FOSSEE Winter Internship Report

On

Development of CAD generation module for Butt Joint Bolted and Welded Connection for Osdag

Submitted by

Sachin Saud

4th Year B.E. Student, Department of Mechanical and Aerospace Engineering

Institute of Engineering, Tribhuvan University

Kathmandu, Nepal

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Ajmal Babu M S

Parth Karia

Ajinkya Dahale

July 14, 2025

Acknowledgments

- Start with a general statement of thanks. Express your overall gratitude to everyone who supported you during your project or research.
- Project staff at the Osdag team, Ajmal Babu M. S., Ajinkya Dahale, and Parth Karia,
- Osdag Principal Investigator (PI) Prof. Siddhartha Ghosh, Department of Civil Engineering at IIT Bombay
- FOSSEE PI Prof. Kannan M. Moudgalya, FOSSEE Project Investigator, Department of Chemical Engineering, IIT Bombay
- FOSSEE Managers Usha Viswanathan and Vineeta Parmar and their entire team
- Acknowledge the support from the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India, for their role in facilitating this project
- Acknowledge your colleagues who worked with you during your internship or project.
- If appropriate, thank your college, department, head, and principal for their support during your studies.

Contents

Chapter 1

Introduction

1.1 National Mission in Education through ICT

The National Mission on Education through ICT (NMEICT) is a scheme under the Department of Higher Education, Ministry of Education, Government of India. It aims to leverage the potential of ICT to enhance teaching and learning in Higher Education Institutions in an anytime-anywhere mode.

The mission aligns with the three cardinal principles of the Education Policy—**access, equity, and quality**—by:

- Providing connectivity and affordable access devices for learners and institutions.
- Generating high-quality e-content free of cost.

NMEICT seeks to bridge the digital divide by empowering learners and teachers in urban and rural areas, fostering inclusivity in the knowledge economy. Key focus areas include:

- Development of e-learning pedagogies and virtual laboratories.
- Online testing, certification, and mentorship through accessible platforms like EduSAT and DTH.
- Training and empowering teachers to adopt ICT-based teaching methods.

For further details, visit the official website: www.nmeict.ac.in.

1.1.1 ICT Initiatives of MoE

The Ministry of Education (MoE) has launched several ICT initiatives aimed at students, researchers, and institutions. The table below summarizes the key details:

No.	Resource	For Students/Researchers	For Institutions
Audio-Video e-content			
1	SWAYAM	Earn credit via online courses	Develop and host courses; accept credits
2	SWAYAMPRABHA	Access 24x7 TV programs	Enable SWAYAMPRABHA viewing facilities
Digital Content Access			
3	National Digital Library	Access e-content in multiple disciplines	List e-content; form NDL Clubs
4	e-PG Pathshala	Access free books and e-content	Host e-books
5	Shodhganga	Access Indian research theses	List institutional theses
6	e-ShodhSindhu	Access full-text e-resources	Access e-resources for institutions
Hands-on Learning			
7	e-Yantra	Hands-on embedded systems training	Create e-Yantra labs with IIT Bombay
8	FOSSEE	Volunteer for open-source software	Run labs with open-source software
9	Spoken Tutorial	Learn IT skills via tutorials	Provide self-learning IT content
10	Virtual Labs	Perform online experiments	Develop curriculum-based experiments
E-Governance			
11	SAMARTH ERP	Manage student lifecycle digitally	Enable institutional e-governance
Tracking and Research Tools			
12	VIDWAN	Register and access experts	Monitor faculty research outcomes
13	Shodh Shuddhi	Ensure plagiarism-free work	Improve research quality and reputation
14	Academic Bank of Credits	Store and transfer credits	Facilitate credit redemption

Table 1.1: Summary of ICT Initiatives by the Ministry of Education

1.2 FOSSEE Project

The FOSSEE (Free/Libre and Open Source Software for Education) project promotes the use of FLOSS tools in academia and research. It is part of the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education (MoE), Government of India.

1.2.1 Projects and Activities

The FOSSEE Project supports the use of various FLOSS tools to enhance education and research. Key activities include:

- **Textbook Companion**: Porting solved examples from textbooks using FLOSS.
- **Lab Migration**: Facilitating the migration of proprietary labs to FLOSS alternatives.
- **Niche Software Activities**: Specialized activities to promote niche software tools.
- **Forums**: Providing a collaborative space for users.
- **Workshops and Conferences**: Organizing events to train and inform users.

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

Students from any degree and academic stage can apply for these internships. Selection is based on the completion of screening tasks involving programming, scientific computing, or data collection that benefit the FLOSS community. These tasks are designed to be completed within a week.

For more details, visit the official FOSSEE website.

Figure 1.1: FOSSEE Projects and Activities

1.3 Osdag Software

Osdag (Open steel design and graphics) is a cross-platform, free/libre and open-source software designed for the detailing and design of steel structures based on the Indian Standard IS 800:2007. It allows users to design steel connections, members, and systems through an interactive graphical user interface (GUI) and provides 3D visualizations of designed components. The software enables easy export of CAD models to drafting tools for construction/fabrication drawings, with optimized designs following industry best practices [1, 2, 3]. Built on Python and several Python-based FLOSS tools (e.g., PyQt and PythonOCC), Osdag is licensed under the GNU Lesser General Public License (LGPL) Version 3.

1.3.1 Osdag GUI

The Osdag GUI is designed to be user-friendly and interactive. It consists of

- **Input Dock**: Collects and validates user inputs.
- **Output Dock**: Displays design results after validation.
- **CAD Window**: Displays the 3D CAD model, where users can pan, zoom, and rotate the design.
- **Message Log**: Shows errors, warnings, and suggestions based on design checks.

