there is a negative resultant moment caused by F_x and F_z.
7. The centroid lies on the second quadrant, then there is a negative resultant moment caused by F_x and a positive resultant moment caused by F_z.
8. The centroid lies on the third quadrant, then there is a positive resultant moment caused by F_x and F_z.
9. The centroid lies on the fourth quadrant, then there is a positive resultant moment caused by F_x and a negative resultant moment caused by F_z.

The subsequent step involves determining the moment arm on which the resultant moments are generated. This entails calculating the perpendicular distance from the centroid of the holes to the forces acting on the origin of the x'-z' coordinate system. Consequently, M_y is derived. Subsequently, it is necessary to determine the distance from each hole to the centroid of the holes, denoted as r_i. Utilizing this information, the in-plane force caused by M_y can be readily computed. The resultant force is then determined by considering only their magnitudes, as they do not act along the same axis, using equation 4.6.

$$F_{resultant} = \sqrt{F_{in-plane-x}^2 + F_{in-plane-z}^2 + F_{in-plane-M_y}^2} \tag{4.6}$$

It's important to note that the thermal force can be incorporated into equation 4.6. This addition will lead to a larger resultant force, subsequently causing an increase in stress. To assess the potential yield of the material, a comparison is made between the stress and the yield stress of the selected material for the plate. The stress is determined using equation 4.7.

$$\sigma = \frac{F_{resultant}}{D \cdot t_2} \tag{4.7}$$

Here, D represents the diameter of the holes, and t_2 is the thickness of the back-plate of the lug. It's crucial to recognize that if the holes have varying diameters, the stress on the holes will differ. In such instances, the maximum stress should be compared with the material yield stress, and this maximum stress occurs at the holes with the smallest diameter.

To translate the aforementioned theories into Python code, several functions have been defined. These functions calculate the centroid, the corresponding in-plane forces, and the stress induced by the resultant forces. In the final step, the maximum stress is compared with the yield strength of the selected material for the back-plate of the lug. The main loop returns one of two messages: either "The bearing stress check passed" or "The bearing stress check failed, increase the thickness of the back-plate." It's important to note that no recommendation is provided to alter the configuration of the fastener hole. This is because the size and configuration of the fastener holes significantly impact the final calculation, and adjusting these parameters for fine-tuning is not advisable.

4.3. Pull-through check

Once the force acting on each fastener has been determined, it is necessary to check that this force will not cause failure in the fasteners.

The axial force on each fastener, defined as F_{yi}, produces a normal stress on the area in between the outer diameter of the fastener and the hole, visible in figure 4.2, and a shear stress. The shear stress acts on the vertical plane of the plate at the outer diameter of the fastener, causing pull-through. It is necessary to compare the shear stress with the yield shear stress of the material of the lug and check that the yield shear stress is bigger.

In order to determine the stress on each fastener, the axial loading of each fastener must be known. This can be calculated by considering two components. Firstly, the normal force which they must bear, which is defined as Fpi, and is simply the total force in the y-direction F_y divided by the number of fasteners n_f, as is seen in equation 4.9.

$$F_{pi} = \frac{F_y}{n_f} \tag{4.8}$$

There is another component that must be considered for the axial load on each fastener, which is the contribution from the moments. In this loading case, there will be a moment in the x-direction to be carried by the fasteners, which is the result of the M_x moment presented in chapter 2, added with the F_z multiplied by the height of the flange, as this will also create a moment in the x direction. In the z-direction, there will also be a moment, which is due only to the F_x force, multiplied by the flange height. These two moments will lead to each quadrant of the back plate being loaded differently. This means some of the fasteners will have much more force on them, and under certain loading cases, they might be loaded in compression instead of tension. The contribution to $F_y i$ from the moments can be calculated using equation 4.9, where A represents the area of each fastener hole and r represents the distance from the hole center to the location of the load application, in the relevant direction.

$$F_{p-M-i} = \frac{M_z \cdot A_i \cdot r_i}{\Sigma A_i \cdot r_i^2} \tag{4.9}$$

Summing the total of F_{pi} and F_{p-M-i} gives the F_{yi}, or the axial force on each fastener. In the code, it was observed that the contribution from the moments to the load on each fastener was much larger than the F_y contribution, therefore, the most effective change to make would be to increase the distance from the fastener to the location of load application.
This check will lead to modification of the position of the fasteners. If the shear stress produced by the vertical force is bigger than the shear yield stress, failure would occur and this leads to the need for a modification. On the other hand, if failure does not occur, it is possible to still reduce the distance of the fasteners, saving mass. This optimization of the fastener locations is performed in chapter 6.

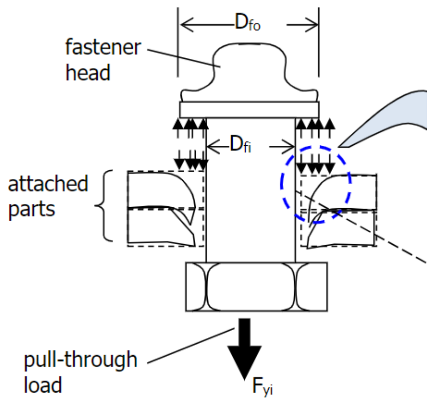


Figure 4.2: *Stresses produced by the fastener [1]*

The calculation needs to be performed separately for each fastener since the force acting on it is different, as well as the inner diameter. Moreover, the material has not been selected yet so it going to be computed as a variable in the code, but the yield stress and the shear yield stress are going to be known parameters for each material. The shear yield stress of the materials is known and it is going to be compared with the shear stress produced by vertical force Fy.

The value of the normal stress σ_z can be calculated using equation 4.10, and it depends on the diameter of the fastener, called D_{fi}, previously calculated, and the outer diameter of the faster, D_{fo}, which has been assumed to be 1.8 times the inner diameter after analysing different bolts from [4]. This assumption was used just for the check function to simplify the calculation. In the final design, given the inner diameter already existing bolts with precise dimensions have been found and used.
The shear stress generated by the vertical force Fy can be found with equation 4.11 and it is the one to be compare with the shear yield stress of the material.

$$\sigma = \frac{Fy}{\pi * 1/4 * (D_{fo}^2 - D_{fi}^2)} \tag{4.10}$$

$$\tau = \frac{Fy}{\pi * D_{fo} * (t_2 + t_3))} \tag{4.11}$$

All of these equations have been put into a check function in Python, where the output is going to tell if it is necessary to increase the thickness or not. In the optimization code, explained in chapter 6, the thickness will be optimized using this function that checks if the lug can withstand the force without failure.

4.4. Selection of fasteners

The decision has been made to opt for an off-the-shelf fastener rather than designing a state-of-the-art fastener. This choice is based on two primary considerations. Firstly, off-the-shelf fasteners undergo rigorous testing and extensive applications, demonstrating reliability within certain limits. Secondly, the customization and design of components, such as fasteners and bearings, tend to significantly extend both the development time and budget of the mission. Furthermore, the outcomes of such customization efforts may not consistently align with the expended effort. In the selection of the fastener, three key criteria are considered:

1. **Material Distinction:**
   - Stakeholder Requirement: The material of the fastener must differ from the material of the back plate.
2. **Shank Length Requirement:**
   - Design Constraint: The length of the shank of the fastener should equal the sum of the thickness of the lug plate and the satellite skin, with an appropriate margin.
3. **Shank Diameter Requirement:**
   - Design Constraint: The diameter of the shank of the fastener is to be equal to the diameter of the respective holes, considering manufacturing margins.

A trade-off table is made as table 7.1. With the known material and section characteristics of the fastener, the force ratio could be determined as equation 4.12.

$$\phi = \frac{\delta_a}{\delta_a + \delta_b} \tag{4.12}$$

Where δ_a is the compliance of the attached parts and δ_b is the compliance of the fastener. To find δ_a and δ_b, use equation 4.13 and equation 4.14.

$$\delta_a = \frac{4t}{E_a \pi (D_{fo}^2 - D_{fi}^2)} \tag{4.13}$$

Where t is the thickness of the back-plate or the vehicle wall, E_a is the young's modulus of the back-plate or the vehicle wall, D_{fo} and D_{fi} are the outer and inner diameter of the fastener as defined in figure 4.2

$$\delta_b = \frac{1}{E_b} \sum \frac{L_i}{A_i} \tag{4.14}$$

Where E_b is the Young's modulus of the fastener material, A_i and L_i are the local cross-section of the segment and the corresponding length.

The equation 4.13 illustrates the computation of two force ratios, crucial for thermal stress assessment, concerning the back-plate of the lug and the satellite's skin. The smaller force ratio, identified as the more restrictive scenario, is denoted as δ_b. Determining δ_b necessitates pinpointing the fastener segment. figure 4.3 displays potential segments of a fastener. First, we identify substitution lengths ($L_{h,sub}$, $L_{eng,sub}$, and $L_{n,sub}$) for deformations in the head, fastener's engaged region, and nut's engaged region, respectively (refer to figure 4.3). Statistical methods, as shown in figure 4.4, guide the dimension determination of these parameters.

When selecting the fastener/joint configuration, a preference for a smaller substitution length is advisable to enhance the force ratio and reduce thermal stress. However, choosing a nut-tightened engaged shank is favoured over a threaded hole due to the impracticality of threading the thin (3 mm) satellite skin. The dimensions of $L_{1,2,3}$ are overlooked since fasteners are typically chosen off-the-shelf, and no evident constraints guide the selection of these dimensions. It is assumed that the shank has a constant diameter, covering the entire distance between the back-plate of the lug and the satellite skin.

In conclusion, the fastener's exact dimensions depend on the hole diameter (equivalent to the shank diameter) and the back-plate thickness. Notably, if the fastener length exceeds the combined thickness of the back-plate and skin (denoted as "t"), considerations arise regarding the threading location on the fastener. If threads are solely outside the shank, the shank length should be smaller than "t" to ensure secure tightening. Although an exact match is ideal, manufacturing variations are common. Alternatively, if the shank length surpasses "t," the entire bolt length must be threaded or at least until the nut contacts the satellite wall. This design choice relies on market availability, as precision machining may be impractical.

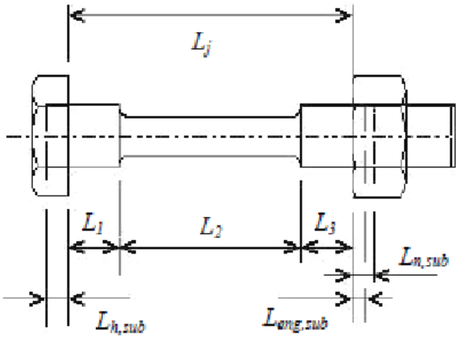


Figure 4.3: *Typical fastener with a shank[5]*

Figure 4.4: *Typical substitution length[5]*

Part of Fastener	Parameter	Fastener/Joint Configuration	Typical Substitution Length
Head	$L_{h,sub}$	Hexagon head	0.5 d
		Cylindrical head	0.4 d
Engaged shank	$L_{eng,sub}$	Nut-Tightened	0.4 d
		Threaded hole	0.33 d
Locking device (nut or insert)	$L_{n,sub}$	Any	0.4 d

To implement the fastener type in the Python project a new class was created to combine all of the variables that need to be selected for the fastener like the material, its Young's Modulus, and the type of bolt used. The creation of such a class is useful to keep the code organized and to facilitate the creation and readability of many functions that have to do with the fastener selection. For example, the force ratio calculations were performed often using this class. The functions follow mainly the equations equation 4.13 , equation 4.14 , equation 4.12. All the equations are to be applied iteratively on all of the holes present in the back plate. For equation 4.13 in particular, two sets of values for compliance can be calculated because both the back-plate and the vehicle will have to be considered. The code calculates both and then checks which set is limiting to find the smallest possible force ratios. All of the code is found in Listing A.9.

Table 4.1: *Fastener Final Choice*

Nr.	Diameter [mm]	Insert Length [mm]	Chosen fastener diameter [mm]	Chosen fastener length [mm]	Material	Manufacturer
1	1.3	3	1.2	8 (Margin + Nut)	Titanium Ti-6AI-4V	Extreme Bolt & Fastener
2	1.7	3	1.6	8	Titanium Ti-6AI-4V	Extreme Bolt & Fastener
3	2.2	3	2	8	Titanium Ti-6AI-4V	Extreme Bolt & Fastener
4	1.3	3	1.2	8	Titanium Ti-6AI-4V	Extreme Bolt & Fastener

The table 4.1 lists the final design of the fasteners and the corresponding selected market-available fasteners. Note that 8 mm is chosen as the length to allow for a margin (threaded length) and nut. The inserted length 3 mm on the other hand is the summation of the optimised thickness for the back-plate and the satellite skin, see chapter 6. For material, see table 7.1.

5

Thermal Stresses

In this chapter, the analysis of thermal stresses will be performed, by checking if the current design will withstand the thermal fluctuations, in section 5.1.

5.1. Thermal stress check

The RODEO satellite will experience a wide range of temperatures. The solar panel of RODEO will experience temperatures up to 83 degrees when facing the Sun and temperatures as low as -70 degrees when in eclipse, which are values calculated in WP2. The temperature of the lug is assumed to vary in the same way as the solar panels as it is essentially an unshielded attachment of the solar panel. These thermal fluctuations cause thermal stresses, assumed to be constrained to each hole in the back plate. The induced loads due to thermal fluctuations can be calculated through the following formula from the reader [6].

$$F_T = (\alpha_b - \alpha_p)(1 - \phi)E_b A_{ref} \Delta T \tag{5.1}$$

Here α refers to the thermal expansion coefficient, ϕ refers to the force ratio calculated in **??**, A_{ref} is the cross-sectional area of fastener which will be assumed to be equal to the area of the hole and E_b which is the material stiffness of the bolt.

This formula starts to break down for low-temperature ranges, this is because for lower temperatures metals often start becoming brittle, and the behavior of deformations can no longer be modeled through the linear expansion coefficient. However, for all aerospace applications, metals with specific crystal structures are selected to limit the ductile to brittle transition [7]. Therefore for this mission, it will be assumed that the expansion coefficient will remain constant even at low temperature ranges. Another assumption is that the temperature differences are to be calculated from a reference temperature of 15° [6] denoted as ΔT_+ and ΔT_- for an increase and decrease in temperature respectively. To make the calculations as limiting as possible the maximal temperatures mentioned earlier were used to calculate the temperature differentials. Once this force is calculated the in-plane forces defined for the bearing check must be summed to see whether the design can withstand such a thermal load. This calculation also depends on the magnitude of the expansion coefficients α. For example, assume that $\alpha_b > \alpha_p$ for a temperature contraction ΔT_- equation 5.2 will yield a positive value of force. This however makes no physical sense since the bolt is shrinking at a faster rate than the plate there is no contact between the two surfaces and therefore no force can be applied. To deal with this problem equation 5.2 can be split into two different calculations.

$$F_T = (\alpha_b - \alpha_p)(1 - \phi)E_b A_{ref} \Delta T_+ \tag{5.2}$$

and

$$F_T = -(\alpha_b - \alpha_p)(1 - \phi)E_b A_{ref} \Delta T_- \tag{5.3}$$

For the same example now equation 5.3 can be used now yielding a negative value of force which can be properly disregarded.

These equations were implemented in Python for each hole from a list to return individual forces that will vary due to the dependence of equation 5.2 on the area of the hole.

6 Optimization Process

As can be seen in the flowchart in figure 6.1, the optimization process is very long and complicated. The process is controlled by MainOptimizer.py. The file starts with an initial guess, which acts as a base design, from which things are changed to make the variables converge to an optimal design. The parameters from this initial design can be found in Listing A.3 in appendix A. The program can run in a High_Accuracy mode with small step sizes, and a Low_Accuracy mode with large step sizes. The default is to run in Low_Accuracy mode. Generally speaking, the starts with a list of materials, and for every material it makes a 'best' design using the optimizer with low accuracy. Then, the best designs of all the materials are compared with each other, and the lightest 'best' design is chosen. This design is put into the optimization algorithm again, this time with high accuracy. The resulting optimal design is the output of the program.

6.1. Optimization: Flanges

Within the LocalLoadCalculatorAndLugDesignerAndLugConfigurator.py file, the first parameters that are optimized for each material are w (defined as $2e$ from [1]), $t1$, $D1$, and h. These meaning of these variables can be found in figure 2. This optimization happens with the Minimize function from the Python package SciPy, using the 'SLSQP' optimization algorithm. The goals for this optimization is to reduce mass, so there is function that can calculate the mass given a combination of parameters. There are also constraints, to make sure the optimizer does not produce unreasonable designs. More about this process can be found in subsection 3.2.2.

Once the w, $t1$, $D1$ and h parameters were chosen, a flange length needed to be calculated. The calculation for this can be found in subsection 3.2.2.

The result from the LocalLoadCalculatorAndLugDesignerAndLugConfigurator.py file is an array of designs. Every entry in this array is an object from the class DesignInstance (see Listing A.1 in appendix A). These objects are called 'Designs', and every Design object contains its parameters. Every entry in the aforementioned array contains the Design Object that is optimal for the given material. So if the program starts with 6 different materials under consideration, then the LocalLoadCalculatorAndLugDesignerAndLugConfigurator.py generates an array with 6 different Designs. The first design will be optimal for the first material under consideration, the second design will be optimal for the second material under consideration, etc.

6.2. Optimization : Back-plate and fasteners

The second optimization happens in the MainOptimizer.py file itself. The following text will describe the optimization process starting with a design object for one material. This process will happen multiple times, once for every material, so the output will again be an array of design objects.