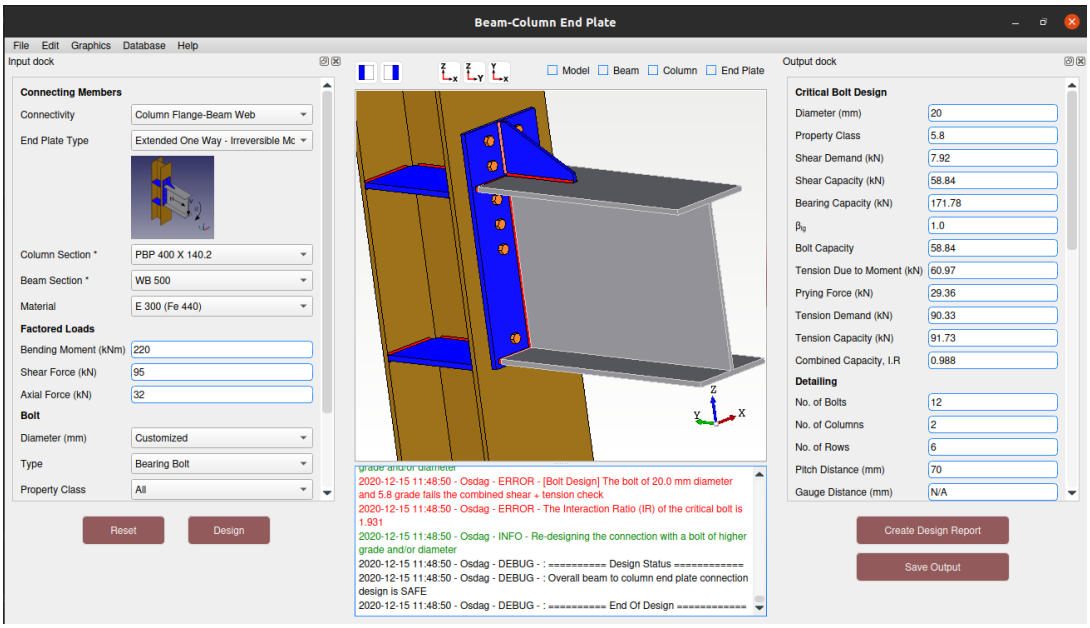

Figure 1.2: Osdag GUI

1.3.2 Features

- **CAD Model**: The 3D CAD model is color-coded and can be saved in multiple formats such as IGS, STL, and STEP.
- **Design Preferences**: Customizes the design process, with advanced users able to set preferences for bolts, welds, and detailing.
- **Design Report**: Creates a detailed report in PDF format, summarizing all checks, calculations, and design details, including any discrepancies.

For more details, visit the official Osdag website.

Chapter 2

Screening Task

2.1 Problem Statement

3D Modeling of a Portal Frame Structure.
The objective was to develop a 3D model of a steel portal frame using the reference figure provided. An incomplete Python script (portal_frame.py) was supplied to serve as a starting point for creating a parametric model. This model was designed so that the frame's dimensions could be easily adjusted. The final 3D visualization of the portal frame was to be rendered using the Open Cascade (OCC) library.

2.2 Screening Tasks Done

I first learned about python Open Cascade- OCC library and its function and then applied it to fix the error and complete the (portal_frame.py) Python script which generated a complete CAD of portal frame as shown below:

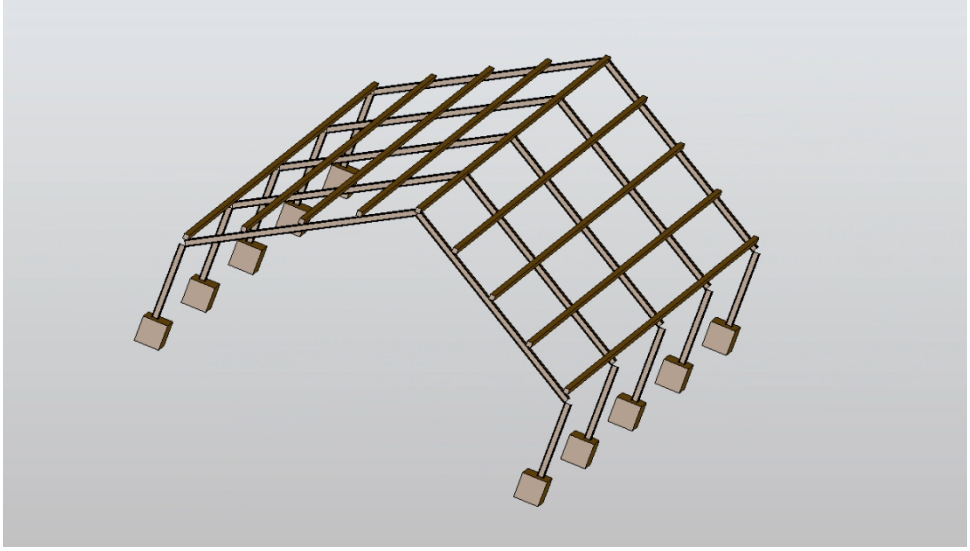

Figure 2.1: Final CAD of portal frame structure

2.3 Screening task: Python Code

2.3.1 Description of the Script

The scripts `portal_frame.py`, `draw_i_section.py`, and `draw_rectangular_prism.py` collectively generate a CAD model of a portal frame structure using the Python Open Cascade (OCC) library. The structure consists of I-section columns and rafters, with rectangular prisms forming the roof, as visualized in the provided screenshot. The scripts are structured as follows:

- **Input parameters**: Specified in `portal_frame.py`, including:
  - `column_height`: 4000 mm
  - `column_thickness`: 250 mm
  - `rafter_length`: 9000 mm
  - `rafter_angle`: user input
  - `roof_length`, `roof_width`, `roof_height`: derived, typically 200 mm each
  - `flange_thickness`: 20 mm
  - `web_thickness`: 9.1 mm
  - `desired_number_of_prisms`: user input

- **Design Calculation**:
  - `draw_i_section.py`: Creates I-section profiles for columns and rafters using `BRepPrimAPI_MakeBox` and `BRepAlgoAPI_Fuse`.
  - `draw_rectangular_prism.py`: Generates roof prisms using `BRepPrimAPI_MakeBox`.
  - `portal_frame.py`: Assembles the structure by:
    * Creating five pairs of vertical I-section columns, spaced by rafter_length.
    * Adding inclined I-section rafters from column tops to a calculated apex.
    * Placing roof prisms along rafters, spaced by rafter_length $\times$ cos(angle)/(num_prisms$-$1).
    * Adding base boxes (1000 $\times$ 1000 $\times$ 1000 mm) at column bases.
- **Output**: The scripts produce a complete CAD model of the portal frame structure as a `TopoDS_Shape`, visualized using the OCC display module (`init_display`).

2.3.2 Python Code

The Python script is shown below. Each section is commented for clarity.

Listing 2.1: Draw_i_section.py

```
1  from OCC.Core.gp import gp_Vec, gp_Trsf
2  from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox
3  from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
4  from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_Transform
5  from OCC.Display.SimpleGui import init_display
6
7
8  def create_i_section(length, width, depth, flange_thickness,
       web_thickness):
9      """
10     Create an I-section CAD model with the specified dimensions.
11
12     Parameters:
13     - length: Length of the I-section
14     - width: Width of the I-section (horizontal dimension)
15     - height: Total height of the I-section (vertical dimension)
16     - flange_thickness: Thickness of the flanges
```

```
17      - web_thickness: Thickness of the web
18
19      Returns:
20      - i_section_solid: The I-section CAD model as a TopoDS_Solid
21      """
22      # Dimensions for the I-section
23      web_height = depth - 2 * flange_thickness
24
25      # Create the bottom flange
26      bottom_flange = BRepPrimAPI_MakeBox(length, width, flange_thickness
            ).Shape()
27
28      # Create the top flange
29      top_flange = BRepPrimAPI_MakeBox(length, width, flange_thickness).
            Shape()
30      trsf = gp_Trsf()
31      trsf.SetTranslation(gp_Vec(0, 0, depth - flange_thickness))  # Move
             to the top
32      top_flange_transform = BRepBuilderAPI_Transform(top_flange, trsf,
            True).Shape()
33
34      # Create the web
35      web = BRepPrimAPI_MakeBox(length, web_thickness, web_height).Shape
            ()
36      trsf = gp_Trsf()
37      trsf.SetTranslation(gp_Vec(0, (width - web_thickness) / 2,
            flange_thickness))  # Centered between flanges
38      web_transform = BRepBuilderAPI_Transform(web, trsf, True).Shape()
39
40      # Combine the flanges and web to form the I-section
41      i_section_solid = BRepAlgoAPI_Fuse(bottom_flange,
            top_flange_transform).Shape()
42      i_section_solid = BRepAlgoAPI_Fuse(i_section_solid, web_transform).
            Shape()
43
44      return i_section_solid
45
46
47  if __name__ == "__main__":
```

```
    length = 1000.0
    width = 100.0  # Width of the I-section
    height = 200.0  # Height of the I-section
    flange_thickness = 10.0  # Thickness of the flanges
    web_thickness = 5.0  # Thickness of the web

    i_section = create_i_section(length, width, height,
        flange_thickness, web_thickness)

    # Visualization
    display, start_display, add_menu, add_function_to_menu =
        init_display()

    # Show the I-section model
    display.DisplayShape(i_section, update=True)
    display.FitAll()
    start_display()
```

Listing 2.2: portal_frame.py

```
import sys
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox
from OCC.Display.SimpleGui import init_display
from OCC.Core.gp import gp_Pnt


def create_rectangular_prism(length, breadth, height):
    # Create a rectangular prism using BRepPrimAPI_MakeBox
    box = BRepPrimAPI_MakeBox(length, breadth, height).Shape()
    return box


def display_prism(box):
    # Initialize the display window
    display, start_display, add_menu, add_function_to_menu =
        init_display()

    # Display the box
    display.DisplayShape(box, update=True)

```

```
    # Start the display loop
    start_display()


if __name__ == "__main__":
    # Dimensions of the rectangular prism
    length = 40.0
    breadth = 20.0
    height = 100.0

    # Create the rectangular prism
    box = create_rectangular_prism(length, breadth, height)

    # Display the rectangular prism
    display_prism(box)
```

Listing 2.3: draw_rectangular_prism

```
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_Transform
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox
from OCC.Core.gp import gp_Vec, gp_Trsf, gp_Ax1, gp_Dir, gp_Pnt
from OCC.Display.SimpleGui import init_display
from draw_i_section import create_i_section
from draw_rectangular_prism import create_rectangular_prism
import math

def create_custom_structure(
    column_height, column_thickness, rafter_length, rafter_angle,
    roof_length, roof_width, roof_height, flange_thickness,
        web_thickness,
    prism_spacing
):
    """
    Create a CAD model of the described structure with prisms placed
        uniformly along the rafters to form the roof.

    Parameters:
    - column_height: Height of the columns
    - column_thickness: Thickness of the I-section columns
```

```
    - rafter_length: Length of the rafters
    - rafter_angle: Angle of inclination of the rafters (in degrees)
    - roof_length: Length of the rectangular roof prism
    - roof_width: Width of the rectangular roof prism
    - roof_height: Height of the rectangular roof prism
    - flange_thickness: Thickness of the I-section flanges
    - web_thickness: Thickness of the I-section web
    - prism_spacing: Horizontal distance between each roof prism along
        the rafter

    Returns:
    - structure: The complete structure as a TopoDS_Shape
    """
    # Convert angle from degrees to radians
    angle_rad = math.radians(rafter_angle)

    # Distance between opposite vertical columns
    column_spacing_x = 2 * rafter_length * math.cos(angle_rad)

    # Create vertical I-section columns
     # Create vertical I-section columns (passing column_height as
        first parameter)
    column = create_i_section(column_height, column_thickness,
       column_thickness, flange_thickness, web_thickness)

    # Rotate to make it vertical
    trsf = gp_Trsf()
    trsf.SetRotation(gp_Ax1(gp_Pnt(0, 0, 0), gp_Dir(0, 1, 0)), -math.pi
        / 2)  # Rotate 90   around Y-axis
    column = BRepBuilderAPI_Transform(column, trsf, True).Shape()
    columns = []

    # Position columns
    num_columns = 5
    column_spacing_y = rafter_length
    for i in range(num_columns):
        # Left row
        trsf = gp_Trsf()
        trsf.SetTranslation(gp_Vec(0, i * column_spacing_y, 0))
```

```
        left_column = BRepBuilderAPI_Transform(column, trsf, True).
            Shape()
        columns.append(left_column)

        # Right row
        trsf = gp_Trsf()
        trsf.SetTranslation(gp_Vec(column_spacing_x+column_thickness, i
            * column_spacing_y, 0))
        right_column = BRepBuilderAPI_Transform(column, trsf, True).
            Shape()
        columns.append(right_column)

    # Fuse columns into a single shape
    structure = columns[0]
    for col in columns[1:]:
        structure = BRepAlgoAPI_Fuse(structure, col).Shape()

    box = BRepPrimAPI_MakeBox(1000, 1000, 1000).Shape()  # Create a box
        of size 1000x1000x100
    for i in range(num_columns):
        # Left column base position
        column_base_left = gp_Vec(-column_thickness*2, i *
            column_spacing_y-column_thickness*1.5, 0)
        trsf_left = gp_Trsf()
        trsf_left.SetTranslation(column_base_left)
        box_left = BRepBuilderAPI_Transform(box, trsf_left, True).Shape
            ()

        # Right column base position
        column_base_right = gp_Vec(column_spacing_x-column_thickness*2,
            i * column_spacing_y-column_thickness*1.5, 0)
        trsf_right = gp_Trsf()
        trsf_right.SetTranslation(column_base_right)
        box_right = BRepBuilderAPI_Transform(box, trsf_right, True).
            Shape()

        # Fuse the boxes into the structure
        structure = BRepAlgoAPI_Fuse(structure, box_left).Shape()
        structure = BRepAlgoAPI_Fuse(structure, box_right).Shape()
```

```
87
88      # Create inclined rafters for each column
89      rafter = create_i_section(rafter_length, column_thickness,
            column_thickness, flange_thickness, web_thickness)
90
91      for i in range(num_columns):
92          # Get column tops for left and right sides
93          left_column_top = gp_Pnt(0, i * column_spacing_y, column_height
                )
94          right_column_top = gp_Pnt(0, i * column_spacing_y,
                column_height)
95
96          # Apex of the rafters (the center point above the two columns)
97          apex = gp_Pnt(column_spacing_x / 2, i * column_spacing_y,
                column_height + math.tan(angle_rad) * (column_spacing_x / 2)
                )
98
99          # Left rafter for the current frame (connects left column to
                apex)
100         trsf = gp_Trsf()
101         trsf.SetTranslation(gp_Vec(0, i * column_spacing_y,
                column_height))  # Position at left column
102         rafter_left = BRepBuilderAPI_Transform(rafter, trsf, True).
                Shape()
103
104         # Right rafter for the current frame (connects right column to
                apex)
105         trsf = gp_Trsf()
106         trsf.SetTranslation(gp_Vec(column_spacing_x, i *
                column_spacing_y, column_height))  # Position at right
                column
107         rafter_right = BRepBuilderAPI_Transform(rafter, trsf, True).
                Shape()
108
109         # Adjust the rafter to the apex, and change its orientation
                accordingly
110         trsf_left = gp_Trsf()
111         trsf_left.SetTranslation(gp_Vec(0, i * column_spacing_y,
                column_height))  # Left rafter base at the left column
```

```
112         trsf_left.SetRotation(gp_Ax1(gp_Pnt(0, i * column_spacing_y,
                column_height), gp_Dir(0, 1, 0)), -angle_rad)  # Rotate to
                desired angle
113         rafter_left = BRepBuilderAPI_Transform(rafter_left, trsf_left,
                True).Shape()
114
115         trsf_right = gp_Trsf()