The optimization first applies the bearing check as described in section 4.2, and if the design fails this check, that means that the thickness of the backplate needs to increase. If it passes this check, it can go on to the next check, which is the Pull Through Check, as described in section 4.3. If it fails this check it changes the coordinates of the holes as described in section 4.3, or it can also increase the thicknesses of the backplate and the vehicle wall. If it *does* pass the Pull Through Check, then goes into a new loop. This loop repeats the following cycle: Calculating the fastener size, calculating the thermal loads, using the thermal loads to redo the Bearing Check, and finally redo the Pull Through check. The loop stops once both the Bearing Check and the Pull Through Check are passed with the thermal loads taken into account. The selection of the fasteners is explained in section 4.4. Once the fasteners are selected, the bolt sizes are used to calculate the thermal loads as described in section 5.1. The resulting Thermal Loads are then used by the Bearing Check function to see whether the designs also passes the Bearing Check with the Thermal Loads applied. This is also explained in section 4.2. If the Bearing Check with the thermal loads fails, then the thickness of the backplate neecs to increase, and if it passes, the Pull Through Check is applied again. The Pull Through Check changes the hole coordinates again, if necessary.

Once both the Bearing Check with thermal loads and the Pull Through Check are passed, the thicknesses of the backplate, the vehicle wall, and the coordinates of the holes are known. It should be noted that the t_3 cannot be changed in the design of the RODEO spacecraft, because that design is already done. However, the optimizer will still recommend increasing t_3 anyway, if it is certain that t_3 will fail because of Pull Through Failure.

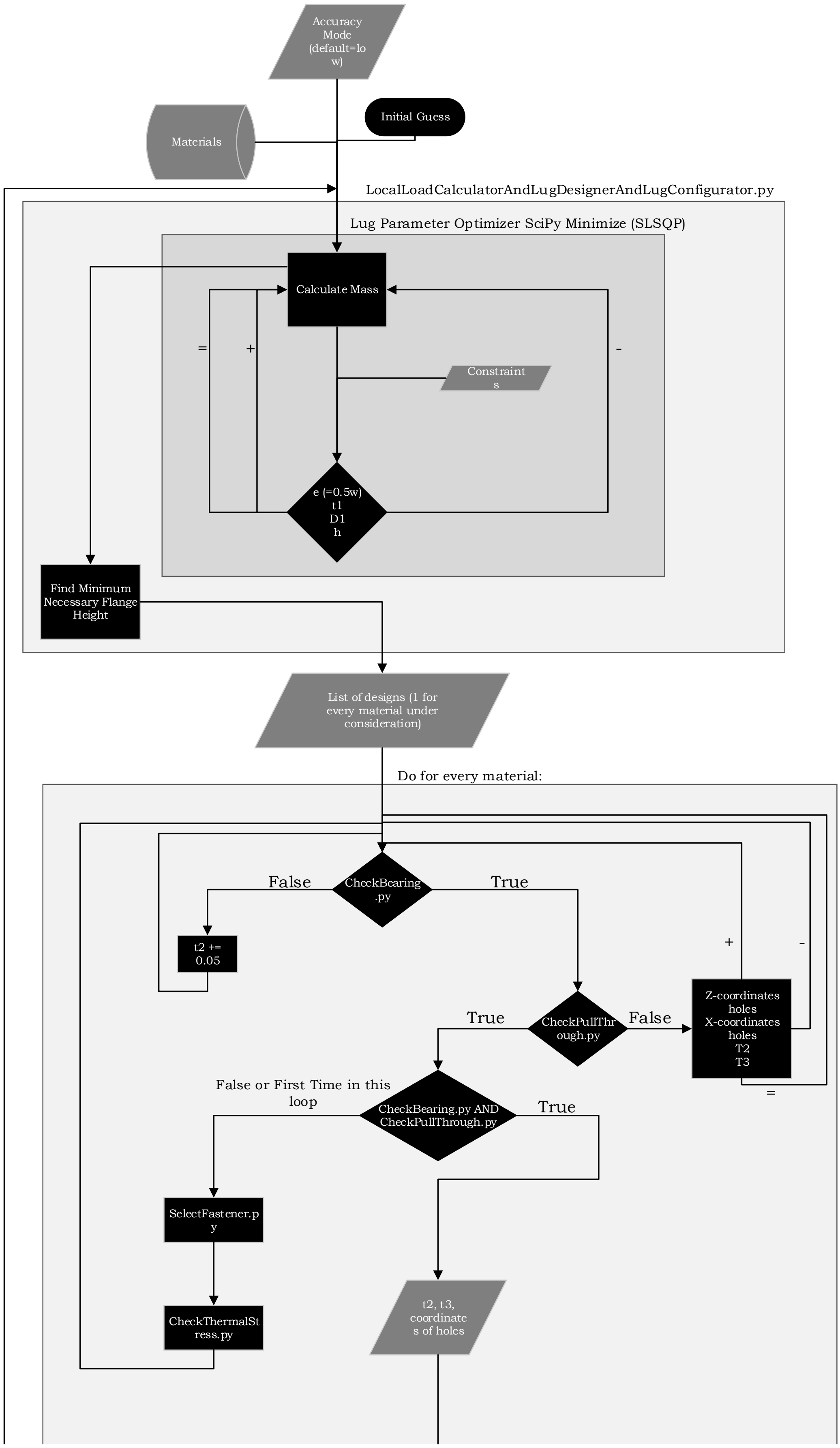
Because the hole coordinates are chosen by the Pull Through Check function in such a way that the fastener under the most stringent loads does not fail, that means that the other holes are under a lower load than their limiting failure load. This also means that these fasteners could become smaller. This is done by the function SelectFastener.check_size_red uction_possibility.

At this point in the process, the coordinates and sizes of the holes are known, but the size of the bottomplate is not known. Therefore, the function SelectFastenerConfiguration.O ptimize_holes calculates the required length and width for the bottomplate such that the holes have enough clearance from the edge as explained in section 4.1.

The next step would be to calculate the Margins of Safety as required in WP4.12. This was not done, because nowhere in the process it is known what the maximum allowable loads would be. This does not mean that our Margin of Safety is zero: safety factors of 1.1 were applied for all material properties, and safety factors of 1.25 were applied for all forces and moments.

Now, the Post Processor calculates the volume of the design with a package called CadQuery, as explained in chapter 8. This volume is multiplied by the density of the material to calculate the mass. The Post Processor also provides a visualization of the design in MatPlotLib, and exports the design to stl, step, and dxf files.

After this whole process, the array with design objects is the result. These objects are compared on the basis of there mass. All properties of all designs are printed to the console. The design with the lowest mass will be optimized again in higher accuracy. Finally, the the parameters of this high accuracy design are printed, and the design is exported to stl, step and dxf files. When this is done, the program stops.


Accuracy Mode (default=low)
Initial Guess
Materials
LocalLoadCalculatorAndLugDesignerAndLugConfigurator.py
Lug Parameter Optimizer SciPy Minimize (SLSQP)
Calculate Mass
=
+
-
Constraints
e (=0.5w)
t1
D1
h
Find Minimum Necessary Flange Height
List of designs (1 for every material under consideration)
Do for every material:
False
CheckBearing.py
True
t2 += 0.05
+
-
True
CheckPullThrough.py
False
Z-coordinates holes
X-coordinates holes
T2
T3
=
False or First Time in this loop
CheckBearing.py AND CheckPullThrough.py
True
SelectFastener.py
CheckThermalStress.py
t2, t3, coordinates of holes

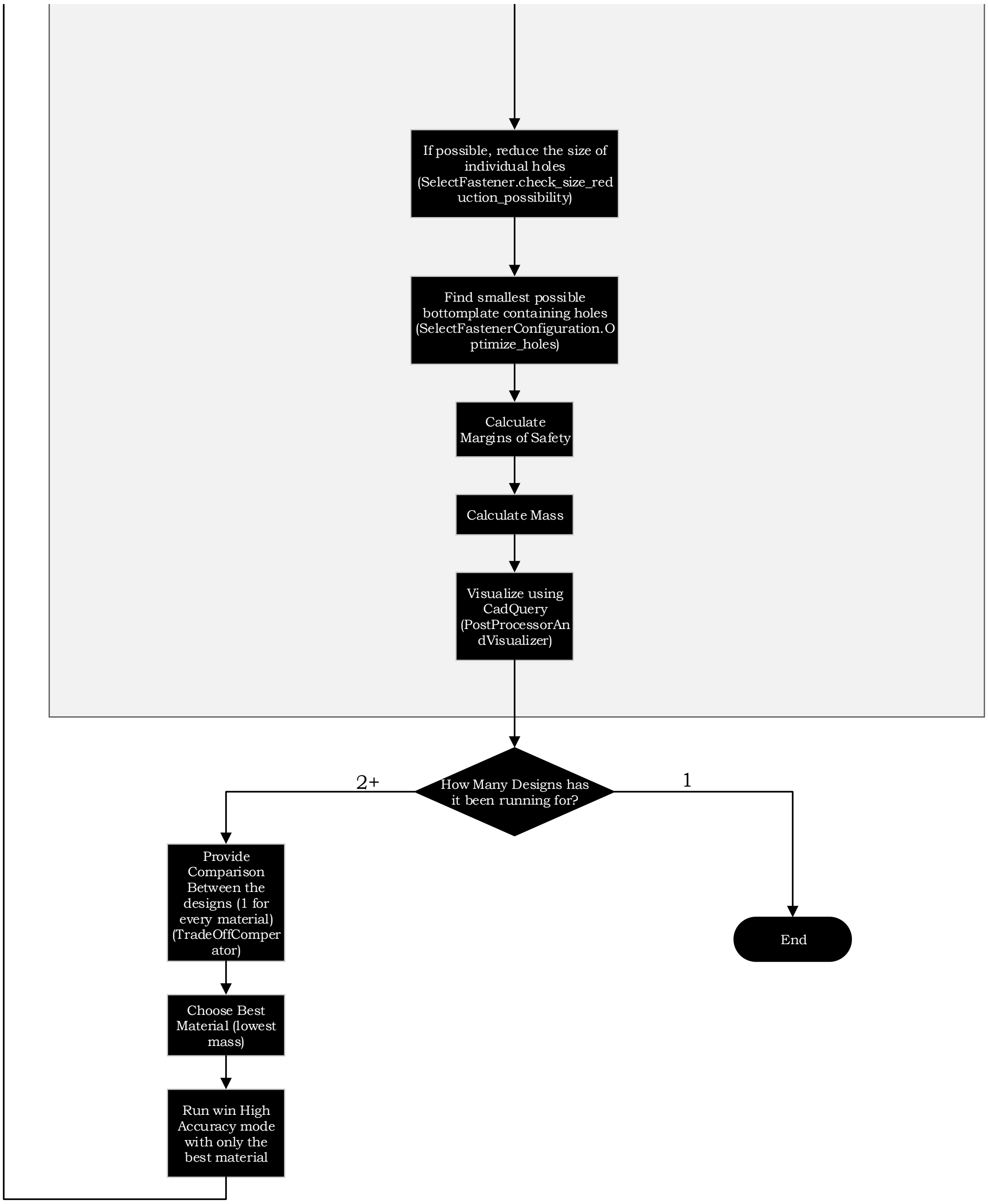


Figure 6.1: *Flowchart of the MainOptimizer*

7

Material Selection

The material for the fastener and the lugs will be selected after a comparison between all the suitable materials found, in section 7.1

7.1. Material Selection: Fasteners

The list of materials suitable to be used in the fasteners is generated by an inspection process. Initially, the companies manufacturing aerospace-grade fasteners are sourced using the industrial sourcing platform Thomasnet.com (‘https://www.thomasnet.com/catalogs/keyword/aerospace-fasteners/‘). Then, the list of the materials that these companies use for producing fasteners is noted and a trade-off is conducted. This method is used because, essentially, an off-the-shelf fastener is decided to be used and these materials are restricting. An off-the-shelf fastener is decided to be used instead of going through a detailed design process of the fasteners. This decision is made due to time constraints and because there are many proven and documented fasteners with different configurations in the market that can be chosen. The selected materials for the fasteners are 18-8 Stainless Steel, 316 Stainless Steel, Yellow Brass, Grade 5 Titanium , and Aluminum 7075. The properties of the selected materials can be seen in table 7.1.

Table 7.1: *Material options that can be used for the fasteners and their properties.*

Material	**Youngs Modulus** (GPa)	**Density** (kg/m^3)	**Thermal Expansion Coefficient** ($10^{-6}/K$)	**Ultimate Tensile Strength** (MPa)	**Elastic Limit** (Mpa)	**Resistance Factors**
316 Stainless Steel[8]	190-205	7870-8070	15-18	480-620	170-310	Excellent
18-8 SS [9]	193	7930	17.2-17.8	515-620	205-310	Respectable but not for salty environments
Carbon Steel[10]	200	7870	11.5 at 20 °C /12.2 at 250 °C	540	415	Excellent
Titanium (Grade 5) [11]	113.8	4430	8.6 at 20 °C/ 9.2 at 250 °C	950	880	Excellent for corrosion but poor with wear
Brass (Yellow) [12]	76	8470	21 in the 20-100 °C range	260	90	Good corrosion resistance
Aluminium 7075 [13]	71.7	2810	25.2 in the 20-300 °C range	572	503	Moderate

Table 7.2: *Trade-off table of the materials that can be used for the fasteners.*

Material	**Youngs Modulus**	**Density, (Weightage 2)**	**Thermal Expansion (Weightage 2)**	**Ultimate Tensile Strength**	**Elastic Limit**	**Resistance Factors**	**SCORE**
316 Stainless Steel	1	10	6	3	5	1	**26**
18-8 SS	3	8	8	2	4	2	**27**
Carbon Steel	2	6	4	5	3	1	**21**
Titanium (Grade 5)	4	4	2	1	1	3	**15**
Brass (Yellow)	5	12	10	6	6	3	**42**
Aluminium 7075	6	2	12	4	2	4	**30**

The trade-off is performed in table 7.2. In the trade-off a score from one to six is given to each material depending on how good are the material‘s properties with respect to the other materials‘ in that class. A score of 1 is the best score a material can get for a property and 6 is the worst. This means that ultimately, the material with the lowest score is going to be selected. Regarding density and the thermal expansion coefficient, a weightage of two is used after the scoring was done, and the final score for that category is shown in the table. A different weightage was used because weight and the thermal expansion coefficient are more critical aspects of the design due to the design objectives. Regarding the density, low density scores lower as lightweight structure is desired to reduce cost. High Young’s modulus scores lower as this will result in lower compliance in the parts. High elastic limit scores lower as high elastic limit means the material will deform elastically under a greater range of stresses. High ultimate tensile stress will score lower because higher the ultimate tensile stress higher the required stress for fracture. Therefore, even if the elastic limit is surpassed, the fastener will need to be subjected to greater stresses to fracture with a higher ultimate tensile stress. Lower thermal expansion coefficient scores lower. The operational temperature range is given in section 5.1 as -70 to 83°C. If the fasteners are assumed to be attached in standard temperature of 25 °C. The thermal stresses created will be higher with a high thermal expansion coefficient when the temperature of the fastener increases to 83°C. This is to be avoided so a material with a lower thermal expansion coefficient is desired.

For the resistance aspects; Stainless steel 316 and carbon steel have excellent corrosion and wear resistance; 18-8 SS has excellent wear resistance but corrodes in saline environments, but for space missions, the lug will not be in a saline environment except for possibly a very short period during the launch phase so it scores almost excellent. Brass has lower resistance to corrosion than titanium but better wear resistance. Titanium scores excellent with corrosion but poor with wear so it requires treatment to increase resistance to wear. In order to increase the wear resistance of titanium, the naturally occurring thin oxide layer of the metal will be enhanced by anodising in acid electrolyte[14]. This technique will be used as it is relatively cheap to apply, it is carried out at room temperature and so does not result in any distortion of the component being treated, and it does not have any adverse effect on the fatigue resistance of the titanium [14].

7.2. Material Selection: Lug

The selection of the material for the lug depends on the procedure described in chapter 3 and chapter 4. Here, the flanges and backplate were designed based on the limiting loads and resulted in a set of configurations for the different materials. Based on the mass of each configuration and the characteristics of the material extracted from Ansys Granta [15], the optimal material(s) can be selected through a trade-off analysis.

As specified during chapter 6, the optimizer aims to find a design that fits the requirements while minimizing mass. Table 7.3 contains the material properties and the total mass of the lug configuration calculated in *low accuracy*. As explained in previous chapters, using *low accuracy* restricts the optimizer and reduces the time required to obtain results.

Table 7.3: *Properties of available materials for the lug design and resultant mass of chosen lug configuration.*

Material	Mass $[g]$	Density $[kg/m^3]$	Young's Modulus $[GPa]$	Ultimate Tensile Strength $[MPa]$	Tensile Yield Strength $[MPa]$	Thermal Expansion Coefficient $[10^{-6}/K]$
2014-T6	99.29	2800	73	483	414	24.4
7075-T6	96.17	2810	71.7	572	503	25.2
4130 Steel	139.78	7850	200-215	670	435	13.7
8630 Steel	165.72	7850	190-210	620	550	11.2
2024-T4	85.20	2780	73	469	324	24.7
2024-T3	91.33	2780	73.1	483	345	24.7
356-T6 Aluminium	197.32	2670	72.4	234	165	23.2

From table 7.3, several designs are obtained. Taking into account the safety factors, it can be safely assumed that these meet the requirements. The trade-off shall be performed based primarily on the mass of the component, so only the aluminium alloys of series 2000 and 7000 will be assessed, since the remaining exhibit high masses. Table 7.4 displays the trade-off system that will be followed.