116         adjust=20
117         trsf_right.SetTranslation(gp_Vec(column_spacing_x, i *
                column_spacing_y, column_height))  # Right rafter base at
                the right column
118         trsf_right.SetRotation(gp_Ax1(gp_Pnt(column_spacing_x+adjust, i
                 * column_spacing_y, column_height+column_thickness/2.1),
                gp_Dir(0, 1, 0)), angle_rad-math.pi)  # Rotate to desired
                angle
119         rafter_right = BRepBuilderAPI_Transform(rafter_right,
                trsf_right, True).Shape()
120
121         # Fuse the rafters into the structure
122         structure = BRepAlgoAPI_Fuse(structure, rafter_left).Shape()
123         structure = BRepAlgoAPI_Fuse(structure, rafter_right).Shape()
124
125     # Left Rafter
126     num_prisms = desired_number_of_prisms  # Set the desired number of
            prisms
127     prism_spacing = rafter_length * math.cos(angle_rad) / (num_prisms -
            1)  # Divide the rafter length by the gaps (num_prisms - 1)
128
129     for i in range(num_prisms):
130         if (i==num_prisms-1):
131         # Calculate the position along the rafter (X and Z coordinates)
132             x_position_left = i * prism_spacing+roof_width/2  # Along
                    the rafter length (X-axis) for the left rafter
133             z_position_left = column_height + roof_height + math.tan(
                    angle_rad) * x_position_left -column_thickness/2#
                    Adjusted Z-position along the slope for the left rafter
134         else:
135             x_position_left = i * prism_spacing  # Along the rafter
                    length (X-axis) for the left rafter
```

```
            z_position_left = column_height + roof_height + math.tan(
                angle_rad) * x_position_left +flange_thickness*3.5 #
                Adjusted Z-position along the slope for the left rafter

        # Create a roof prism
        roof_left = create_rectangular_prism(roof_length, roof_width,
            roof_height)

        # Translation to the position along the left rafter
        translation_left = gp_Trsf()
        translation_left.SetTranslation(gp_Vec(x_position_left, 0,
            z_position_left))  # Move to the left rafter position
        roof_left = BRepBuilderAPI_Transform(roof_left,
            translation_left, True).Shape()

        # Rotation to make the prism perpendicular to the left rafter
        rotation_axis_left = gp_Ax1(gp_Pnt(x_position_left, 0,
            z_position_left), gp_Dir(math.sin(0), 0, math.cos(0)))
        rotation_left = gp_Trsf()
        rotation_left.SetRotation(rotation_axis_left, math.pi / 2)  #
            Rotate 90 degrees to make it perpendicular
        roof_left = BRepBuilderAPI_Transform(roof_left, rotation_left,
            True).Shape()

        # Fuse the prism into the left rafter structure
        structure = BRepAlgoAPI_Fuse(structure, roof_left).Shape()

    # Right Rafter
    for i in range(num_prisms-1):
        # Calculate the position along the rafter (X and Z coordinates)
        x_position_right = column_spacing_x - i * prism_spacing
        x_position = -i * prism_spacing  # Along the rafter length (X-
            axis) for the right rafter (negative direction)
        z_position_right = column_height + math.tan(angle_rad) * abs(
            x_position) + column_thickness*1.45 # Adjusted Z-position
            along the slope for the right rafter

        # Create a roof prism
        roof_right = create_rectangular_prism(roof_length, roof_width,
```

```
            roof_height)
164
165         # Translation to the position along the right rafter
166         translation_right = gp_Trsf()
167         translation_right.SetTranslation(gp_Vec(x_position_right, 0,
                z_position_right))  # Move to the right rafter position
168         roof_right = BRepBuilderAPI_Transform(roof_right,
                translation_right, True).Shape()
169
170         # Rotation to make the prism perpendicular to the right rafter
171         rotation_axis_right = gp_Ax1(gp_Pnt(x_position_right, 0,
                z_position_right), gp_Dir(math.sin(0), 0, math.cos(0)))
172         rotation_right = gp_Trsf()
173         rotation_right.SetRotation(rotation_axis_right, math.pi / 2)  #
                Rotate 90 degrees to make it perpendicular
174         roof_right = BRepBuilderAPI_Transform(roof_right,
                rotation_right, True).Shape()
175
176         # Fuse the prism into the right rafter structure
177         structure = BRepAlgoAPI_Fuse(structure, roof_right).Shape()
178
179     return structure
180
181
182 if __name__ == "__main__":
183     # Define dimensions
184     column_height = 4000.0
185     column_thickness = 250.0#please don't change this value as many
            calculation done on this basis
186     rafter_length = 9000.0
187     rafter_angle = float(input("Enter the angle of inclination:\n"))
188     roof_length = rafter_length * 4 + 200
189     roof_width = 200.0
190     roof_height = 200.0
191     flange_thickness = 20
192     web_thickness = 9.10
193     desired_number_of_prisms = int(input("Enter the number of prisms:\n
            "))
194     prism_spacing = rafter_length / (desired_number_of_prisms - 1)
```

```
195
196     # Create the structure
197     custom_structure = create_custom_structure(
198         column_height, column_thickness, rafter_length, rafter_angle,
199         roof_length, roof_width, roof_height, flange_thickness,
            web_thickness, prism_spacing
200     )
201
202     # Visualization
203     display, start_display, add_menu, add_function_to_menu =
        init_display()
204     display.DisplayShape(custom_structure, update=True)
205     display.FitAll()
206     start_display()
```

Chapter 3

Bolted Butt Joint

3.1 Problem Statement

To write a python script to generate the CAD for displaying the bolted butt joint of two plates.

3.2 Task Done

Bolted Butt Joint A bolted butt joint is a type of mechanical joint commonly used in structural and mechanical engineering to connect two members end-to-end (butted together) using bolts. It is particularly prevalent in steel construction, bridges, trusses, pipelines, and machinery where components need to be extended or assembled modularly.

In this task I created a python script that generate and display the CAD of bolted butt joint where there will be the bolt CAD and two plates

3.3 Python Code

3.3.1 Description of the Script

The script is structured as follows:

- **Input Parameters**: The user specifies input values such as plate thicknesses, plate width, bolt diameter, number of bolt rows and columns, bolt spacing (pitch,

gauge, edge, end), and total number of bolts. These parameters are defined in the `create_bolted_butt_joint` function and control the geometry and layout of the joint components.

- **Design Calculations**: The program calculates derived values such as plate length, bolt head and nut dimensions, and bolt positions using nested loops. It creates instances of `Plate`, `Bolt`, and `Nut` objects and places them at calculated positions to form the butt joint. The bolt arrangement is based on the specified pitch and gauge, and placement stops once the desired number of bolts is reached.

- **Output**: The script generates and visualizes a 3D CAD model of the bolted butt joint using the Open Cascade (OCC) display module. The output includes the top and bottom plates, a cover plate, and correctly positioned bolts and nuts. Components are displayed in distinct colors, and a small red sphere at the origin helps in identifying the global coordinate system.

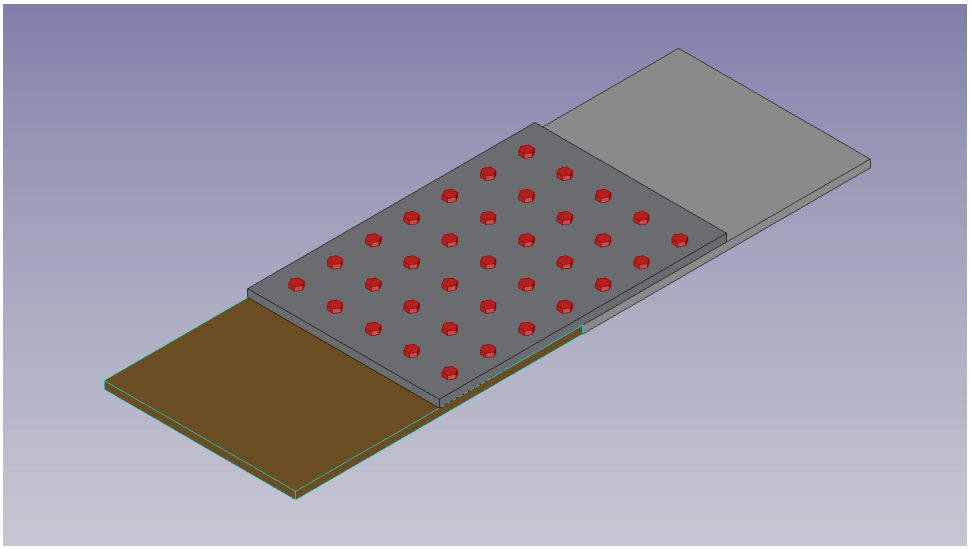

Figure 3.1: Single Plated Bolted Butt Joint

3.3.2 Python Script

The Python script is shown below. Each section is commented for clarity.

Listing 3.1: Single Plated Bolted Butt Joint

```
1 %--------------------begin code-------------
2
3 import numpy
```

```
4  from OCC.Display.SimpleGui import init_display
5  from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
6  from OCC.Core.BOPAlgo import BOPAlgo_Builder
7  from OCC.Core.Quantity import Quantity_NOC_SADDLEBROWN,
       Quantity_NOC_GRAY,Quantity_NOC_BLUE1,Quantity_NOC_RED,Quantity_Color
       , Quantity_TOC_RGB
8  from OCC.Core.Graphic3d import Graphic3d_NOM_ALUMINIUM,
       Graphic3d_NOM_STEEL
9  from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeSphere
10 # Import the component classes
11 from bolt import Bolt
12 from nut import Nut
13 from plate import Plate
14
15 def create_bolted_butt_joint(plate1_thickness = 4, plate2_thickness =
       4,cover_thickness=3, plate_width = 100, bolt_dia = 16,
16                            bolt_rows=5,bolt_cols=7,pitch=20,gauge=20,
                                  edge=12,end=13.6,number_bolts=7):
17     plate_length = 1.5*plate_width
18     nut_thickness = 3.0
19     # Bolt parameters
20     bolt_head_radius = bolt_dia/2
21     bolt_head_thickness = 3.0
22     bolt_length = (plate1_thickness + plate2_thickness) + nut_thickness
           # Enough to go through both plates
23     bolt_shaft_radius = 1.5
24
25     # Nut parameters
26     nut_radius = bolt_head_radius
27
28     nut_height = bolt_head_radius
29     nut_inner_radius = bolt_shaft_radius
30
31     # Create the first plate
32     # Position it at the origin
33     origin1 = numpy.array([0.0, 0.0, 0.0]) # Global origin lies at
           midpoint of plate 1
34     uDir1 = numpy.array([0.0, 0.0, 1.0])  # Points along Z axis (height
           )
```

```
    wDir1 = numpy.array([1.0, 0.0, 0.0])  # Points along X axis (length)

    plate1 = Plate(plate_length, plate_width, plate1_thickness)
    plate1.place(origin1, uDir1, wDir1)
    plate1_model = plate1.create_model()

    # Create the second plate
    # Position it so that it properly overlaps with the first plate
    # The second plate is elevated by plate1_thickness and offset in Y direction

    origin2 = numpy.array([0.0,plate_length, 0])
    uDir2 = numpy.array([0.0, 0.0, 1.0])
    wDir2 = numpy.array([1.0, 0.0, 0.0])

    plate2 = Plate(plate_length, plate_width, plate2_thickness)
    plate2.place(origin2, uDir2, wDir2)
    plate2_model = plate2.create_model()

    origin3 = numpy.array([0.0, (plate_width*1.5)/2, max(plate1_thickness, plate2_thickness)]) # Global origin lies at midpoint of plate 1
    uDir3 = numpy.array([0.0, 0.0, 1.0])  # Points along Z axis (height)
    wDir3 = numpy.array([1.0, 0.0, 0.0])  # Points along X axis (length)

    platec = Plate(plate_length, plate_width, plate1_thickness)
    platec.place(origin3, uDir3, wDir3)
    platec_model = platec.create_model()
    # --- Bolt Dimensions ---
    T = 2
    R, H, r = 3, (max(plate1_thickness, plate2_thickness) + cover_thickness + T) * 1.25, 1
    innerR1 = 1

    # --- Calculate Bolt Positions ---
    bolt_positions = []
```

```
    count = 0
    exit_loops = False

    for col in range(bolt_cols):
        for row in range(bolt_rows):
            bolt_positions.append((
                edge + (row * gauge),
                end + (col * pitch),
                (max(plate1_thickness, plate2_thickness))/2 +
                    cover_thickness+T/2
            ))
            count += 1

            # Exit after completing current column
            if count == number_bolts and row == bolt_rows - 1:
                exit_loops = True
                break
        if exit_loops:
            break

    # --- Create and Place Bolts & Nuts ---
    bolts_models = []
    nuts_models = []
    bolt_uDir = numpy.array([1.0, 0.0, 0.0])
    bolt_shaftDir = numpy.array([0.0, 0.0, -1.0])

    for pos in bolt_positions:
        # Bolt
        bolt = Bolt(R, T, H, r)
        bolt.place(pos, bolt_uDir, bolt_shaftDir)
        bolt_model = bolt.create_model()
        bolts_models.append(bolt_model)

        # Nut
        nut_origin = numpy.array([pos[0], pos[1],-T])
        nut_uDir = numpy.array([1.0, 0.0, 0.0])
        nut_wDir = numpy.array([0.0, 0.0, -1.0])

        nut = Nut(R, T, H, innerR1)
```

```
105             nut.place(nut_origin, nut_uDir, nut_wDir)
106             nut_model = nut.create_model()
107             nuts_models.append(nut_model)
108 
109     return plate1_model, plate2_model,platec_model, bolts_models,
            nuts_models
110 
111 # Main execution
112 if __name__ == "__main__":
113     # Create the bolted lap joint
114     plate1, plate2,platec,bolts,nuts = create_bolted_butt_joint()
115 
116     redd=Quantity_Color(0.28, 0, 0, Quantity_TOC_RGB)
117 
118     # Display the assembly
119     display, start_display, add_menu, add_function_to_menu =
            init_display()
120 
121     # Display individual components with different colors for better
            visualization
122     display.DisplayShape(plate1, update=True)
123     display.DisplayShape(plate2,material=Graphic3d_NOM_ALUMINIUM,
            update=True)
124     display.DisplayShape(platec, material=Graphic3d_NOM_STEEL, update=
            True)
125 
126     # --- Display Bolts and Nuts ---
127     for bolt_model in bolts:
128         display.DisplayShape(bolt_model, color=redd, update=True)
129 
130     for nut_model in nuts:
131         display.DisplayShape(nut_model,  color=redd, update=True)
132 
133     #display.DisplayShape(nut, color=Quantity_NOC_SADDLEBROWN, update=
            True)
134     # Highlight the global origin (0,0,0)
135     origin_point = BRepPrimAPI_MakeSphere(1).Shape()  # Small sphere to
            mark origin
136     display.DisplayShape(origin_point, color=Quantity_NOC_RED, update=
```

```
        True)
137
138     # Alternative: display the full assembly as a single shape
139     # display.DisplayShape(lap_joint, update=True)
140     display.set_bg_gradient_color([51, 51, 102], [150, 150, 170])
141
142     display.DisableAntiAliasing()
143     display.FitAll()
144     start_display()
145 %-------------------- end code ---------------
```