Table 7.4: *Trade-off system displaying average weights*

Material	Mass [3]	Specific Stiffness [3]	Thermal Expansion Coefficient [1]	Fatigue Strength (10^7 Cycles) [2]	Cost [1]
2014-T6	99.29	25.7-27.5	24.4	119-133	2.98-4
7075-T6	96.17	24.6-27.2	25.2	152-168	5.54-7.48
2024-T4	85.20	26-27.8	24.7	133-147	3.01-4.03
2024-T3	91.33	26-27.4	24.7	118-168	3.01-4.03

The average weights have been decided based on the importance of each of the parameters. The mass of the component and specific stiffness of the material, which equals the ratio of Young's Modulus and density, shall have an average weight of 3. Also, the lug is to withstand loads for the entirety of the mission duration and will be affected by fatigue due to thermal and mechanical stress, therefore the fatigue strength after 10^7 cycles will be considered with an average weight of 2. Thermal expansion has already been considered for the design and therefore poses the lowest priority for the trade-off. Table 7.5 contains the score system.

Table 7.5: *Trade-off table displaying the score*

Material	Mass [3]	Specific Stiffness [3]	Thermal Expansion Coefficient [1]	Fatigue Strength (10^7 Cycles) [2]	Cost [1]	Total
2014-T6	4	4	1	4	1	34
7075-T6	3	3	4	1	4	28
2024-T4	1	1	3	3	3	18
2024-T3	2	2	3	2	3	22

The trade-off table in table 7.5 displays the amount of points scored by each material for the different features. The scoring system works as in section 7.1, the material with the best performance for the given property scores the lowest, a one, while the one performing poorly scores a four. The total amount of points is calculated by summing up the amount of points multiplied by a factor representing the importance of the feature compared to the others. The material scoring the lowest is the aluminium alloy 2024-T4. Therefore, this material is selected to manufacture the lug.

7.3. Final Design

Having chosen the material for the lug, the mass and configuration can be recalculated in *high accuracy* mode, which produces more detailed parameters and mass. The parameters for the final configuration are shown in table 7.6.

Table 7.6: *Final Configuration*

Material	2024-T4
Mass [g]	79.050
Number of Lug(s)	2
Number of Flange(s)	2
Number of Fasteners	4
h [mm]	10.000
t1 [mm]	1.001
t2 [mm]	1.050
t3 [mm]	2.000
D1 [mm]	5.000
w [mm]	15.000
Flange Height [mm]	22.500
Plate Length [mm]	68.000
Plate Width [mm]	191.000
Distance between lugs [mm]	1400.000
Hole Coordinates [mm]	(4.443, 4.570)
	(4.443, 186.430)
	(63.557, 4.570)
	(63.557, 186.430)
D2 (holes) [mm]	1.300
	1.700
	2.200
	1.300
Nut type	Hexagonal
Hole Type	Nut-Tightened
Material	Titanium (Grade 5)
Bolt Length [mm]	8
Bolt Manufacturer	Extreme B&F
Bolt Type	M1.2
	M1.6
	M2
	M1.2

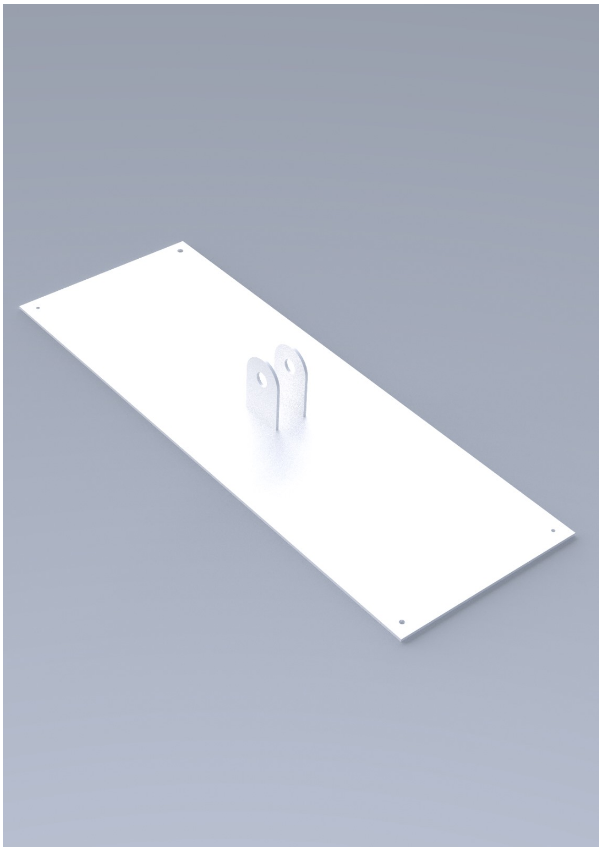


Figure 7.1: *Final Design*

The high-accuracy optimisation leads to a lightweight component that should withstand the loads. It is important to consider that some of the parameters look unreasonable. The distance between flanges or their width could be considered too low, which should be assessed in future iterations.

Despite having considered several safety factors, the design is based on yield. This means that the lug will not break during its lifetime, but it is very inclined to deform or bend as design parameters such as maximum deformation angles were not considered. To obtain an optimal design, it is recommended to subject the component to testing and further iterate the design if required. Including more design parameters into the constraints of the optimiser would improve the design.

8 Post Processing and Visualization

When a new configuration was outputted by the optimization software, a 3D model of this configuration was generated by the CadQuery package. This package also exported the 3D model to stl, which could be opened in visualization software such as STL viewer. The necessity of this software was threefold:

- Firstly due to the large amount of parameters a visual representation of the lug was more useful than the value of the geometric parameters. This was especially useful in the debugging of the optimizer as a ”sanity” check for the lug’s calculated parameters.
- The software allows the calculation of the volume of the part which can then be used to calculate its mass very quickly, and nonsensical values for the mass also aided the debugging of the optimizer.
- The CadQuery package also exports the 3D model as a step file, which was imported into CATIA to make the technical drawings.

The CadQuery package provides a way to turn the parameters into a 3D model. CadQuery works a bit different than an normal parametric modelling CAD program, because all 3D modelling steps (such as extrude, make a hole, etc.) were done by code. The advantage of this is that this code can automatically use the parameters as outputted from the optimization process, and it would not be necessary to manually make a 3D model in a normal CAD program each time a new design is made. The code that generates the 3D model can be found in Listing A.10 in appendix A.

9 Conclusion

This project aimed to determine the design for the attachment of the solar panels for the RODEO mission to the main spacecraft body. To understand the problem better the limiting load cases that the attachment should withstand were first analyzed in chapter 2. It was concluded that the loads experienced due to the launch phase of the mission were limiting and this load case would drive the design. Next in chapter 3, the configuration and the design of the lugs were selected through the optimization of the dimensions to withstand the loads whilst minimizing the mass. The next chapter focuses on the design of the fasteners and the dimensions of the back plate of the lug. Functions in Python were created to verify each of the individual constraints, ensuring a comprehensive assessment of the bearing stress, pull-through forces, and stress concentrations. An iterative process of updating an initial design based on the output of these functions is described in chapter 6. After this initial optimization, the thermal effects of the temperature changes experienced by RODEO were also added to the fastener checks. The same optimization was then performed again to identify the optimal material for the back plate and the fasteners. This eventually led to the final design described in section 7.2 that involved Titanium fasteners and bolts and an aluminum 2024-T4 lug.

Several steps that could be taken to improve the code. One of these is to use the Python package NUMBA, which would convert the slow Python code to a faster species of code. NUMBA also has the possibility to use CUDA technology, which would run the code on the GPU instead of the CPU, which would be faster for some kinds of calculations. This would not improve the speed of the code on our laptops, as our laptops are not compatible with CUDA, but it would massively reduce the time needed to run the code on a Microsoft Azure server or AWS EC2 server. Another way that the code could be improved would be to provide more data *during* the optimization process, rather then only after. This way you can diagnose problems, and stop the optimization process earlier to change things. An example of data that could be shared during the process could be margins of safety, which could be tracked over time.

Besides using packages to speed up running the code, the code itself could also be improved to make it more efficient. In the LocalLoadCalculatorAndLugDesignerAndLugConfigurator.py there are many for-loops that go through every possibility to find the best combination of variables. The range of possibilities it goes through can be reduced in size, because it also includes some combinations of variables that are obviously not optimal (very large lengths/widths, very small lengths/widths, etc.). Another way to make the code more efficient would be to use dynamic step sizes, that first optimizes the variables roughly (with large step sizes), and then the best combination of variables is used to run the optimizer again, with a finer step size. This way it would not be necessary to go through all possibilities with the fine step size, and would speed up the code.

Figure 9.1: *Topology Optimization*

One whole other way to do the optimization would be to use the technology Topology Optimization. This technology strategically adds and removes material in certain places, as to reduce the mass as much as possible, given the loads. The results from Topology Optimization look very different than what is normally designed by engineers: the designs are very organic. This can be seen in figure 9.1.

References

[1] Kassapoglou, C., "AE2111-I Aerospace System Design Spacecraft WP4: A solar array or an RTG attachment", Delft University of Technology, Faculty of Aerospace Engineering, 2023.

[2] European Cooperation for Space Standardization, "ECSS-E-ST-32-10C Rev.2 Corr.1 – Structural factors of safety for spaceflight hardware", European Cooperation for Space Standardization, 2019.
https://ecss.nl/standard/ecss-e-st-32-10c-rev-2-corr-1-structural-factors-of-safety-for-spaceflight-hardware-1-august-2019/.

[3] "Analysis and Design of Flight Vehicle Structures by E F Bruhn PDF | PDF",
https://www.scribd.com/doc/220947115/Analysis-and-Design-of-Flight-Vehicle-Structures-by-E-F-Bruhn-pdf.

[4] "Hex Bolt Dimensions", Corp., B. S. . B.
https://www.badensteel.com/bolt-dimensions.

[5] *ECSS-E-HB-32-23A: Threaded fasteners handbook (16 April 2010)*, .
https://ecss.nl/hbstms/ecss-e-hb-32-23a-threaded-fasteners-handbook/.

[6] Akay, I., and Vanhamel, J., "AE2111-I Aerospace System Design Spacecraft WP1-3", Delft University of Technology, Faculty of Aerospace Engineering, 2023.

[7] Ashby, M. F., and Jones, D. R. H., *Materials: Engineering, Science, Processing, and Design*, Butterworth-Heinemann, 2019.

[8] "Stainless Steel - Grade 316 (UNS S31600)", Azomaterials.
https://www.azom.com/article.aspx?ArticleID=863.

[9] "18/8 Stainless Steel, Grade 18-8 SS Properties & Meaning",
18/8StainlessSteel,Grade18-8SSProperties&Meaning.

[10] "Carbon Steel Mechanical Properties", Ezlokltd.
https://www.ezlok.com/carbon-steel-properties.

[11] Boyer, R., Welsch, G., and Collings, E. W., *Materials Properties Handbook: Titanium Alloys*, ASM International, 1994.

[12] "Yellow Brass (UNS C26800)", Azonetwork.
https://www.azom.com/article.aspx?ArticleID=6360.

[13] ASMInternational, *Metals Handbook, Vol.2 - Properties and Selection: Nonferrous Alloys and Special-Purpose Materials*, 10th ed., 1990.
https://asm.matweb.com/search/SpecificMaterial.asp?bassnum=ma7075t6.

[14] "Titanium Alloys - Wear Resistance", Azomaterials.
https://www.azom.com/article.aspx?ArticleID=1219.

[15] "Ansys Granta EduPack software", ANSYS, Inc., www.ansys.com/materials.

Appendices

A Code

The code used in the project is listed below in multiple code blocks. To run the code, execute MainOptimzer.py after installing all the packages in Requirements.txt. The code can also be cloned from the GitHub Repo ”HuiLucas/LugDesign” [1]. If you just want to run the code without an IDE, you can pull the Docker Repo ”lhuirne/lug_design3” [2].

```
from InputVariables import materials_fasteners
class DesignInstance:
    def __init__(self, h, t1, t2, t3, D1, w, material, n_fast, length, offset, flange_height,
     hole_coordinate_list, \
                 D2_list, yieldstrength, N_lugs, N_Flanges, Dist_between_lugs=0,
    bottomplatewidth=100, shearstrength=550):
        self.h = h
        self.t1 = t1
        self.t2 = t2
        self.t3 = t3
        self.D1 = D1
        #self.l = l #length of backplate
        self.w = w
        self.material = material
        self.n_fast = n_fast
        self.length = length
        self.offset = offset
        self.flange_height = flange_height
        self.hole_coordinate_list = hole_coordinate_list
        self.n_fast = len(hole_coordinate_list)
        self.D2_list=D2_list
        self.yieldstrength=yieldstrength
        self.N_lugs = N_lugs
        self.N_Flanges = N_Flanges
        self.Dist_between_lugs = Dist_between_lugs
        self.bottomplatewidth = bottomplatewidth
        self.shearstrength = shearstrength
        self.MS = None
class Load:
    def __init__(self, F_x, F_y, F_z, M_x, M_y, M_z):
        self.F_x = F_x
        self.F_y = F_y
        self.F_z = F_z
        self.M_x = M_x
        self.M_y = M_y
        self.M_z = M_z

Launch_loads = Load(346.9,346.9,1040.7,653.9,653.9,0)

# Your main file

# Nut type can either be ”Hexagonal” , ”Cylindrical” and hole type can either be ”Nut-
    Tightened” or ”Threaded hole”

class FastenerType:
    def __init__(self, material_name, nut_type=None, hole_type=None):
        self.material = None
        self.youngs_modulus = None
        self.thermal_coeff = None
        self.yield_stress = None
```

[1] https://github.com/HuiLucas/LugDesign.git
[2] https://hub.docker.com/repository/docker/lhuirne/lug_design3/general

```python
        self.nut_type = nut_type
        self.hole_type = hole_type
        self.type_bolt = "unknown"

        if nut_type not in ["Hexagonal", "Cylindrical"]:
            print("Nut type can either be 'Hexagonal' or 'Cylindrical'")
        else:
            self.nut_type = nut_type

        if hole_type not in ["Nut-Tightened", "Threaded hole"]:
            print("Hole type can either be 'Nut-Tightened' or 'Threaded hole'")
        else:
            self.hole_type = hole_type

        if material_name:
            self.set_material(material_name)

    def set_material(self, material_name):
        for material in materials_fasteners:
            if material["Material"] == material_name:
                self.material = material_name
                self.youngs_modulus = material["Youngs Modulus (GPa)"]
                self.thermal_coeff = material["Thermal Expansion (10^-6)/K"]
                self.yield_stress = material["Ultimate Tensile Strength (MPa)"]
                return
        print("Material is not in InputVariables/materials_fastener")
```

Listing A.1: *DesignClass.py code*

```python
# This File is going to contain the input variables for the whole program, like material properties of the material
# under consideration.
#hello

Material = ['2014-T6(DF-L)', '2014-T6(DF-LT)', '2014-T6(P)', '7075-T6(P)', '7075-T6(DF-L)', '7075-T6(DF-LT)',
            '4130 Steel', '8630 Steel', '2024-T4', '356-T6 Aluminium', '2024-T3']
sigma_yield = [414, 414, 414, 503, 503, 503, 435, 550, 324, 165, 345]
shear_strength = [290, 290, 290, 331, 331,331, 345, 345, 283, 143, 283] #MPa
Density = [2800, 2800, 2800, 2810, 2810, 2810, 7850, 7850, 2780, 2670, 2780]

materials_fasteners = [
    {"Material": "316 Stainless Steel", "Youngs Modulus (GPa)": 190, "Density (kg/m^3)": 8070,
     "Thermal Expansion (10^-6)/K": 18, "Ultimate Tensile Strength (MPa)": 620,
     "Elastic Limit(Mpa)": 170, "Resistance Factors": "Excellent"},

    {"Material": "18-8 SS", "Youngs Modulus (GPa)": 193, "Density (kg/m^3)": 7930,
     "Thermal Expansion (10^-6)/K": 17.8, "Ultimate Tensile Strength (MPa)":  620,
     "Elastic Limit(Mpa)":  310, "Resistance Factors": "Respectable but not for salty environments"},

    {"Material": "Carbon Steel", "Youngs Modulus (GPa)": 200, "Density (kg/m^3)": 7870,
     "Thermal Expansion (10^-6)/K": 11.5, "Ultimate Tensile Strength (MPa)": 540, "Elastic Limit(Mpa)": 415,
     "Resistance Factors": "Excellent"},

    {"Material": "Titanium (Grade 5)", "Youngs Modulus (GPa)": 113.8, "Density (kg/m^3)": 4430,
     "Thermal Expansion (10^-6)/K": 8.6, "Ultimate Tensile Strength (MPa)": 950, "Elastic Limit(Mpa)": 880,
     "Resistance Factors": "Excellent for corrosion but poor with wear"},

    {"Material": "Brass (Yellow)", "Youngs Modulus (GPa)": 76, "Density (kg/m^3)": 8740,
     "Thermal Expansion (10^-6)/K": 21, "Ultimate Tensile Strength (MPa)": 260, "Elastic Limit(Mpa)": 90,
     "Resistance Factors": "Good corrosion resistance"},

    {"Material": "Aluminium 7075", "Youngs Modulus (GPa)": 71.7, "Density (kg/m^3)": 2810,
     "Thermal Expansion (10^-6)/K": 25.2, "Ultimate Tensile Strength (MPa)": 572, "Elastic Limit(Mpa)": 503,
```

```
    "Resistance Factors": "Moderate"}
]

materials_lug = [
    {'material': '2014-T6(DF-L)', 'thermal_expansion_coefficient': 23 ,'elastic module': 73.1 },
    {'material': '2014-T6(DF-LT)', 'thermal_expansion_coefficient': 23,'elastic module': 73.1},
    {'material': '2014-T6(P)', 'thermal_expansion_coefficient': 23,'elastic module': 73.1},
    {'material': '7075-T6(P)', 'thermal_expansion_coefficient': 23.4,'elastic module': 71.7},
    {'material': '7075-T6(DF-L)', 'thermal_expansion_coefficient': 23.4,'elastic module': 71.7},
    {'material': '7075-T6(DF-LT)', 'thermal_expansion_coefficient': 23.4,'elastic module': 71.7},
    {'material': '4130 Steel', 'thermal_expansion_coefficient': 11.1,'elastic module': 205},
    {'material': '8630 Steel', 'thermal_expansion_coefficient': 11.3,'elastic module': 200},
    {'material': '2024-T4', 'thermal_expansion_coefficient': 23.2,'elastic module': 73.1},
    {'material': '356-T6 Aluminium', 'thermal_expansion_coefficient': 23.8,'elastic module': 72.4},
    {'material': '2024-T3', 'thermal_expansion_coefficient': 21.6,'elastic module': 73.1}
]
```