3.3.3 Explanation of the Code

- **Lines 1–5**: Imports necessary Python libraries and Open Cascade (OCC) modules for CAD modeling and display.
- **Lines 7–11**: Defines default parameters for the bolted butt joint, such as plate thicknesses, bolt dimensions, plate width, and bolt arrangement (rows, columns, pitch, gauge, edge, and number of bolts).
- **Lines 13–26**: Calculates the plate dimensions and bolt properties, such as bolt shaft radius, head thickness, and nut thickness. These values are used to create bolts and nuts that match the connection geometry.
- **Lines 28–47**: Creates and positions the first plate (`plate1`) at the global origin using the `Plate` class. The plate is placed with its length along the X-axis and height along the Z-axis.
- **Lines 49–58**: Creates and places the second plate (`plate2`) above and behind the first plate, offset along the Y-axis by the plate length, and elevated along Z to avoid overlap.
- **Lines 60–69**: Creates a cover plate (`platec`) to simulate the middle joining plate of the butt joint, positioned above the two base plates.
- **Lines 73–86**: Sets parameters for bolt geometry and calculates the positions of bolts using nested loops. The loop exits once the required number of bolts is reached.

- **Lines 88–105**: Instantiates each `Bolt` and `Nut` object at the calculated positions. Bolts are oriented vertically through the joint, and nuts are placed below the bottom plate.
- **Lines 109–112**: Calls the `create_bolted_butt_joint` function in the `main` block to generate the 3D models of plates, bolts, and nuts.
- **Lines 114–129**: Initializes the OCC 3D display window and renders the assembled joint. Individual components (plates, bolts, and nuts) are displayed in distinct colors for clarity.
- **Lines 131–133**: Adds a red sphere at the origin for reference and sets a background gradient for better visualization. The view is then fitted and the OCC GUI loop is started.

3.4 Documentation

My contribution your contribution towards Osdag on the directory below:

3.4.1 Directory Structure

```
Osdag
├─ osdagMainPage.py
├─ cad
└─ gui
```

Chapter 4

Welded Butt Joint

4.1 Problem Statement

To write a python script to generate the CAD for displaying the Welded Bolt Joint of two plates.

4.2 Task Done

Welded Butt Joint

A welded butt joint is a type of permanent mechanical connection where two structural members are joined end-to-end (butted together) and fused using welding. This type of joint is widely used in structural steel fabrication, pipelines, pressure vessels, and automotive or aerospace components, offering high strength and continuity without the need for mechanical fasteners.

In this task, I created a Python script that generates and displays the CAD of a welded butt joint. The model includes two plates joined end-to-end with a visual representation of the weld seam, simulating a typical full penetration butt weld in CAD. The weld geometry can be modeled as a fillet, groove, or a simplified seam depending on the level of detail required.

4.3 Python Code

4.3.1 Description of the Script

The script is structured as follows:

- **Input Parameters**: The user specifies values such as plate width, plate thicknesses (for both plates and the cover plate), weld size (leg length), and weld length. These inputs are defined in the `create_welded_butt_joint` function and control the geometry and configuration of the welded joint components.
- **Design Calculations**: The script calculates derived values such as plate length (1.5 times the plate width), the 3D placement origins and directions for each plate and weld, and appropriate positioning to avoid overlap. The plates are created using the `Plate` class, and welds are created using the `FilletWeld` class. Two welds are placed at the junctions of the two main plates to simulate a full-strength butt weld on both ends. Direction vectors ensure proper alignment of the weld geometry.
- **Output**: The script generates a 3D CAD model of a welded butt joint assembly using the Open Cascade (OCC) display module. It includes two plates joined end-to-end, a cover plate placed centrally on top, and fillet welds shown in red for visual emphasis. A sphere at the global origin is added for reference, and a background gradient improves visual clarity in the OCC viewer.

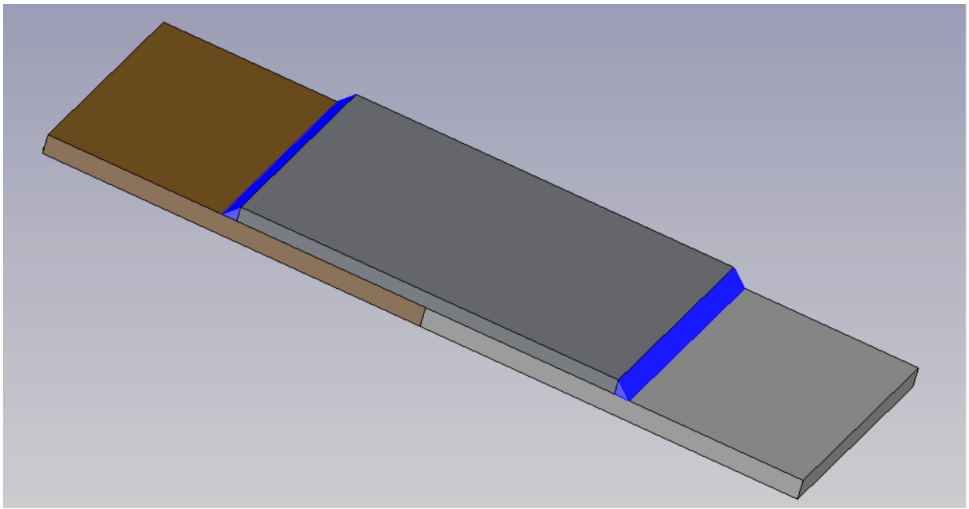

Figure 4.1: Single Plated Welded Butt Joint

4.3.2 Python Script

The Python script is shown below. Each section is commented for clarity.