Listing A.2: *InputVariables.py code*

```
# This file will change the design based on the part checks. It will start from an initial design, then perform all
# the checks as written in the other software components, and improve the design if possible with iterations.
import copy

import CheckBearing, CheckThermalStress, CheckPullThrough, GlobalLoadsCalculator, InputVariables, \
    PostProcessorAndVisualizer, SelectFastener, TradeOffComperator, \
    DesignClass, LocalLoadCalculatorAndLugDesignerAndLugConfigurator
import numpy as np
import SelectFastenerConfiguration

# !!!!!!!!!!!! TBD:
# Done !!!!!!!!!!!! For CheckPullThrough: shearstrength is now set for one material, but needs to be done for other materials as well
# Done !!!!!!!!!!!! optimize/calculate dist_between_lugs (is now set at the beginning, and never changed). Maybe set equal to upper limit of what fits on the satellite?
# Done !!!!!!!!!!!! Optimize (?) D2_holes. Is now set at the beginning, and does not change troughout the process. However, it was
# chosen to change the thickness t2 instead of the diameters of the holes, but maybe it is still possible to do both? I was thinking, maybe we could make a function that looks whether it is possible to reduce the size of a given hole (in
# discrete steps that correspond to  bolt diameters), given an existing design. Then this function could be applied all the way at the end
# to reduce the size of the the holes that are not limiting. Such a function would need to find out if the design with a smaller reduced hole
# would still pass the Bearing Check (incl. updated fastener design & thermal loads) and Pull Through Check, and if that is the case, then the size of that hole would actually be decreased.
# !!!!!!!!!!!! Check EVERYTHING, make sure no mistakes in calculations. Look for mistakes in the code. Confirm results by performing checks on the resulting designs by hand.
# !!!!!!!!!!!! Run with high_accuracy = True (once) to find out if there's strange behaviour
# !!!!!!!!!!!! Finish comparison between materials (WP4.13)
# Done !!!!!!!!!!!! Set constraint that h >= 0 if N_lugs == 2
# Done !!!!!!!!!!!! Provide list of Margins of Safety, as in WP4.11


# ------------------------------------------------------------------------------------------------------------------------
# Do not change:
initial_design = DesignClass.DesignInstance(h=30, t1=5, t2=0.05, t3=2, D1=10, w=80, material="metal", n_fast=4, \
                                            length=10, offset=20,flange_height=80, \
                                            hole_coordinate_list=[(3, 35), (3, 65), (7, 35), (7, 65)], \
```

```
                                              D2_list=[2.2, 2.2, 2.2, 2.2], yieldstrength=83, N_lugs=2,N_Flanges=2, bottomplatewidth=100)
if initial_design.N_lugs ==1:
    initial_design.D2_list = [2.2,2.2,2.2,2.2]
# -----------------------------------------------------------------------------------------------------------------------

if initial_design.N_Flanges ==2:
    initial_design.offset = (initial_design.length - initial_design.t1 - initial_design.h)/2
else:
    initial_design.offset = (initial_design.length - initial_design.t1)/2
loads_with_SF = DesignClass.Load(433.6,433.6,1300.81,817.34,817.34,0)

fastener_array =[]
design_array2 = []
design_array = LocalLoadCalculatorAndLugDesignerAndLugConfigurator.Optimize_Lug(InputVariables.Material, \
                                                                  InputVariables.sigma_yield, InputVariables.Density,\
                                                                  initial_design, loads_with_SF, False)

for designindex in range(len(design_array)):
    out1 = copy.deepcopy(design_array[designindex])
    print(out1.h, out1.t1, out1.t2, out1.t3, out1.D1, out1.w, out1.length, out1.offset, out1.flange_height, out1.yieldstrength, out1.material, out1.Dist_between_lugs, out1.N_lugs)


    #check1 = checkbearing without thermal loads, follow advice from result

    check1 = False
    if not CheckBearing.check_bearing_stress(out1, loads_with_SF, [0, 0, 0,
                                                                    0]) == "Bearing Stress Check Failed, increase the thickness of the backplate ":
        check1 = True
    counter1 = 0
    print(check1, counter1, out1.t2)
    while check1 == False and counter1 < 1000:
        out1.t2 += 0.05
        if not CheckBearing.check_bearing_stress(out1, loads_with_SF,[0,0,0,0]) == "Bearing Stress Check Failed, increase the thickness of the backplate ":
            check1=True
        counter1 +=1
    print(check1, counter1, out1.t2)
    #check2 = checkpullthrough, follow advice from result

    check2 = False
    counter2 = 0
    print(check2, counter2, out1.t2)
    if CheckPullThrough.check_pullthrough(out1, loads_with_SF)[0] == True:
        check2 = True
    print(check2, counter2, out1.t2)
    print("here2", out1.hole_coordinate_list)
    while check2 == False and counter2 < 1000:
        #print(out1.hole_coordinate_list)
        print(CheckPullThrough.check_pullthrough(out1, loads_with_SF)[1])
        if CheckPullThrough.check_pullthrough(out1, loads_with_SF)[1] == "decrease z":
            for ix in range(len(out1.hole_coordinate_list)):
                out1.hole_coordinate_list[ix] = (
                    out1.hole_coordinate_list[ix][0],
                    0.5 * out1.bottomplatewidth + (out1.hole_coordinate_list[ix][1] - 0.5 * out1.bottomplatewidth) * 0.98)
        elif CheckPullThrough.check_pullthrough(out1, loads_with_SF)[1] == "increase z":
            for ix in range(len(out1.hole_coordinate_list)):
                out1.hole_coordinate_list[ix] = (
                    out1.hole_coordinate_list[ix][0],
                    0.5 * out1.bottomplatewidth + (out1.hole_coordinate_list[ix][1] - 0.5 * out1.bottomplatewidth) * 1.02)
        elif CheckPullThrough.check_pullthrough(out1, loads_with_SF)[1] == "increase x":
            for ix in range(len(out1.hole_coordinate_list)):
```

```python
            out1.hole_coordinate_list[ix] = (
                0.5 * out1.length + (out1.hole_coordinate_list[ix][0] - 0.5 * out1.length ) * 1.02,
                out1.hole_coordinate_list[ix][1])
    elif CheckPullThrough.check_pullthrough(out1, loads_with_SF)[1] == "decrease x":
        for ix in range(len(out1.hole_coordinate_list)):
            out1.hole_coordinate_list[ix] = (
                0.5 * out1.length + (out1.hole_coordinate_list[ix][0] - 0.5 * out1.length ) * 0.98,
                out1.hole_coordinate_list[ix][1])
    elif CheckPullThrough.check_pullthrough(out1, loads_with_SF)[1] == "increase t2":
        out1.t2 += 0.05
    elif CheckPullThrough.check_pullthrough(out1, loads_with_SF)[1] == "increase t3":
        out1.t3 += 0.1
    else:
        check2 = True

    counter2 += 1
print(check2, counter2, out1.t2)




checklist = [False, False]

# checklist = [check1, check2]
counter3 = 0
firsttime = True
while not checklist == [True, True] and counter3<100:
    out1.fasteners = DesignClass.FastenerType("Titanium (Grade 5)","Hexagonal","Nut-Tightened")
    philist = SelectFastener.calculate_force_ratio(out1.fasteners, out1,out1.material,"7075-T6(DF-LT)")[0]
    thermal_loads = CheckThermalStress.thermal_stress_calculation(out1, 150, -90, 15, philist
                                                                  ,material_fastener=out1.fasteners.material,
                                                                  material_plate=out1.material)[0]
    check1 = False
    if not CheckBearing.check_bearing_stress(out1, loads_with_SF, thermal_loads) == "Bearing Stress Check Failed, increase the thickness of the backplate ":
        check1 = True
    counter1 = 0
    counter4 = 0
    print(check1, counter1, out1.t2)
    while check1 == False and counter1 < 1000:
        if firsttime == False:
            out1.t2 += 0.05
            print("please work",out1.t2)
            counter4 +=1
        if not CheckBearing.check_bearing_stress(out1, loads_with_SF, thermal_loads) == "Bearing Stress Check Failed, increase the thickness of the backplate ":
            check1 = True
        counter1 += 1
    print(check1, counter1, out1.t2)
    print("here", out1.hole_coordinate_list)

    firsttime=False
    # check2 = checkpullthrough, follow advice from result
    check2 = False
    counter2 = 0
    counter5 = 0
    print("point1",check2, counter2, out1.t2, out1.hole_coordinate_list)
    if CheckPullThrough.check_pullthrough(out1, loads_with_SF)[0] == True:
        check2 = True
    print("point2",check2, counter2, out1.t2, out1.hole_coordinate_list)
    while check2 == False and counter2 < 1000:
        if CheckPullThrough.check_pullthrough(out1, loads_with_SF)[1] == "decrease z":
            counter5 +=1
            for ix in range(len(out1.hole_coordinate_list)):
```

```
                    out1.hole_coordinate_list[ix] = (
                    out1.hole_coordinate_list[ix][0], 0.5*out1.bottomplatewidth+(out1.hole_coordinate_list[ix][1] -0.5*out1.bottomplatewidth)* 0.98)
            elif CheckPullThrough.check_pullthrough(out1, loads_with_SF)[1] == "increase z":
                counter5 +=1
                for ix in range(len(out1.hole_coordinate_list)):
                    out1.hole_coordinate_list[ix] = (
                    out1.hole_coordinate_list[ix][0],0.5*out1.bottomplatewidth+(out1.hole_coordinate_list[ix][1] -0.5*out1.bottomplatewidth)* 1.02)
            elif CheckPullThrough.check_pullthrough(out1, loads_with_SF)[1] == "increase x":
                counter5 += 1
                for ix in range(len(out1.hole_coordinate_list)):
                    out1.hole_coordinate_list[ix] = (
                    0.5*out1.length + (out1.hole_coordinate_list[ix][0]-0.5*out1.length) * 1.02, out1.hole_coordinate_list[ix][1])
            elif CheckPullThrough.check_pullthrough(out1, loads_with_SF)[1] == "decrease x":
                counter5 += 1
                for ix in range(len(out1.hole_coordinate_list)):
                    out1.hole_coordinate_list[ix] = (
                    0.5*out1.length + (out1.hole_coordinate_list[ix][0]-0.5*out1.length) * 0.98, out1.hole_coordinate_list[ix][1])
            elif CheckPullThrough.check_pullthrough(out1, loads_with_SF)[1] == "increase t2":
                counter5 += 1
                out1.t2 += 0.05
            elif CheckPullThrough.check_pullthrough(out1, loads_with_SF)[1] == "increase t3":
                counter5 += 1
                out1.t3 += 0.1
            else:
                check2 = True

            counter2 += 1
        print(check2, counter2, out1.t2)
        checklist = [check1, check2]
        counter3 += 1
        print("counters", counter1, counter2,counter3,  counter4, counter5)



    print(out1.h, out1.t1, out1.t2, out1.t3, out1.D1, out1.w, out1.length, out1.offset, out1.flange_height, out1.yieldstrength, out1.material, out1.Dist_between_lugs, out1.N_lugs, out1.bottomplatewidth)
    print(out1.hole_coordinate_list)
    #PostProcessorAndVisualizer.Visualize2(out1)
    if out1.h > out1.length and out1.N_Flanges == 2:
        changex = out1.h - out1.length
        out1.length = out1.h
        for j in range(len(out1.hole_coordinate_list)):
            deltaX = changex*0.5
            out1.hole_coordinate_list[j] = (out1.hole_coordinate_list[j][0] + deltaX, out1.hole_coordinate_list[j][1])
    changez = out1.bottomplatewidth - out1.w
    out1.bottomplatewidth = out1.w
    for j in range(len(out1.hole_coordinate_list)):
        deltaZ = changez*0.5
        out1.hole_coordinate_list[j] = (out1.hole_coordinate_list[j][0], out1.hole_coordinate_list[j][1]-deltaZ)
    #PostProcessorAndVisualizer.Visualize2(out1)
    print(out1.h, out1.t1, out1.t2, out1.t3, out1.D1, out1.w, out1.length, out1.offset, out1.flange_height, out1.yieldstrength, out1.material, out1.Dist_between_lugs, out1.N_lugs, out1.bottomplatewidth)
    print("this", out1.hole_coordinate_list)

    for m in range(len(out1.D2_list)):
        SelectFastener.check_size_reduction_possibility(out1,m,loads_with_SF)
    print("why is this:", CheckPullThrough.check_pullthrough(out1,loads_with_SF))

    out1 = SelectFastenerConfiguration.Optimize_holes(out1, False)

    MS_lug_Appendix_A = LocalLoadCalculatorAndLugDesignerAndLugConfigurator.M_S
    MS_lug_breaking_flange = 0
    MS_backupwallbearing = "allowable stress is unknown because different method is used"
```

```
    MS_backupwallbearinginclthermal = "allowable stress is unknown because different method
     is used"
    MS_backupwallpullthrough = "allowable stress is unknown because different method is used"
    MS_VehicleWallBearinginclThermal = "allowable stress is unknown because different method
     is used"
    MS_VehicleWallPullthrough = "allowable stress is unknown because different method is used
     "

    out1.MS =[]
    out1.MS.append(MS_lug_Appendix_A)
    out1.MS.append(MS_lug_breaking_flange)
    out1.MS.append(MS_backupwallbearing)
    out1.MS.append(MS_backupwallbearinginclthermal)
    out1.MS.append(MS_backupwallpullthrough)
    out1.MS.append(MS_VehicleWallBearinginclThermal)
    out1.MS.append(MS_VehicleWallPullthrough)

    #PostProcessorAndVisualizer.Visualize(initial_design)
    print(out1.hole_coordinate_list)
    #PostProcessorAndVisualizer.Visualize2(out1, designindex)

    print(out1.h, out1.t1, out1.t2, out1.t3, out1.D1, out1.w, out1.length, out1.offset, out1.
     flange_height, out1.yieldstrength, out1.material, out1.Dist_between_lugs, out1.N_lugs)
    out1.checklist = checklist
    print(checklist)
    design_array2.append(out1)
    print(designindex)
    print(out1.hole_coordinate_list)


for designindex in range(len(design_array2)):
    design_array2[designindex].volume = 0
    PostProcessorAndVisualizer.Visualize2(design_array2[designindex], designindex)
    # print(design_array2[designindex].material)
    # print(InputVariables.Material.index(design_array2[designindex].material))
    # print(InputVariables.Density[InputVariables.Material.index(design_array2[designindex].
     material)], design_array2[designindex].volume)
    # print(InputVariables.Density[InputVariables.Material.index(design_array2[designindex].
     material)]*design_array2[designindex].volume)
    design_array2[designindex].mass = InputVariables.Density[InputVariables.Material.index(
     design_array2[designindex].material)]*design_array2[designindex].volume
# trade-off stuff
TradeOffComperator.TradeOff(design_array2)
```

Listing A.3: *MainOptimizer.py code*

```

# This software component will calculate the local loads on a parametric model of the lug
     based on the given global
# loads, and then choose values for the variables of the lug such that the lug can sustain
     these loads (see WP4.3).
# This Software Component will also decide between a 1-lug or 2-lug configuration.

import numpy as np
from scipy.optimize import minimize
import math
import DesignClass
import InputVariables
from numba import jit

debug_design3 = DesignClass.DesignInstance(h=30, t1=5, t2=10, t3=2, D1=10, w=80, material="
     metal", n_fast=4, \
                                            length=200, offset=20,flange_height=80, \
                                            hole_coordinate_list=[(20, 10), (180, 30), (160,
     20), (30, 30)], \
                                           D2_list=[10, 5, 9, 8], yieldstrength=83,N_lugs=1,
     N_Flanges=2)

debug_loads = DesignClass.Load(433.6,433.6,1300.81,817.34,817.34,0)
@jit(nopython=True)
def calculate_ci(i, x):
```

```
    if i ==1:
        return 0.8534 + 0.2891 * x - 0.1511 * x ** 2 - 0.0035 * x ** 3 + 0.0174 * x ** 4 - 0.0038 * x ** 5 + 0.0002 * x ** 6
    elif i==2:
        return 1.5030 - 1.5054 * x + 1.7453 * x ** 2 - 0.9890 * x ** 3 + 0.2838 * x ** 4 - 0.0390 * x ** 5 + 0.0020 * x ** 6
    elif i ==3:
        return 0.6270 + 0.9428 * x - 0.8837 * x ** 2 + 0.3900 * x ** 3 - 0.0928 * x ** 4 + 0.0115 * x ** 5 - 0.0005 * x ** 6
    elif i ==4:
        return 0.9083 + 0.3195 * x - 0.2985 * x ** 2 + 0.0912 * x ** 3 - 0.0121 * x ** 4 + 0.0006 * x ** 5
    elif i==5:
        return 0.6115 + 1.4003 * x - 1.6563 * x ** 2 + 0.8517 * x ** 3 - 0.2273 * x ** 4 + 0.0306 * x ** 5 - 0.0016 * x ** 6
    elif i==6:
        return 0.7625 + 1.1900 * x - 1.5365 * x ** 2 + 0.7699 * x ** 3 - 0.1987 * x ** 4 + 0.0258 * x ** 5 - 0.0013 * x ** 6
    elif i ==7:
        return 1.0065 - 0.7188 * x + 0.6110 * x ** 2 - 0.3044 * x ** 3 + 0.0813 * x ** 4 - 0.0109 * x ** 5 + 0.0006 * x ** 6
    else:
        return 0

# Material Functions Lists (Kt)
def calculate_kt(e, D, M, t, Material_In):
    W = 2 * e
    x = W / D
    Mat = M
    #c1 = 0.8534 + 0.2891 * x - 0.1511 * x ** 2 - 0.0035 * x ** 3 + 0.0174 * x ** 4 - 0.0038 * x ** 5 + 0.0002 * x ** 6
    #c2 = 1.5030 - 1.5054 * x + 1.7453 * x ** 2 - 0.9890 * x ** 3 + 0.2838 * x ** 4 - 0.0390 * x ** 5 + 0.0020 * x ** 6
    #c3 = 0.6270 + 0.9428 * x - 0.8837 * x ** 2 + 0.3900 * x ** 3 - 0.0928 * x ** 4 + 0.0115 * x ** 5 - 0.0005 * x ** 6
    #c4 = 0.9083 + 0.3195 * x - 0.2985 * x ** 2 + 0.0912 * x ** 3 - 0.0121 * x ** 4 + 0.0006 * x ** 5
    #c5 = 0.6115 + 1.4003 * x - 1.6563 * x ** 2 + 0.8517 * x ** 3 - 0.2273 * x ** 4 + 0.0306 * x ** 5 - 0.0016 * x ** 6
    #c6 = 0.7625 + 1.1900 * x - 1.5365 * x ** 2 + 0.7699 * x ** 3 - 0.1987 * x ** 4 + 0.0258 * x ** 5 - 0.0013 * x ** 6
    #c7 = 1.0065 - 0.7188 * x + 0.6110 * x ** 2 - 0.3044 * x ** 3 + 0.0813 * x ** 4 - 0.0109 * x ** 5 + 0.0006 * x ** 6

    if Mat == Material_In[0] or Mat == Material_In[4] or Mat == Material_In[6] or Mat == Material_In[7]:
        kt = calculate_ci(1,x)
    elif (Mat == Material_In[2] or Mat == Material_In[3]) and t <= 1.27:
        kt = calculate_ci(2,x)
    elif Mat == Material_In[1] or Mat == Material_In[5]:
        kt = calculate_ci(2,x)
    elif (Mat == Material_In[2] or Mat == Material_In[3]) and t >= 1.27:
        kt = calculate_ci(4,x)
    elif Mat == Material_In[8] or Mat == Material_In[10]:
        kt = calculate_ci(4,x)
    elif Mat == Material_In[9]:
        kt = calculate_ci(7,x)
    else:
        kt = 0
        pass
    return kt

@jit(nopython=True)
def calculate_kty(w, D, t):
    x = (6 / ((4 / (0.5 * (w - D) + D / (2 * 2 ** 0.5))*t) + 2 / (0.5 * (w - D))*t)) / (D * t)
    curve = -0.0074 + 1.384 * x - 0.5613 * x ** 2 + 1.46159 * x ** 3 - 2.6979 * x ** 4 + 1.912 * x ** 5 - 0.4626 * x ** 6
    return curve

@jit(nopython=True)
```

```
def calculate_vol(t, e, D):
    volume = math.pi * (e** 2 - (D / 2) ** 2) * t
    return volume

@jit(nopython=True)
def calculate_tension_area(t, e, D):
    W = 2 * e
    A_t = t * (W - D)
    return A_t

@jit(nopython=True)
def calculate_bearing_area(t, D):
    A_br = D * t
    return A_br

@jit(nopython=True)
def choose_kby(t, D, e):
    x = e / D
    if t / D > 0.4:
        kby = -1.4512 + 4.006 * x - 2.4375 * (x ** 2) + 1.04689 * (x ** 3) - 0.3279 * (x **
    4) + 0.0612 * (
                        x ** 5) - 0.0047 * (x ** 6)
    if 0.3 < t / D <= 0.4:
        kby = -1.4512 + 4.006 * x - 2.4375 * (x ** 2) + 1.04689 * (x ** 3) - 0.32799 * (x **
    4) + 0.0612 * (
                        x ** 5) - 0.0047 * (x ** 6)
    if 0.2 < t / D <= 0.3:
        kby = -1.0836 + 2.43934 * x + 0.09407 * (x ** 2) - 0.9348 * (x ** 3) + 0.45635 * (x
    ** 4) - 0.0908 * (
                        x ** 5) + 0.00671 * (x ** 6)
    if 0.15 < t / D <= 0.2:
        kby = -1.4092 + 3.62945 * x - 1.3188 * (x ** 2) - 0.219 * (x ** 3) + 0.27892 * (x **
    4) - 0.07 * (
                        x ** 5) + 0.00582 * (x ** 6)
    if 0.12 < t / D <= 0.15:
        kby = -1.7669 + 5.01225 * x - 3.2457 * (x ** 2) + 0.9993 * (x ** 3) - 0.119 * (x **
    4) - 0.0044 * (
                        x ** 5) + 0.00149 * (x ** 6)
    if 0.1 < t / D <= 0.12:
        kby = -3.0275 + 9.93272 * x - 10.321 * (x ** 2) + 5.8327 * (x ** 3) - 1.8303 * (x **
    4) + 0.29884 * (
                        x ** 5) - 0.0197 * (x ** 6)
    if 0.08 < t / D <= 0.1:
        kby = -2.7484 + 8.61564 * x - 8.1903 * (x ** 2) + 4.174 * (x ** 3) - 1.1742 * (x **
    4) + 0.17149 * (
                        x ** 5) - 0.0101 * (x ** 6)
    if 0.06 < t / D <= 0.08:
        kby = -2.4114 + 7.7648 * x - 7.6285 * (x ** 2) + 4.0767 * (x ** 3) - 1.2130 * (x **
    4) + 0.1882 * (
                        x ** 5) - 0.0118 * (x ** 6)
    if 0 < t / D <= 0.06:
        kby = -2.6227 + 8.91273 * x - 0.8543 * (x ** 2) + 5.8749 * (x ** 3) - 1.9336 * (x **
    4) + 0.3295 * (
                        x ** 5) - 0.0226 * (x ** 6)

    return kby
M_S =0 # 1.25 * 1.1 - 1   The safety factors are already applied at the locations where the
    material properties are used (x1.1) and for the forces (x1.25)
def Optimize_Lug(Material_In2,Sigma_In,Density_In,design_object, design_loads, high_accuracy)
    :
    # to be changed

    N_lugs = design_object.N_lugs
    N_Flanges = design_object.N_Flanges
    if high_accuracy == True:
        [i_step, j_step, k_step,l_step, Material_List, tolerance] = [20, 5, 20, 50,
    Material_In2, 0.001]
    else:
        [i_step, j_step, k_step, l_step, Material_List, tolerance] = [40, 10, 40, 100,
    Material_In2,0.01]
    # calculate Dist_between_lugs here:
```

```
    design_object.Dist_between_lugs = 1.4
    # design_object.Dist_between_lugs = calculated_Dist_between_lugs
    if design_object.Dist_between_lugs == 0:
        design_object.Dist_between_lugs = 0.8
        distance = design_object.Dist_between_lugs
    else:
        distance = design_object.Dist_between_lugs
    # FORCES with safety factor of 1.25
    Fx = design_loads.F_x / (N_Flanges * N_lugs)
    Fy = design_loads.F_y / (N_Flanges * N_lugs)
    Fz = design_loads.F_z / (N_Flanges * N_lugs)
    Mx = design_loads.M_x
    My = design_loads.M_y
    Mz = design_loads.M_z  # to be changed
    material_best_configuration_dictionnary = []
    design_array = []
    # Optimisation for each material and compare the options
    # intial guesses for '2014-T6(DF-L)':
    dictionnary = []
    for material in Material_List:
        for i in Material_In2:
            if i == material:
                sigma_y = Sigma_In[Material_In2.index(i)]
        for i in range(10, 500, i_step):
            e = i * 10 ** (-3)
            for j in range(1, 50, j_step):
                t = j * 10 ** (-3)
                print(f"Progress: {round((i/500+j/(50*j_step))*100/1.062,1)}% of Material: {material}", flush=True)
                for k in range(10, 500, k_step):
                    D = k * 10 ** (-3)
                    for l in range(10, 900, l_step):
                        h = l * 10 ** (-3)
                        initial_guess = [e, t, D, h]
                        # e=radius outer flange, t=thickness, D=diameter of the inner circle, material

                        K_t = calculate_kt(initial_guess[0], initial_guess[1], material, initial_guess[2], Material_In2)
                        K_ty = choose_kby(initial_guess[2], initial_guess[1], initial_guess[0])
                        def objective_function(variables, material=material):
                            e, t, D, h= variables
                            volume = calculate_vol(t, e, D)
                            for i in Material_In2:
                                if i == material:
                                    rho = Density_In[Material_In2.index(i)]
                            m = rho * volume
                            return m

                        def volume_constraint(variables):
                            e, t, D, h = variables
                            return calculate_vol(t, e, D)

                        def principal_constraint(variables):
                            e, t, D, h = variables
                            # K_t = calculate_kt(e,D,material,t)
                            # K_ty = choose_kby(t,D,e)
                            A_t = calculate_tension_area(t, e, D)
                            A_br = calculate_bearing_area(t, D)
                            for i in Material_In2:
                                if i == material:
                                    sigma_y = Sigma_In[Material_In2.index(i)]*1.1 #SF for material porperties
                            if N_lugs == 2:
                                force_couple_y = My/(distance*N_Flanges)
                            else:
                                force_couple_y = My/h
                            return ((Fx / (K_t * sigma_y* A_t)) ** 1.6 + ((Fz + force_couple_y)/ (K_ty * A_br * sigma_y)) ** 1.6)**(-0.625) - 1 - M_S
                        def constraint_thickness(variables):
```

```
            e,t,D,h =variables
            return -t + 0.05
        def constraint_thickness_bigger_zero(variables):
            e,t,D,h =variables
            return t-0.0001
        def constraint_outer_radius(variables):
            e,t,D,h=variables
            return -e+0.2
        def constraint_outer_radius_bigger_zero(variables):
            e,t,D,h =variables
            return e-0.0001
        def constraint_inner_diameter(variables):
            e,t,D,h= variables
            return  -D+0.39
        def constraint_inner_diameter_bigger_zero(variables):
            e,t,D,h= variables
            return  D-0.005
        def constraint_dimension(variables):
            e, t, D, h = variables
            return e-D/2 -0.005

        def constraint_inter_flange_distance(variables):
            e, t, D, h = variables
            return h-0.0001

        def constraint_inter_flange_distance_max(variables):
            e, t, D, h = variables
            return -h+1
        if N_Flanges == 2:
            def constraint_zero_h(variables):
                e, t, D, h = variables
                return h
        def moment_x_constraint(variables):
            e,t,D,h = variables
            sigma = (Mx*e)/((t*(2*e)**3)/12)-sigma_y*1.1
            return sigma

        def thickness_over_diameter_lower_limit(variables):
            e,t,D,h = variables
            return - e/t + 10
        if design_object.N_Flanges ==2:
            constraints = [
                {'type': 'ineq', 'fun': volume_constraint},
                {'type': 'eq', 'fun': principal_constraint},
                {'type': 'ineq', 'fun': constraint_thickness},
                {'type': 'ineq', 'fun': constraint_thickness_bigger_zero},
                {'type': 'ineq', 'fun': constraint_outer_radius},
                {'type': 'ineq', 'fun': constraint_outer_radius_bigger_zero},
                {'type': 'ineq', 'fun': constraint_inner_diameter},
                {'type': 'ineq', 'fun': constraint_inner_diameter_bigger_zero
    },
                {'type': 'ineq', 'fun': constraint_dimension},
                {'type': 'ineq', 'fun': constraint_inter_flange_distance},
                {'type': 'ineq', 'fun': constraint_inter_flange_distance_max
    },
                {'type': 'ineq', 'fun': moment_x_constraint},
                {'type': 'ineq', 'fun': thickness_over_diameter_lower_limit},
                {'type': 'ineq', 'fun': constraint_zero_h}
            ]
        else:
            constraints = [
                {'type': 'ineq', 'fun': volume_constraint},
                {'type': 'eq', 'fun': principal_constraint},
                {'type': 'ineq', 'fun': constraint_thickness},
                {'type': 'ineq', 'fun': constraint_thickness_bigger_zero},
                {'type': 'ineq', 'fun': constraint_outer_radius},
                {'type': 'ineq', 'fun': constraint_outer_radius_bigger_zero},
                {'type': 'ineq', 'fun': constraint_inner_diameter},
                {'type': 'ineq', 'fun': constraint_inner_diameter_bigger_zero
    },
                {'type': 'ineq', 'fun': constraint_dimension},
```

```
                                {'type': 'ineq', 'fun': constraint_inter_flange_distance},
                                {'type': 'ineq', 'fun': constraint_inter_flange_distance_max},
                                {'type': 'ineq', 'fun': moment_x_constraint},
                                {'type': 'ineq', 'fun': thickness_over_diameter_lower_limit},
                                #{'type': 'ineq', 'fun': constraint_zero_h}
                            ]

                        # Choose an optimization method
                        method = 'SLSQP'
                        # Call the minimize function (Turning of display makes it WAY faster as printing takes a lot of memory)
                        result = minimize(objective_function, initial_guess, method=method, constraints=constraints,
                                          options={'disp': False}, tol=tolerance)

                        if result.success == True and 0.01 <= result.fun <= 0.9:
                            dictionnary.append([result.x, result.fun])
                        else:
                            pass
            best_configuration = None
            min_mass = float('inf')
            # Iterate through the configurations
            for config in dictionnary:
                dimensions, mass = config
                if mass < min_mass:
                    min_mass = mass
                    best_configuration = config
            material_best_configuration_dictionnary.append((material,best_configuration))

        #Check of the height of the flange limited by the Fy = 433:
        # if stress is exceeding the yield stress = fail
        MMOI = ((2*best_configuration[0][0]) *(best_configuration[0][1])**3)/12
        if MMOI == 0:
            MMOI = 0.0001

        height_flange = best_configuration[0][0] + 2*sigma_y*MMOI/(best_configuration[0][1]*Fy) #np.sqrt((MMOI*sigma_y*1.1)/Fy)
        if height_flange <= 3*best_configuration[0][0]:
            height_flange = 3*best_configuration[0][0]
            #required_MMOI = (Fy * (height_flange - best_configuration[0][0])**2)/(sigma_y*1.1)
            #best_configuration[0][1] = required_MMOI*12/(2*(best_configuration[0][0])**(1/3))
            best_configuration[0][1] = np.sqrt(3*Fx/(sigma_y * 10**6))
        design_array.append(DesignClass.DesignInstance(h=1000*best_configuration[0][3], t1=1000*best_configuration[0][1], t2=design_object.t2, t3=design_object.t3, D1=1000*best_configuration[0][2], \
                                                            w=2*1000*best_configuration[0][0], material=material, n_fast=design_object.n_fast, length=design_object.length, \
                                                            offset=0.5*(design_object.length-1000*best_configuration[0][1]-1000*best_configuration[0][3]*(design_object.N_Flanges-1)),flange_height=1000*height_flange,hole_coordinate_list=design_object.hole_coordinate_list, \
                                                            D2_list=design_object.D2_list, yieldstrength=sigma_y,N_lugs=design_object.N_lugs,N_Flanges=design_object.N_Flanges, Dist_between_lugs=design_object.Dist_between_lugs*1000)) #convert meters to millimeters

    print(material_best_configuration_dictionnary)
    return design_array
```