Listing 4.1: Single Plated Welded Butt Joint

```
%--------------------begin code-------------

import numpy
from OCC.Display.SimpleGui import init_display
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeSphere
from OCC.Core.Quantity import Quantity_NOC_RED, Quantity_Color,
    Quantity_TOC_RGB
from OCC.Core.Graphic3d import Graphic3d_NOM_ALUMINIUM,
    Graphic3d_NOM_STEEL
from OCC.Core.gp import gp_Pnt
from plate import Plate
from filletweld import FilletWeld  # Import FilletWeld class

def create_welded_butt_joint(plate_width=100, plate1_thickness=10,
    plate2_thickness=10, cover_thickness=8, weld_size=6, weld_length=80)
    :
    plate_length = 1.5 * plate_width
    weld_length=plate_width
    # --- Create the first plate ---
    origin1 = numpy.array([0.0, 0.0, 0.0])
    uDir1 = numpy.array([0.0, 0.0, 1.0])  # X direction
    wDir1 = numpy.array([1.0, 0.0, 0.0])  # Z direction

    plate1 = Plate(plate_length, plate_width, plate1_thickness)
    plate1.place(origin1, uDir1, wDir1)
    plate1_model = plate1.create_model()

    # --- Create the second plate ---
    origin2 = numpy.array([0.0, plate_length, 0.0])
    uDir2 = numpy.array([0.0, 0.0, 1.0])
    wDir2 = numpy.array([1.0, 0.0, 0.0])

    plate2 = Plate(plate_length, plate_width, plate2_thickness)
    plate2.place(origin2, uDir2, wDir2)
```

```
    plate2_model = plate2.create_model()

    # --- Create the cover plate ---
    origin3 = numpy.array([0.0, plate_length/ 2, max(plate1_thickness,
        plate2_thickness)/2+cover_thickness/2])
    uDir3 = numpy.array([0.0, 0.0, 1.0])
    wDir3 = numpy.array([1.0, 0.0, 0.0])

    cover_plate = Plate(plate_length, plate_width, cover_thickness)
    #cover_plate_center_origin = origin3 + numpy.array([0, plate_length
        / 2, cover_thickness / 2])
    cover_plate.place(origin3, uDir3, wDir3)
    cover_plate_model = cover_plate.create_model()

    # --- Create the Fillet Welds ---
    welds_models = []

    weld_b = weld_size
    weld_h = weld_size

    # Corrected weld placement for each butt interface
    weld_origins = [
        numpy.array([(plate_width - weld_length) / 2, 0, max(
            plate1_thickness, plate2_thickness)/2]),  # Left weld
        numpy.array([(plate_width - weld_length) / 2, plate_length, max
            (plate1_thickness, plate2_thickness)/2])  # Right weld
    ]

    weld_dirs = [
        {'uDir': numpy.array([0.0, 0.0, 1.0]), 'shaftDir': numpy.array
            ([1.0, 0.0, 0.0])},   # Upward facing weld
        {'uDir': numpy.array([0.0, 1.0, 0.0]), 'shaftDir': numpy.array
            ([1.0, 0.0, 0.0])}   # Downward facing weld
    ]

    for i, origin in enumerate(weld_origins):
        weld = FilletWeld(weld_b, weld_h, weld_length)
        weld.place(origin, weld_dirs[i]['uDir'], weld_dirs[i]['shaftDir
            '])
```

```
        weld_model = weld.create_model()
        welds_models.append(weld_model)

    return plate1_model, plate2_model, cover_plate_model, welds_models

# ================================================================================

if __name__ == "__main__":
    plate1, plate2, cover_plate, welds = create_welded_butt_joint(
        plate_width=120,
        plate1_thickness=12,
        plate2_thickness=12,
        cover_thickness=10,
        weld_size=8,
        weld_length=100
    )

    # Initialize 3D viewer
    display, start_display, add_menu, add_function_to_menu =
        init_display()

    weld_color = Quantity_Color(0.8, 0.1, 0.1, Quantity_TOC_RGB)

    # Display all parts
    display.DisplayShape(plate1, update=True, color='silver')
    display.DisplayShape(plate2, update=True, material=
        Graphic3d_NOM_ALUMINIUM)
    display.DisplayShape(cover_plate, update=True, material=
        Graphic3d_NOM_STEEL)

    for weld_model in welds:
        display.DisplayShape(weld_model, color=weld_color, update=True)

    # Mark origin
    origin_marker = BRepPrimAPI_MakeSphere(gp_Pnt(0, 0, 0), 1.5).Shape
        ()
    display.DisplayShape(origin_marker, color=Quantity_NOC_RED, update=
        True)
```

```
95
96      display.set_bg_gradient_color([20, 20, 50], [150, 150, 170])
97      display.DisableAntiAliasing()
98      display.FitAll()
99      start_display()
100
101 %-------------------- end code ---------------
```

4.3.3 Explanation of the Code

- **Lines 1-5**: Imports essential Python libraries and Open Cascade (OCC) modules for 3D CAD modeling and visualization. This includes tools for geometric primitives, colors, materials, and graphical display.
- **Lines 6-7**: Imports the `Plate` class to create plate components and the `FilletWeld` class to simulate fillet weld geometry at the joint.
- **Lines 9-44**: Defines the function `create_welded_butt_joint`, which takes user-defined parameters such as plate thicknesses, weld size, and weld length. The function calculates derived values like plate length and sets up directions for placement.
- **Lines 11-16**: Creates the first plate (`plate1`) at the global origin using the `Plate` class. The orientation vectors ensure the plate is placed along the global XZ plane.
- **Lines 18-23**: Creates and places the second plate (`plate2`) above the first one along the Y-axis. This simulates the second half of a butt joint, matching the first in dimensions.
- **Lines 25-30**: Creates a centrally placed cover plate above the junction of the two plates to simulate a reinforcing plate in the weld region. It is positioned slightly above the plate top surfaces.
- **Lines 32-42**: Defines the geometry and placement of two welds using the `FilletWeld` class. Two fillet welds are created—one at the front and one at the back—simulating a full joint connection. Direction vectors control the weld orientation (upward/-downward along Z/Y).