Listing A.4: *LocalLoadCalculatorAndLugDesignerAndLugConfigurator.py code*

```

# Please add the following: as output return True/False for whether the input design passes the bearingcheck, and if it does not, add an additional output
# that says one of the following: ”x needs to increase”, ”y needs to increase”, ”x needs to decrease”, ”y needs to decrease”, ”Diameter needs to increase”, ”Diameter can decrease”
# as well as the index i that corresponds to the hole that this advice applies to. Only one advice about one hole needs to be returned
# at a time; the function can be run again to get advices for the other holes.
```

```python
import DesignClass as dc
import numpy as np
debug_design_2 = dc.DesignInstance(h=10, t1=0.8862884667966812, t2=2.6499999999999986, t3=2, D1=5, w=19.580136993010818, material="metal", n_fast=4, \
                                            length=47, offset=-0.44314423339834086, flange_height=29.370205489516227, \
                                            hole_coordinate_list=[(4.756555492274806, 4.626310958321483), (4.756555492274806, 119.95382603468913), (42.24344450772523, 4.626310958321483), (42.24344450772523, 119.95382603468913)], \
                                           D2_list=[2.4,2.4,2.4, 2.4], yieldstrength=414, N_lugs=2,N_Flanges=2, Dist_between_lugs=1400, bottomplatewidth=124.58013699301083)
# debug_design_2.minimum_diameter = 3
# debug_design_2.maximum_diameter = 5
# debug_design_2.fastener_rows = 2
# debug_design_2.n_fast = 4
#debug_design_2.l = 6
#hole_coordinate_list = [(-30, -80), (20, 80), (-20, 60), (20, -70)]
#D2_list = [10, 5, 9, 8]
# Fx = 400
# Fz = 1200

debug_loads = dc.Load(433.6,433.6,1300.81,817.34,817.34,0)

def calculate_centroid(design_object):
    holes_area = np.pi * np.array(design_object.D2_list) ** 2 / 4
    weighted_sum_z = np.sum(np.array(design_object.hole_coordinate_list)[:, 1] * holes_area)
    weighted_sum_x = np.sum(np.array(design_object.hole_coordinate_list)[:, 0] * holes_area)
    centroid_x = weighted_sum_x / np.sum(holes_area)
    centroid_z = weighted_sum_z / np.sum(holes_area)

    return (centroid_x,centroid_z)

centroid_x, centroid_z = calculate_centroid(debug_design_2)
def get_in_plane_loads(design_object, load_object):
    f_in_planex = load_object.F_x / len(design_object.D2_list)
    f_in_planez = load_object.F_z / len(design_object.D2_list)
    r_to_cg = np.sqrt((centroid_x - design_object.length/2)**2 + (centroid_z - design_object.bottomplatewidth/2)**2)/1000

    M_y=0
    if centroid_x == design_object.length/2 and centroid_z == design_object.bottomplatewidth/2:
        M_y = 0
    elif centroid_x == design_object.length/2 and centroid_z > design_object.bottomplatewidth/2:
        M_y = - load_object.F_x * r_to_cg
    elif centroid_x == design_object.length/2 and centroid_z < design_object.bottomplatewidth/2:
        M_y =  load_object.F_x * r_to_cg
    elif centroid_x > design_object.length / 2 and centroid_z == design_object.bottomplatewidth / 2:
        M_y = - load_object.F_z * r_to_cg
    elif centroid_x < design_object.length / 2 and centroid_z == design_object.bottomplatewidth / 2:
        M_y =  load_object.F_z * r_to_cg
    elif centroid_x < design_object.length / 2 and centroid_z < design_object.bottomplatewidth / 2:
        theta = np.arctan(np.absolute((centroid_z - design_object.bottomplatewidth / 2) / (centroid_x - design_object.length / 2)))
        M_y =  (load_object.F_z * np.cos(theta) + load_object.F_x * np.sin(theta)) * r_to_cg

    elif centroid_x < design_object.length / 2 and centroid_z > design_object.bottomplatewidth / 2:
        theta = np.arctan(np.absolute((centroid_z - design_object.bottomplatewidth / 2) / (centroid_x - design_object.length / 2)))
        M_y =  (load_object.F_z * np.cos(theta)  - load_object.F_x * np.sin(theta)) * r_to_cg

    elif centroid_x > design_object.length / 2 and centroid_z > design_object.bottomplatewidth / 2:
```

```python
        theta = np.arctan(np.absolute((centroid_z - design_object.bottomplatewidth / 2) / (centroid_x - design_object.length / 2)))
        M_y =  - (load_object.F_z * np.cos(theta)  + load_object.F_x * np.sin(theta)) * r_to_cg

    elif centroid_x > design_object.length / 2 and centroid_z < design_object.bottomplatewidth / 2:
        theta = np.arctan(np.absolute((centroid_z - design_object.bottomplatewidth / 2) / (centroid_x - design_object.length / 2)))
        M_y =  (-load_object.F_z * np.cos(theta)  + load_object.F_x * np.sin(theta)) * r_to_cg

    return f_in_planex , f_in_planez, M_y

def get_F_in_plane_My(design_object, load_object3):
    S = np.sum(((np.array(design_object.hole_coordinate_list)[:, 0] - centroid_x) ** 2 + (np.array(design_object.hole_coordinate_list)[:, 1] - centroid_z) ** 2) *np.pi * np.array(design_object.D2_list) ** 2 / 4) / (1000 ** 2)
    M_y = get_in_plane_loads(design_object, load_object3)[2]

    F_in_plane_My = []

    for i in range(len(design_object.D2_list)):
        distance_x_1 = design_object.hole_coordinate_list[i][0] - centroid_x
        distance_z_1 = design_object.hole_coordinate_list[i][1] - centroid_z
        r = np.sqrt(distance_x_1 ** 2 + distance_z_1 ** 2)/1000
        A = (np.pi * design_object.D2_list[i] ** 2 / 4)
        F_in_plane_My_value = M_y * A * r / S
        F_in_plane_My.append(F_in_plane_My_value)
    return F_in_plane_My





#print(get_in_plane_loads(debug_design_2, debug_loads))
#print(calculate_centroid(debug_design_2))
#print(get_F_in_plane_My(debug_design_2, debug_loads))

def check_bearing_stress(design_object, load_object2,thermal_force):

    M_y = get_in_plane_loads(design_object, load_object2)[2]
    S = np.sum(((np.array(design_object.hole_coordinate_list)[:, 0] - centroid_x) ** 2 + (np.array(design_object.hole_coordinate_list)[:, 1] - centroid_z) ** 2) * np.pi * np.array(design_object.D2_list) ** 2 / 4) / (1000 ** 4)

    sigma0 = []
    for i in range(len(design_object.D2_list)):
        distance_x_1 = design_object.hole_coordinate_list[i][0] - centroid_x
        distance_z_1 = design_object.hole_coordinate_list[i][1] - centroid_z
        r = np.sqrt(distance_x_1 ** 2 + distance_z_1 ** 2)/1000
        A = (np.pi * design_object.D2_list[i] ** 2 / 4)/(1000**2)
        P = np.sqrt(get_in_plane_loads(design_object, load_object2)[0]**2 + get_in_plane_loads(design_object, load_object2)[1]**2 + (M_y * A * r / S)**2 + thermal_force[i]**2)
        sigma = (P / (design_object.D2_list[i] * design_object.t2))
        sigma0.append(sigma)

    if np.max(sigma0) < design_object.yieldstrength:
        return "Bearing Stress Check Pass"
    else:
        return "Bearing Stress Check Failed, increase the thickness of the backplate "

print(check_bearing_stress(debug_design_2, debug_loads,[1533.2328,    383.3082, 1241.918568,  981.268992]))
#
# print(debug_design_2.t2)
# check1 = False
# print(check_bearing_stress(debug_design_2, debug_loads, [0, 0, 0,
#                                                                     0]))
# if not check_bearing_stress(debug_design_2, debug_loads, [0, 0, 0,
```

```
#                                                              0]) == "Bearing Stress Check
     Failed, increase the thickness of the backplate ":
#     check1 = True
# counter1 = 0
# print(check1)
# while check1 == False and counter1 < 50:
#     debug_design_2.t2 += 0.2
#     if not check_bearing_stress(debug_design_2, debug_loads,[0,0,0,0]) == "Bearing Stress
    Check Failed, increase the thickness of the backplate ":
#         check1=True
#     else:
#         print("gonna increase")
#     counter1 += 1
#
# print(debug_design_2.t2, counter1)
#
#
```

Listing A.5: *CheckBearing.py code*

```
# Please add the following: as output return True/False for whether the input design passes
    the pull through check, and if it does not, add an additional output
# that says one of the following: "x needs to increase", "y needs to increase", "x needs to
    decrease", "y needs to decrease", "Diameter needs to increase", "Diameter can decrease"
# as well as the index i that corresponds to the hole that this advice applies to. Only one
    advice about one hole needs to be returned
# at a time; the function can be run again to get advices for the other holes.

# This software component will check the given input design for Pull Through Failure.
import DesignClass
import numpy as np

import InputVariables

debug_design_2 = DesignClass.DesignInstance(h=30, t1=5, t2=0.1, t3=2, D1=10, w=80, material="
    2014-T6(P)", n_fast=4, \
                                            length=10, offset=20,flange_height=80, \
                                            hole_coordinate_list=[(3, 35), (3, 65), (7, 35),
    (7, 65)], \
                                           D2_list=[10.5, 10.5, 10.5, 10.5], yieldstrength
    =83,N_lugs=2,N_Flanges=2, bottomplatewidth=100)
debug_design_1 = DesignClass.DesignInstance(h=30, t1=5, t2=0.2, t3=0.2, D1=10, w=50, material
    ="2014-T6(P)", n_fast=4,
                                            length=100, offset=20, flange_height=60,
    bottomplatewidth=100,
                                            hole_coordinate_list=[(10, 10), (10, 90), (90,
    10), (90, 90)],
                                            D2_list=[5, 5, 5, 5], yieldstrength=83,
    shearstrength=550,N_lugs=1,N_Flanges=2)
debug_loads = DesignClass.Load(433.6,433.6,1300.81,817.34,817.34,0)


def check_pullthrough(design_object, load_object): #checks pullout shear, if smaller than max
     we can decrease thickness,
    index = InputVariables.Material.index(design_object.material)
    design_object.shearstrength = InputVariables.shear_strength[index]
    n_fast = len(design_object.D2_list)
    F_x = load_object.F_x
    F_y = load_object.F_y  # 433.60
    F_z = load_object.F_z
    M_x = load_object.M_x + F_z * 0.001 * (design_object.flange_height - design_object.w / 2)
      # M_x(817.34) plus moment from F_z
    M_z = F_x * 0.001 * (design_object.flange_height - design_object.w / 2)
    Sum_A_rz2 = 0
    Sum_A_rx2 = 0
    F_yi = []
    F_y_Mx = []
    F_y_Mz = []
    F_y_load = F_y / n_fast
    # just direct load F_y:
    for i in range(len(design_object.D2_list)):
```

```
        F_yi.append(F_y_load)
    # contribution from M_x:
    for i in range(len(design_object.D2_list)):
        rz_i = design_object.hole_coordinate_list[i][1] - design_object.bottomplatewidth/2
        Sum_A_rz2 += (np.pi * 0.25 * design_object.D2_list[i] ** 2) * (rz_i ** 2)  # in mm^4
    for i in range(len(design_object.D2_list)):
        rz_i = design_object.hole_coordinate_list[i][1] - design_object.bottomplatewidth/2
        F_y_Mx.append((M_x * 1000 * rz_i * np.pi * 0.25 * design_object.D2_list[i] ** 2) / Sum_A_rz2)
    # contribution from M_z:
    for i in range(len(design_object.D2_list)):
        rx_i = design_object.hole_coordinate_list[i][0] - design_object.length/2
        Sum_A_rx2 += (np.pi * 0.25 * design_object.D2_list[i] ** 2) * (rx_i ** 2)  # in mm^4
    for i in range(len(design_object.D2_list)):
        rx_i = design_object.hole_coordinate_list[i][0] - design_object.length/2
        F_y_Mz.append((M_z * 1000 * rx_i * np.pi * 0.25 * design_object.D2_list[i] ** 2) / Sum_A_rx2)
    # adding all three components:
    for i in range(len(design_object.D2_list)):
        F_yi[i] += F_y_Mx[i] + F_y_Mz[i]

    # checking for failure:
    for i in range(len(design_object.D2_list)):
        Dfi = design_object.D2_list[i]
        Dfo = 1.8 * Dfi
        #print(”Dfo”,Dfo)
        shear = F_yi[i] / (np.pi * (Dfo/1000) * (design_object.t2/1000))
        shear2 = F_yi[i] / (np.pi * (Dfo / 1000) * (design_object.t3 / 1000))
        print(”shears”,shear2, shear)
        sigma_y = F_yi[i]/(np.pi / 4 * ((Dfo/1000)**2-(Dfi/1000)**2))
        shearmax = design_object.shearstrength*10**6 #np.sqrt((design_object.yieldstrength**2 - sigma_y**2)/3)
        print(”shearmax”, shearmax)

        if not abs(shear) < abs(shearmax):
            xmax = 0
            xmin = design_object.length
            for hole in design_object.hole_coordinate_list:
                if hole[0] > xmax:
                    xmax = hole[0]
                if hole[0] < xmin:
                    xmin = hole[0]
            DeltaX = xmax-xmin
            zmax = 0
            zmin = design_object.bottomplatewidth
            for hole in design_object.hole_coordinate_list:
                if hole[1] > zmax:
                    zmax = hole[1]
                if hole[1] < zmin:
                    zmin = hole[1]
            DeltaZ = zmax - zmin
            if abs(design_object.hole_coordinate_list[i][0]-0.5*design_object.length) < 0.5*design_object.h*(design_object.N_Flanges-1) * design_object.t1*design_object.N_Flanges*0.5:
                return [False, ”increase x”, i]
            if (F_x*design_object.flange_height/(M_x+F_z*design_object.flange_height)) < DeltaX/DeltaZ:
                # if design_object.hole_coordinate_list[i][0] < design_object.bottomplatewidth/2:
                #     if  design_object.hole_coordinate_list[i][1] - 2 * design_object.D2_list[i] > 0:
                #         return [False, ”decrease x”, i]
                # if design_object.hole_coordinate_list[i][1] > design_object.bottomplatewidth/2:
                return [False, ”increase z”, i]
            else:
                return [False, ”increase x”, i]
            # elif abs(F_y_Mx[i]) < abs(F_y_Mz[i]):
            #     return [False, ”increase t2”]
            # else:
            #     if design_object.hole_coordinate_list[i][0] > design_object.length / 2:
```

```
            #         return [False, "increase x", i]
            #     if design_object.hole_coordinate_list[i][0] < design_object.length / 2:
            #         if (design_object.hole_coordinate_list[i][0] + design_object.D2_list[i] / 2) < design_object.length/2 - design_object.h/2:
            #             if design_object.hole_coordinate_list[i][0] - 2 * design_object.D2_list[i] > 0:
            #                 return [False, "decrease x", i]
        if not shear2 < shearmax:
            return [False, "increase t3"]
    return [True, "do nothing"]

print(check_pullthrough(debug_design_1, debug_loads))

# diameter head and shank diameter ratio is 1.8
#print(calculate_centroid(debug_design_1),check_pull_through(debug_design_1) )
print("here",check_pullthrough(debug_design_2,debug_loads))
print(debug_design_2.shearstrength)
```

Listing A.6: *CheckPullThrough.py code*

```
# This software component will check the given input design for Thermal Stress Failure.
import numpy as np
import InputVariables
import DesignClass

# assumption is that for the temperature the coefficient remains linear . mention limitation
# aluminum https://ntrs.nasa.gov/api/citations/19720023885/downloads/19720023885.pdf
# 7075-T6 https://www.matweb.com/search/datasheet_print.aspx?matguid=4f19a42be94546b686bbf43f79c51b7d
# 4130 steel https://industeel.arcelormittal.com/fichier/ds-mold-4130/
#8630 steel https://dl.asminternational.org/handbooks/edited-volume/9/chapter/113443/Thermal-Properties-of-Carbon-and-Low-Alloy-Steels
# 2024-T4 https://www.gabrian.com/2024-aluminum-properties/
# 356-T6 Aluminium https://metalitec.zriha.com/eng/raw-materials/a356-t6
# 2024-T3 2024-T3 https://www.gabrian.com/2024-aluminum-properties

#sources for elastic module
# aluminum 2014-T6 https://asm.matweb.com/search/SpecificMaterial.asp?bassnum=MA2014T6
# 7075-T6 https://asm.matweb.com/search/SpecificMaterial.asp?bassnum=ma7075t6
# 4130 steel https://asm.matweb.com/search/SpecificMaterial.asp?bassnum=m4130r
#8630 steel https://www.efunda.com/materials/alloys/alloy_steels/show_alloy.cfm?ID=AISI_8630&show_prop=all&Page_Title=AISI%208630
# 2024-T4 https://asm.matweb.com/search/SpecificMaterial.asp?bassnum=ma2024t4
# 356-T6 Aluminium https://www.matweb.com/search/datasheet_print.aspx?matguid=d524d6bf305c4ce99414cabd1c7ed070
# 2024-T3 https://asm.matweb.com/search/SpecificMaterial.asp?bassnum=ma2024t3

debug_design_2 = DesignClass.DesignInstance(h=30, t1=5, t2=10, t3=2, D1=10, w=40, material="metal", n_fast=4, \
                                            length=80, offset=20,flange_height=80, \
                                            hole_coordinate_list=[(20, 10), (20, 30), (60, 10), (60, 30)], \
                                           D2_list=[6, 6, 6, 6], yieldstrength=83,N_lugs=1, N_Flanges=2)

def thermal_stress_calculation(design_object, upper_temp , lower_temp, ref_temp , phi_list , material_fastener , material_plate):
    temp_diff_upper = upper_temp - ref_temp
    temp_diff_lower = ref_temp - lower_temp
    np_d2_list = np.array(design_object.D2_list)
    np_phi_list = np.array(phi_list)
    for materials in InputVariables.materials_lug:
        if materials.get("material") == material_plate:
            material_wall_coeff = materials.get('thermal_expansion_coefficient')
    for materials in InputVariables.materials_fasteners:
        if materials.get("Material") == material_fastener:
            material_fastener_stiffness = float(materials.get("Youngs Modulus (GPa)"))
            material_fastener_coeff = float(materials.get("Thermal Expansion (10^-6)/K"))
    # units of thermal coefficient should be in micro(10^-6) and elasitic modulus in mega(10^9)
    print(material_fastener_coeff , "fast"),print(material_wall_coeff , "wall")
```

```
    thermal_force_upper = (np_d2_list/1000) ** 2 / 4 * 3 * temp_diff_upper * (
    material_fastener_coeff - material_wall_coeff) * (1-np_phi_list) *
    material_fastener_stiffness * 1000
    thermal_force_lower = (np_d2_list/1000) ** 2 / 4 * 3 * temp_diff_lower * (
    material_fastener_coeff - material_wall_coeff) * (1-np_phi_list) *
    material_fastener_stiffness * 1000 * -1
    if np.min(thermal_force_lower) > 0:
        return thermal_force_lower , "temperature decrease"
    else:
        return thermal_force_upper , "temperature increase"

print("this", thermal_stress_calculation(debug_design_2,150,-90,15,[0.04,0.04,0.04,0.04] ,"
    Titanium (Grade 5)",'2014-T6(DF-L)'))
```

Listing A.7: *CheckThermalStress.py code*

```
# This software component will select the fastener configuration according to WP4.4.
# this checks if given w allows for the number and size of fastners given constraints
import copy

import DesignClass
import math
import numpy as np


debug_design = DesignClass.DesignInstance(h=30, t1=5, t2=10, t3=2, D1=10, w=80, material="
    metal", n_fast=4, \
                                           length=200, offset=20,flange_height=80, \
                                           hole_coordinate_list=[(22, 20), (180, 30), (160,
    20), (25, 25)], \
                                          D2_list=[10, 5, 9, 8], yieldstrength=83,N_lugs=1,
    N_Flanges=2)

#check wether spacing constraints detailed in 4.4 are met, inp
# buts are list of coordinates and list of diameter sizes
def fastener_spacing_check(design_object):
    np_D2_list = np.array(design_object.D2_list)
    np_hole_coordinate_list = np.array(design_object.hole_coordinate_list)
    # these
    if True == True: #design_object.material == "metal": # for now we only consider metals
        lower_limit = 2 * np.max(np_D2_list)
        upper_limit = 3 * np.max(np_D2_list)

    elif design_object.material == "composite":
        lower_limit = 4 * np.max(np_D2_list)
        upper_limit = 5 * np.max(np_D2_list)

    #for loop checks wether the holes are within the margin of 2 * D2 from edges. For every
    hole
    for i in range(len(np_D2_list)):
        if np_hole_coordinate_list[i,0] <= 2 * np_D2_list[i] or design_object.length -
    np_hole_coordinate_list[i,0] <= 2 * np_D2_list[i]:
            #print("LengthMarginErr")
            if np_hole_coordinate_list[i,0] - 2 * np_D2_list[i] <= 0:
                #print(np_hole_coordinate_list[i,0], np_D2_list[i])
                #print("LengthMarginErr1")
                return False , i , np_hole_coordinate_list[i,0] - 2 * np_D2_list[i], "
    LengthMarginError"
            elif design_object.length - np_hole_coordinate_list[i,0] - 2 * np_D2_list[i] <=0:
                #print("LengthMarginErr2")
                return False , i , design_object.length - np_hole_coordinate_list[i,0]- 2 *
    np_D2_list[i] , "LengthMarginError"
        if np_hole_coordinate_list[i,1]  < 2 * np_D2_list[i] or design_object.
    bottomplatewidth - np_hole_coordinate_list[i,1] < 2 * np_D2_list[i]:
            if np_hole_coordinate_list[i,1] - 2 * np_D2_list[i] <= 0:
                return False , i , np_hole_coordinate_list[i,1] - 2 * np_D2_list[i] , "
    WidthMarginError"
            elif design_object.bottomplatewidth - np_hole_coordinate_list[i,1] - 2 *
    np_D2_list[i] <=0:
                return False , i , design_object.bottomplatewidth - np_hole_coordinate_list[i
    ,1] - 2 * np_D2_list[i] , "WidthMarginError"
```

```python
    #lower limit and upper limit are calculated based on the material and the for loop checks wether the center to
    #center constraint complies...
    for i in range(len(np_D2_list)):
        for k in range(i + 1, len(np_D2_list)):
            distance_x = np_hole_coordinate_list[i][0] - np_hole_coordinate_list[k][0]
            distance_y = np_hole_coordinate_list[i][1] - np_hole_coordinate_list[k][1]
            distance = math.sqrt(distance_x ** 2 + distance_y ** 2)
            if i != k and not distance >= lower_limit:
                # returns the indexs of the hole_list that are causing an issue and wether the distance betweeen them
                # should be increased or decreased
                return False, i, k, "HoleDistanceTooSmall"
            # elif i != k and not distance <= upper_limit:
            #     return False, i , k , "HoleDistanceTooLarge"
    return [True]


def Optimize_holes(design_object, recursive):
    new_object = copy.deepcopy(design_object)
    if fastener_spacing_check(new_object)[0] == True:
        return new_object
    if fastener_spacing_check(new_object)[3] == "LengthMarginError" and new_object.length < 500:
        #print("LengthMarginError")
        new_object.length += 1
        for i in range(len(new_object.hole_coordinate_list)):
            new_object.hole_coordinate_list[i] = (new_object.hole_coordinate_list[i][0]+0.5, new_object.hole_coordinate_list[i][1])
        #print(new_object.length)
        return Optimize_holes(new_object, True)
    elif fastener_spacing_check(new_object)[3] == "WidthMarginError" and new_object.bottomplatewidth < 400:
        #print("WidthMarginError")
        new_object.bottomplatewidth += 1
        for i in range(len(new_object.hole_coordinate_list)):
            new_object.hole_coordinate_list[i] = (new_object.hole_coordinate_list[i][0], new_object.hole_coordinate_list[i][1]+0.5)
        #print(new_object.bottomplatewidth, new_object.hole_coordinate_list[0])
        #print(fastener_spacing_check(new_object))
        return Optimize_holes(new_object, True)
    elif fastener_spacing_check(new_object)[3] == "HoleDistanceTooSmall":
        index1 = fastener_spacing_check(new_object)[1]
        index2 = fastener_spacing_check(new_object)[2]
        if abs(new_object.hole_coordinate_list[index1][0] - new_object.hole_coordinate_list[index2][0]) >= abs(new_object.hole_coordinate_list[index1][1] - new_object.hole_coordinate_list[index2][1]):
            if new_object.hole_coordinate_list[index1][0] - new_object.hole_coordinate_list[index2][0] <= 0:
                new_object.hole_coordinate_list[index1] = (new_object.hole_coordinate_list[index1][0]-1, new_object.hole_coordinate_list[index1][1])
                new_object.hole_coordinate_list[index2] = (
                new_object.hole_coordinate_list[index2][0] + 1,
                new_object.hole_coordinate_list[index2][1])
            if new_object.hole_coordinate_list[index1][0] - new_object.hole_coordinate_list[index2][0] >= 0:
                new_object.hole_coordinate_list[index1] = (new_object.hole_coordinate_list[index1][0]+1, new_object.hole_coordinate_list[index1][1])
                new_object.hole_coordinate_list[index2] = (
                new_object.hole_coordinate_list[index2][0] - 1,
                new_object.hole_coordinate_list[index2][1])
        else:
            if new_object.hole_coordinate_list[index1][1] - new_object.hole_coordinate_list[index2][1] <= 0:
                new_object.hole_coordinate_list[index1] = (new_object.hole_coordinate_list[index1][0], new_object.hole_coordinate_list[index1][1]-1)
                new_object.hole_coordinate_list[index2] = (
                new_object.hole_coordinate_list[index2][0],
                new_object.hole_coordinate_list[index2][1]+1)
            if new_object.hole_coordinate_list[index1][1] - new_object.hole_coordinate_list[index2][1] >= 0:
```

```
                new_object.hole_coordinate_list[index1] = (new_object.hole_coordinate_list[
    index1][0], new_object.hole_coordinate_list[index1][1]+1)
                new_object.hole_coordinate_list[index2] = (
                new_object.hole_coordinate_list[index2][0],
                new_object.hole_coordinate_list[index2][1]-1)
        return Optimize_holes(new_object, True)
    else:
        #print(new_object.length, new_object.bottomplatewidth, "dit")
        return new_object
```

Listing A.8: *SelectFastenerConfiguration.py code*

```
# This software component will select the fastener based on WP4.10.
import copy

import numpy as np
import DesignClass
debug_design = DesignClass.DesignInstance(h=30, t1=5, t2=2, t3=4, D1=10, w=80, material="
    metal", n_fast=4,length=200, offset=20, flange_height=80, hole_coordinate_list=[(20, 10),
     (180, 60), (160, 20), (30, 60)], D2_list=[9, 4, 6, 8], yieldstrength=83, N_lugs=1,
    N_Flanges=1)
import InputVariables
import CheckThermalStress
import CheckBearing
import CheckPullThrough



def get_youngs_modulus_lug(material_name):
    for material in InputVariables.materials_lug:
        if material["material"] == material_name:
            if isinstance(material["elastic module"], tuple):
                # If the Young's Modulus is given as a range, you can return the average
                return sum(material["elastic module"]) / len(material["Youngs Modulus (GPa)"
    ])
            else:
                return material["elastic module"]
    # If the material name is not found
    return None




# debug_design.Ea = get_youngs_modulus("Aluminium 7075") * 10 ** 9
# debug_design.L_h_sub_type = "Hexagonal"
# debug_design.L_eng_sub_type = "Nut-Tightened"
# debug_design.Eb = get_youngs_modulus(selected_material_fastener) * 10 ** 9
# debug_design.En = get_youngs_modulus(selected_material_fastener) * 10 ** 9
# debug_design.L = [1, 2,2, 1]  # shank length
# debug_design.D = [10, 5, 9, 8]  # shank diamete

### Im gonna try to redo a bit of the code.
fastener_debug = DesignClass.FastenerType("Titanium (Grade 5)","Hexagonal","Nut-Tightened")

def get_fastener_dimensions(FastenerType,DesignInstance):
    np_D2list = np.array(DesignInstance.D2_list)
    if FastenerType.nut_type == "Hexagonal":
        height_head = (np_D2list * 0.5)
    elif FastenerType.nut_type == "Cylindrical":
        height_head = (np_D2list * 0.4)

    if FastenerType.hole_type == "Threaded hole":
        engaged_shank_length = (np_D2list * 0.33)
    elif FastenerType.hole_type == "Nut-Tightened":
        engaged_shank_length = (np_D2list * 0.4)
    #this could be the height of the nut or of the threaded insert thus the general name
    locking device height
    locking_device_height = (np_D2list * 0.4)
    return [height_head , engaged_shank_length , locking_device_height]

def calculate_delta_a(DesignInstance, plate_material , wall_material):
```

```
    Df_I = np.array(DesignInstance.D2_list)
    Df_O = 1.8 * Df_I
    thickness = [DesignInstance.t2,DesignInstance.t3]
    E = [get_youngs_modulus_lug(plate_material), get_youngs_modulus_lug(wall_material)]
    delta_a = []
    for i in range(2):
        delta_a_new = (4 * thickness[i]) / (E[i] * 10 ** 9 * np.pi * (Df_O ** 2 - Df_I ** 2))
        delta_a.append(delta_a_new)

    ”””delta_a_max = []

    for i in range(len(Df_O)):
        if delta_a[0][i] > delta_a[1][i]:
            delta_a_max.append(delta_a[0][i])
        else:
            delta_a_max.append(delta_a[1][i])”””

    return delta_a

print(calculate_delta_a(debug_design, ”7075-T6(DF-LT)”,”7075-T6(DF-LT)”))

def calculate_deta_b(FastenerType,DesignInstance):
    np_D2_list = np.array(DesignInstance.D2_list)
    nominal_area = (1.8 * np_D2_list) ** 2 / 4 # maximal area of the fastener (bolt and head area --> assumption)
    shank_area = (np_D2_list - 1) ** 2 / 4 # minimum area of fastener
    head_height = get_fastener_dimensions(FastenerType,DesignInstance)[0]
    engaged_shank_length = get_fastener_dimensions(FastenerType, DesignInstance)[1]
    bolt_height = get_fastener_dimensions(FastenerType, DesignInstance)[2]
    shank_length = DesignInstance.t2 + DesignInstance.t3
    E_b = FastenerType.youngs_modulus * 10 ** 9
    delta_b = (1/E_b)*((head_height/nominal_area)+((shank_length + engaged_shank_length)/shank_area) + bolt_height/nominal_area)
    return delta_b


print(calculate_deta_b(fastener_debug,debug_design))


def calculate_force_ratio(FastenerType, DesignInstance , plate_material , wall_material):
    force_ratio = []
    delta_a = calculate_delta_a(DesignInstance,plate_material,wall_material)
    delta_b = calculate_deta_b(FastenerType, DesignInstance)
    for i in range(2):
        force_ratio_element = list(delta_a[i]/(delta_a[i]+delta_b))
        force_ratio.append(force_ratio_element)
    #check which compliance is limiting
    if force_ratio[0] > force_ratio[1]:
        return force_ratio[1] , ”vehicle wall/fastener compliance is limiting”
    else:
        return force_ratio[0] , ”back plate/fastener compliance is limiting”

    return force_ratio[0]

print(calculate_force_ratio(fastener_debug,debug_design,”7075-T6(DF-LT)”,”7075-T6(DF-LT)”)[0])

def print_material_info(material_name):
    for material in InputVariables.materials_fasteners:
        if material[”Material”] == material_name:
            print(f”Material: {material['Material']}”)
            print(f”Young's Modulus (GPa): {material['Youngs Modulus (GPa)']}”)
            print(f”Density (kg/m^3): {material['Density (kg/m^3)']}”)
            print(f”Thermal Expansion (10^-6)/K: {material['Thermal Expansion (10^-6)/K']}”)
            print(f”Ultimate Tensile Strength (MPa): {material['Ultimate Tensile Strength (MPa)']}”)
            print(f”Elastic Limit(Mpa): {material['Elastic Limit(Mpa)']}”)
            print(f”Resistance Factors: {material['Resistance Factors']}”)
            return
    # If the material name is not found
    print(f”Material '{material_name}' not found.”)
```

```
print_material_info(material_name="Titanium (Grade 5)")

def check_size_reduction_possibility(design_object, i, design_loads):
    design_object2 = copy.deepcopy(design_object)
    loads2 = copy.deepcopy(design_loads)
    loads2.F_y = 3.97 * design_loads.F_y
    if design_object2.D2_list[i] == 10.5: #https://amesweb.info/screws/Metric-Clearance-Hole-Chart.aspx
        design_object2.D2_list[i] = 8.4  #https://fractory.com/metric-bolt-clearance-hole-size-chart/
        typer = "M8"
    elif design_object2.D2_list[i] == 8.4:
        design_object2.D2_list[i] = 6.4
        typer = "M6"
    elif design_object2.D2_list[i] == 6.4:
        design_object2.D2_list[i] = 5.3
        typer = "M5"
    elif design_object2.D2_list[i] == 5.3:
        design_object2.D2_list[i] = 4.3
        typer = "M4"
    elif design_object2.D2_list[i] ==4.3:
        design_object2.D2_list[i] = 3.2
        typer = "M3"
    elif design_object2.D2_list[i] == 3.2:
        design_object2.D2_list[i] = 2.2
        typer = "M2"
    elif design_object2.D2_list[i] == 2.2:
        design_object2.D2_list[i] = 1.7
        typer = "M1.6"
    elif design_object2.D2_list[i] == 1.7:
        design_object2.D2_list[i] = 1.5
        typer = "M1.4"
    elif design_object2.D2_list[i] == 1.5:
        design_object2.D2_list[i] = 1.3
        typer = "M1.2"
    elif design_object2.D2_list[i] == 1.3:
        design_object2.D2_list[i] = 1.1
        typer = "M1"
    else:
        return
    design_object2.fasteners = DesignClass.FastenerType("Titanium (Grade 5)","Hexagonal","Nut-Tightened")
    design_object2.fasteners.type_bolt = typer
    philist = calculate_force_ratio(design_object2.fasteners, design_object2, design_object2.material, "7075-T6(DF-LT)")[0]
    thermal_loads = CheckThermalStress.thermal_stress_calculation(design_object2, 150, -90, 15, philist
                                                                  , material_fastener=design_object2.fasteners.material,
                                                                  material_plate=design_object2.material)[0]
    check1a = False
    if CheckBearing.check_bearing_stress(design_object2, loads2,
                                         thermal_loads) == "Bearing Stress Check Pass":
        print("reduce hole acc to check1a, this one SHOULD be limiting")
        check1a = True
    check2a = False
    if CheckPullThrough.check_pullthrough(design_object2, loads2)[0] == True:
        print("reduce hole acc to check2a")
        check2a = True
    if check2a == True and check1a == True:
        print("decrease hole size")
        design_object.D2_list[i] = design_object2.D2_list[i]
        design_object.fasteners = design_object2.fasteners
        check_size_reduction_possibility(design_object,i,design_loads)
```

Listing A.9: *SelectFastener.py code*

```
# This software component will return the results and provide a visualization in a graph. (maybe use the Inkscape
# package to make a 3-view)
```

```
import cadquery as cq
import DesignClass
#import cadquery.cqgi as cqgi


from stl import mesh
from mpl_toolkits import mplot3d
from matplotlib import pyplot
import copy

def Visualize(design_object, index, move_y=0):
    design_object.hole_coordinate_list2=copy.deepcopy(design_object.hole_coordinate_list)
    for i in range(len(design_object.hole_coordinate_list2)):
        a = design_object.hole_coordinate_list2[i][0]
        design_object.hole_coordinate_list2[i] = (
        design_object.hole_coordinate_list2[i][1] - design_object.bottomplatewidth / 2, a -
    design_object.length / 2)
        # make the base
    result = (
        cq.Workplane("XY")
        .box(design_object.bottomplatewidth, design_object.length, design_object.t2).faces(">
    Z").workplane().tag("noholes")
    )

    for i in range(len(design_object.D2_list)):
        result = result.workplaneFromTagged("noholes").pushPoints(design_object.
    hole_coordinate_list2[i:i + 1]).hole(
            design_object.D2_list[i])

    if design_object.w <= design_object.flange_height:
        filletrad = design_object.w / 2
    else:
        filletrad = design_object.flange_height / 2
    print(filletrad)
    result = result.faces("<Y").workplane(
        offset=-design_object.offset
    )  # workplane is offset from the object surface
    result = result.union(
        cq.Workplane("XZ").box(design_object.w, design_object.flange_height + filletrad,
    design_object.t1,
                               centered=[True, False, True]).edges(
            "|Y").fillet(filletrad - 0.01).translate((0, 0, +design_object.t2 / 2 - filletrad
    )).center(0,

              -filletrad + design_object.t2 / 2).rect(
            design_object.w, 2 * filletrad).cutThruAll().center(0, design_object.
    flange_height).circle(
            design_object.D1/2).cutThruAll().translate(
            (0, (design_object.t1/2+design_object.h/2)*(design_object.N_Flanges-1), 0)))
    if design_object.N_Flanges == 2:
        result = result.union(
            cq.Workplane("XZ").box(design_object.w, design_object.flange_height + filletrad,
    design_object.t1,
                                   centered=[True, False, True]).edges(
                "|Y").fillet(filletrad - 0.01).translate((0, 0, +design_object.t2 / 2 -
    filletrad)).center(0,

                  -filletrad + design_object.t2 / 2).rect(
                design_object.w, 2 * filletrad).cutThruAll().center(0, design_object.
    flange_height).circle(
                design_object.D1/2).cutThruAll().translate(
                (0, -design_object.t1/2-design_object.h/2, 0)))
    if design_object.N_lugs == 2:
        result = result.union(result.translate((0,move_y, 0)))
    design_object.volume = result.val().Volume()
    # Export
    if design_object.N_lugs == 2 and design_object.N_Flanges == 2:
        cq.exporters.export(result, f"result{index}.stl")
        cq.exporters.export(result.section(), f"result{index}.dxf")
        cq.exporters.export(result, f"result{index}.step")
    elif design_object.N_lugs == 2 and design_object.N_Flanges == 1:
```

```
        cq.exporters.export(result, f”result{index}with2lugs1flange.stl”)
        cq.exporters.export(result.section(), f”result{index}with2lugs1flange.dxf”)
        cq.exporters.export(result, f”result{index}with2lugs1flange.step”)
    elif design_object.N_lugs == 1 and design_object.N_Flanges == 2:
        cq.exporters.export(result, f”result{index}with1lug2flanges.stl”)
        cq.exporters.export(result.section(), f”result{index}with1lug2flanges.dxf”)
        cq.exporters.export(result, f”result{index}with1lug2flanges.step”)
    elif design_object.N_lugs == 1 and design_object.N_Flanges == 1:
        cq.exporters.export(result, f”result{index}with1lug1flange.stl”)
        cq.exporters.export(result.section(), f”result{index}with1lug1flange.dxf”)
        cq.exporters.export(result, f”result{index}with1lug1flange.step”)

    # Create a new plot
    figure = pyplot.figure()
    axes = figure.add_subplot(131, projection='3d')
    axes.view_init(elev=0, azim=0, roll=0)
    axes2 = figure.add_subplot(132, projection='3d')
    axes2.view_init(elev=90, azim=0, roll=0)
    pyplot.title(”A 3D STL file is generated in the main directory which \n can be viewed in other software”)
    axes3 = figure.add_subplot(133, projection='3d')
    axes3.view_init(elev=0, azim=90, roll=0)

    # Load the STL files and add the vectors to the plot
    your_mesh = mesh.Mesh.from_file('result.stl')
    axes.add_collection3d(mplot3d.art3d.Poly3DCollection(your_mesh.vectors))
    axes2.add_collection3d(mplot3d.art3d.Poly3DCollection(your_mesh.vectors))
    axes3.add_collection3d(mplot3d.art3d.Poly3DCollection(your_mesh.vectors))

    # Auto scale to the mesh size
    scale = your_mesh.points.flatten()
    axes.auto_scale_xyz(scale, scale, scale)
    axes2.auto_scale_xyz(scale, scale, scale)
    axes3.auto_scale_xyz(scale, scale, scale)
    # Show the plot to the screen
    pyplot.show()

def Visualize2(design_object, index):
    #Visualize(design_object)
    if design_object.N_lugs == 2:
        Visualize(design_object,index=index, move_y=design_object.Dist_between_lugs)
    else:
        Visualize(design_object, index=index ,move_y=0)

# debug_design4 = DesignClass.DesignInstance(h=30, t1=5, t2=10, t3=2, D1=10, w=80, material=”metal”, n_fast=4, \
#                                           length=200, offset=20,flange_height=80, \
#                                           hole_coordinate_list=[(20, 10), (180, 30), (160, 20), (30, 30)], \
#                                          D2_list=[10, 5, 9, 8], yieldstrength=83,N_lugs=2,N_Flanges=1)
# debug_design4.Dist_between_lugs = 300
# Visualize2(debug_design4)

# #debug_design4 = DesignClass.DesignInstance(h=30, t1=5, t2=0.1, t3=2, D1=10, w=40, material=”metal”, n_fast=4, \
#                                           length=80, offset=20,flange_height=80, \
#                                           hole_coordinate_list=[(21, 20.5), (81, 20.5), (41, 20.5), (61, 20.5)], \
#                                          D2_list=[6, 6, 6, 6], yieldstrength=83,N_lugs=2,N_Flanges=2, Dist_between_lugs=300)
# #Visualize2(debug_design4)
```

Listing A.10: *PostProcessorAndVisualizer.py code*

```
# This software component will do the whole analysis for two different materials, and then compare them.
def TradeOff(design_array2):
    for design in design_array2:
        print(f”{design.material}”)
        print(f”checklist: {design.checklist}, h: {design.h}, t1: {design.t1}, t2: {design.t2}, t3: {design.t3}, D1: {design.D1}, w: {design.w}, length: {design.length}, offset: {
```

```
    design.offset}, flange height: {design.flange_height}, yieldstrength: {design.
    yieldstrength}, material: {design.material}, Dist_between_lugs: {design.Dist_between_lugs
    }, N_lugs: {design.N_lugs}, N_Flanges: {design.N_Flanges}, hole coords: {design.
    hole_coordinate_list}, n_fast: {design.n_fast}, D2holes: {design.D2_list},
    bottomplatewidth: {design.bottomplatewidth}, shearstrength: {design.shearstrength}")
        print(f"nut type: {design.fasteners.nut_type}, hole type: {design.fasteners.hole_type
    }, material: {design.fasteners.material}, bolt type: {design.fasteners.type_bolt}")
        print(f"mass: {design.mass}")
        print(f"MS: {design.MS}")
        print("")

    # add comparison/trade-off here
```

Listing A.11: *TradeOffComperator.py code*

```
# Use the official Python image from the Docker Hub
FROM python:3.9

# Set the working directory in the container to /app
WORKDIR /app

# Copy all contents from the current directory to /app in the container
COPY . /app

# Install any dependencies your Python application needs
RUN pip install -r requirements.txt  # If you have a requirements.txt file

# Set the entry point to your main Python file
CMD ["python", "MainOptimizer.py"]
```

Listing A.12: *Dockerfile code*

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class
*.pyc
*dxf
*.step
*.stl

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
share/python-wheels/
*.egg-info/
.installed.cfg
*.egg
MANIFEST

# PyInstaller
#  Usually these files are written by a python script from a template
#  before PyInstaller builds the exe, so as to inject date/other infos into it.
*.manifest
*.spec

# Installer logs
```

```
pip-log.txt
pip-delete-this-directory.txt

# Unit test / coverage reports
htmlcov/
.tox/
.nox/
.coverage
.coverage.*
.cache
nosetests.xml
coverage.xml
*.cover
*.py,cover
.hypothesis/
.pytest_cache/
cover/

# Translations
*.mo
*.pot

# Django stuff:
*.log
local_settings.py
db.sqlite3
db.sqlite3-journal

# Flask stuff:
instance/
.webassets-cache

# Scrapy stuff:
.scrapy

# Sphinx documentation
docs/_build/

# PyBuilder
.pybuilder/
target/

# Jupyter Notebook
.ipynb_checkpoints

# IPython
profile_default/
ipython_config.py

# pyenv
#   For a library or package, you might want to ignore these files since the code is
#   intended to run in multiple environments; otherwise, check them in:
# .python-version

# pipenv
#   According to pypa/pipenv#598, it is recommended to include Pipfile.lock in version
    control.
#   However, in case of collaboration, if having platform-specific dependencies or
    dependencies
#   having no cross-platform support, pipenv may install dependencies that don't work, or not
#   install all needed dependencies.
#Pipfile.lock

# poetry
#   Similar to Pipfile.lock, it is generally recommended to include poetry.lock in version
    control.
#   This is especially recommended for binary packages to ensure reproducibility, and is more
#   commonly ignored for libraries.
#   https://python-poetry.org/docs/basic-usage/#commit-your-poetrylock-file-to-version-
    control
#poetry.lock
```

```

# pdm
#   Similar to Pipfile.lock, it is generally recommended to include pdm.lock in version
    control.
#pdm.lock
#   pdm stores project-wide configurations in .pdm.toml, but it is recommended to not include
     it
#   in version control.
#   https://pdm.fming.dev/#use-with-ide
.pdm.toml

# PEP 582; used by e.g. github.com/David-OConnor/pyflow and github.com/pdm-project/pdm
__pypackages__/

# Celery stuff
celerybeat-schedule
celerybeat.pid

# SageMath parsed files
*.sage.py

# Environments
.env
.venv
env/
venv/
ENV/
env.bak/
venv.bak/

# Spyder project settings
.spyderproject
.spyproject

# Rope project settings
.ropeproject

# mkdocs documentation
/site

# mypy
.mypy_cache/
.dmypy.json
dmypy.json

# Pyre type checker
.pyre/

# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

# PyCharm
#  JetBrains specific template is maintained in a separate JetBrains.gitignore that can
#  be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
#  and can be added to the global gitignore or merged into this file.  For a more nuclear
#  option (not recommended) you can uncomment the following to ignore the entire idea folder.
#.idea/
```

Listing A.13: *Python.gitignore code*

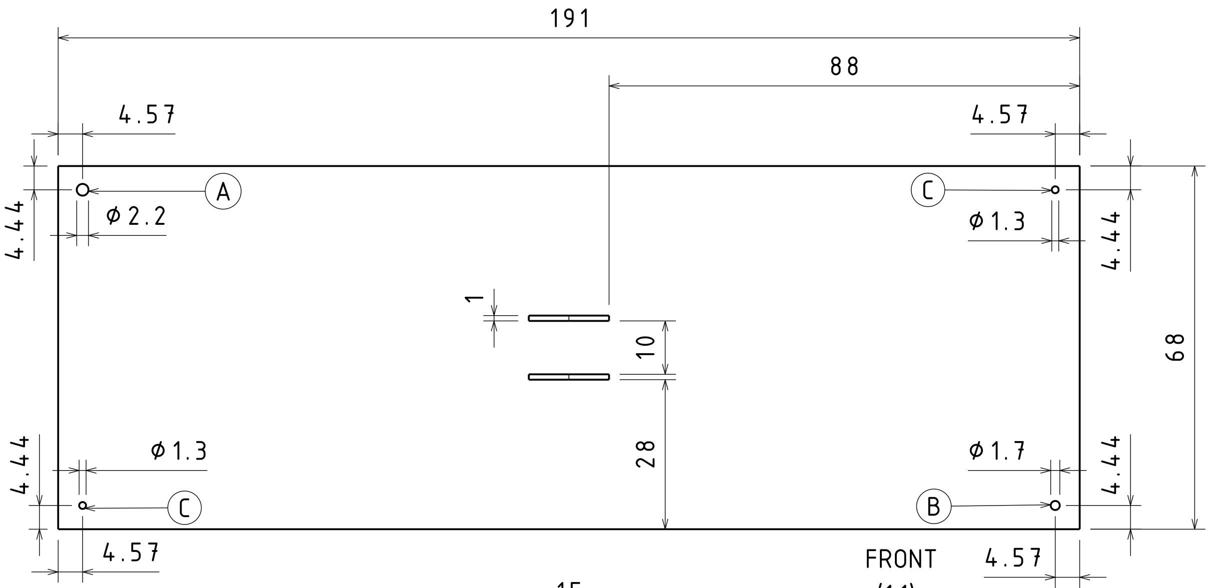
```
numpy==1.24.3
matplotlib==3.7.1
scipy==1.10.1
cadquery==2.3.1
numba == 0.58.1
numpy-stl == 3.1.1
```

Listing A.14: *Requirements.txt code*

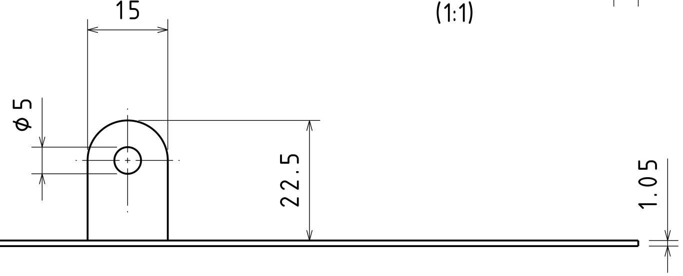
B
CAD Drawings

The CAD Drawing can be found on the next page.

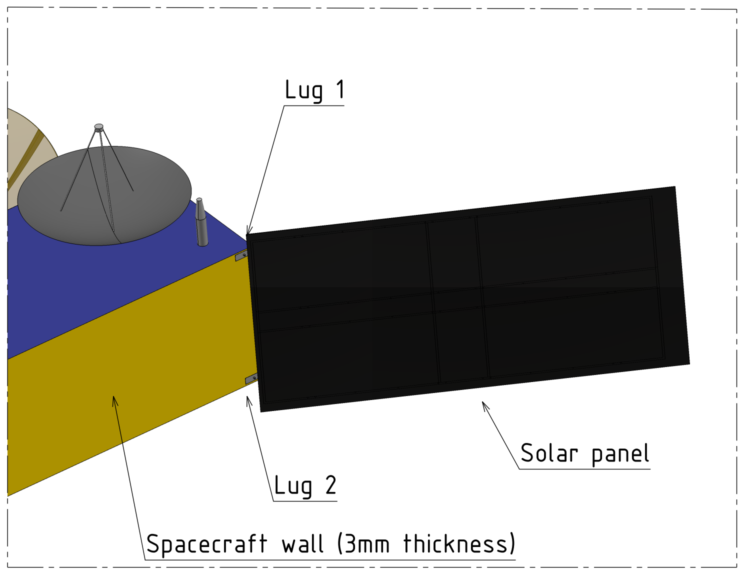
This page was intentionally left blank.



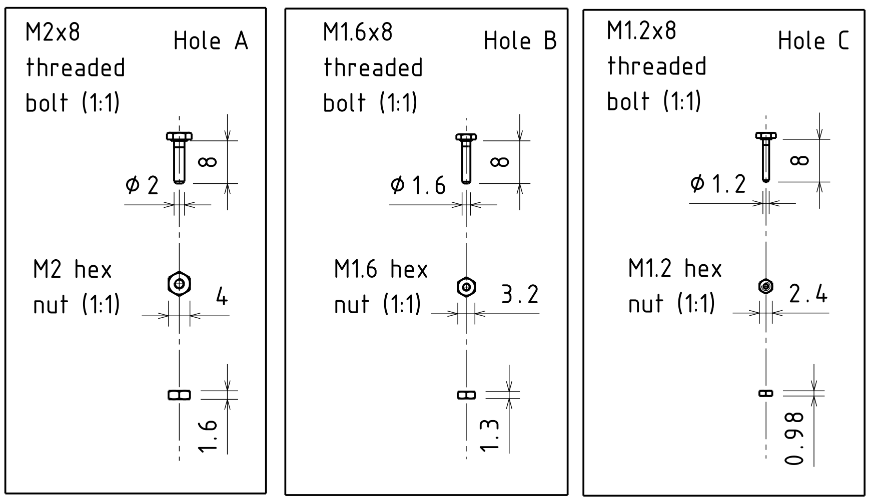
FRONT
(1:1)



BOTTOM
(1:1)



Spacecraft
implementration (1:50)



DESIGNED BY: Miguel Murcia DATE: 05/12/2023 CHECKED BY: Group D03 DATE: 5/12/2023	AE2111-I-D03 Lug design	
SIZE A3	TU Delft AE2111-1 WP4	
SCALE 1:1	WEIGHT (kg) 0.0426	DRAWING NUMBER XXX · SHEET 1/1

C
Task Distribution

Table C.1: *Task Distribution*

Chapter	**Task**	**Who**
Load Analysis	Launch Load	V. Fossa, G.Ruiz, A.Bilbao
	Operational Load	M.Murcia, A.Bilbao
Lug Configuration	Selection of lug configuration	A.Orban, L. Huirne, M.Murcia
	Design of the flange	V. Fossa, A. Bilbao, A.Orban, L. Huirne
	Optimization of the flange design	A.Orban
Fastener Configuration	Select fastener pattern	G.Ruiz
	Bearing check	J. Qi , M. Özözgür
	Pull through check	V. Fossa, G.Ruiz, M.Murcia
	Selection of fastener	J. Qi, M. Özözgür, L. Huirne
Thermal Stress and Material Selection	Thermal stress check	V. Fossa, G.Ruiz
	Fastener material selection	M. Özözgür
	Lug material selection	A.Bilbao, A.Orban
	Final Design	A.Bilbao
Post Processing and Visualisation	Post Processing and Visualisation	L. Huirne
CAD Drawing	CAD Drawing	M.Murcia
Miscellaneous	Introduction	V. Fossa
	Conclusion	G.Ruiz
	Summary and preface	V. Fossa
	Nomenclature	G.Ruiz
	Main Optimizer	L. Huirne
	Set-up of Code repository	L. Huirne