- **Lines 44-46**: Returns the models of both plates, the cover plate, and all welds for rendering.
- **Lines 48-55**: In the `main` block, calls the `create_welded_butt_joint` function with specified parameter values. This generates the required 3D models of all components.
- **Lines 57-59**: Initializes the OCC display window and defines the color for welds using RGB format.
- **Lines 61-65**: Renders the individual components in the viewer. Plates are displayed using different OCC material styles for better differentiation. Welds are shown in red for emphasis.
- **Lines 67-69**: Places a small red sphere at the global origin for reference. A gradient background is applied, anti-aliasing is disabled, and the entire model is fitted to the viewport. The GUI loop is started using `start_display()`.

Chapter 5

Double Angle Gusset Joint

5.1 Problem Statement

To write a python script to generate the CAD for displaying the bolted and welded double angle with gusset plate joint places back to back on the same side of the gusset plate.

5.2 Task Done

Double Angle Gusset Joint A gusset plate angle connection is a structural joint used in steel or other frameworks where a gusset plate, typically a triangular or rectangular metal plate, is employed to connect two or more structural members, such as angles, beams, or columns. The gusset plate strengthens the connection by distributing loads and providing stability, often used in trusses, bridges, and buildings.

In an angle connection, the gusset plate is bolted or welded to angle sections (L-shaped steel members) to create a rigid joint.

In this task I created a python script that generate and display the CAD of Double angle gusset Joint with bolted and welded connection where there are two angles placed back to back on the same side of the gusset plate

5.3 Python Code(Bolted gusset joint)

5.3.1 Description of the Script

The script is structured as follows:

- **Input Parameters:** User-defined inputs include plate thickness, width, gusset dimensions, bolt radius, height, and spacing. These control the size and layout of all components in the joint.
- **Design Calculations:** Bolt and nut positions are computed using the trapezoid and bolt geometry. Positions are based on fractions of gusset height and width, and nuts are placed below the plates. Components are placed using orientation vectors.
- **Output:** The OCC viewer renders the full assembly: two gusset plates, an angle section, bolts, and nuts. Each part is displayed with appropriate color/material. The scene is auto-fitted for display using `FitAll()`.

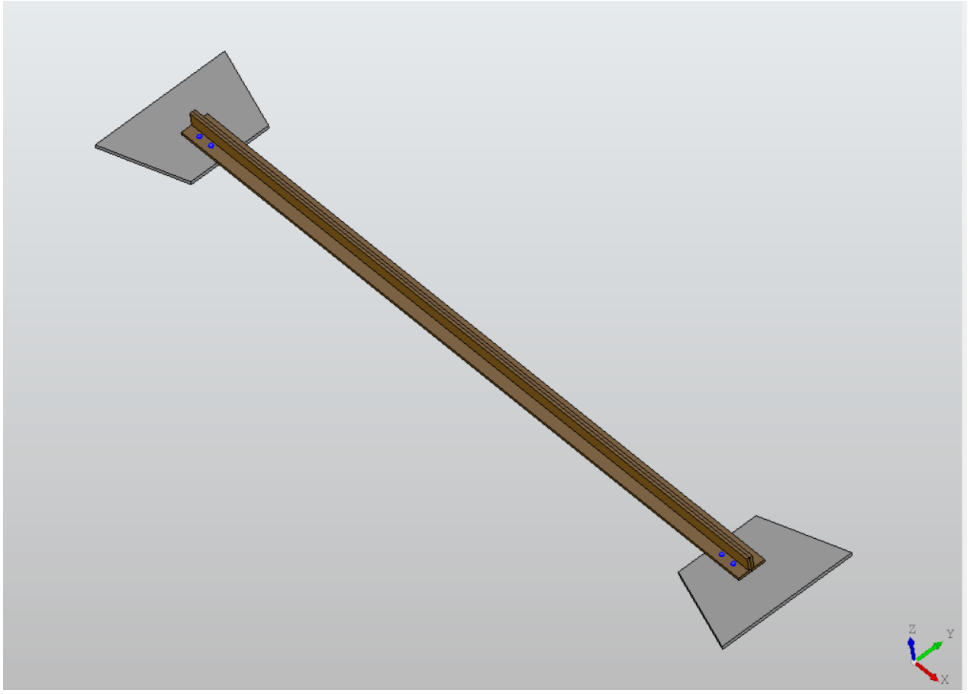

Figure 5.1: Double Angle Bolted Gusset Plate Joint

5.3.2 Python Script

The Python script is shown below. Each section is commented for clarity.

Listing 5.1: Double Angle Bolted Gusset Plate Joint

```
%--------------------begin code-------------

from OCC.Core.gp import gp_Vec, gp_Trsf,gp_Pnt,gp_Ax1,gp_Dir
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_Transform
from OCC.Display.SimpleGui import init_display
from angle_create import create_angle_section
from bolt import Bolt  # Import the Bolt class
from nut import Nut
import numpy
from trapizoid_plate import create_trapizoid
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_MakePolygon,
    BRepBuilderAPI_MakeFace
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakePrism
from OCC.Core.gp import gp_Vec

from OCC.Core.Quantity import Quantity_NOC_SADDLEBROWN #for bolt colors
from OCC.Core.Graphic3d import * #for gray colour
from OCC.Core.Graphic3d import Graphic3d_NOM_ALUMINIUM,
    Graphic3d_NOM_JADE,Graphic3d_NOM_OBSIDIAN,Graphic3d_NOM_NEON_PHC,
    Graphic3d_NOM_STEEL


# Visualization
display, start_display, add_menu, add_function_to_menu = init_display()

length = 1000
width = 20 # Width of the angle
depth = 20  # Height of angle
flange_thickness = 4  # Thickness of the flanges
web_thickness =4 # Thickness of the web
plate_width=100
plate_thickness=6

large_len=200
small_len=40
```

```
trap_height=125


# Define trapezoid corner points (parallel sides along Y-axis)
p1 = gp_Pnt(0, 0, 0)     # Bottom-left (short side)
p2 = gp_Pnt(0, large_len, 0)  # Top-left (short side)
p3 = gp_Pnt(trap_height, 160, 0)  # Top-right (wide side)
p4 = gp_Pnt(trap_height, 40, 0)  # Bottom-right (wide side)

# Create a polygon (wire)
polygon = BRepBuilderAPI_MakePolygon()
polygon.Add(p1)
polygon.Add(p2)
polygon.Add(p3)
polygon.Add(p4)
polygon.Close()

# Create a face from the polygon
face = BRepBuilderAPI_MakeFace(polygon.Wire())

# Extrude to make a 3D plate (thickness of 5 units)

trap_plate1 = BRepPrimAPI_MakePrism(face.Face(), gp_Vec(0, 0,
    plate_thickness)).Shape()


# Define transformation for rotation
trsf = gp_Trsf()
rotation_axis = gp_Ax1(gp_Pnt(0, 0, 0), gp_Dir(0, 0, 1))  # Z-axis
angle = 180 * (3.14159265 / 180)  # Convert degrees to radians
trsf.SetRotation(rotation_axis, angle)
trap_plate2r = BRepBuilderAPI_Transform(trap_plate1, trsf, True).Shape
    ()

# transformation aftter rotation plate2
trsf = gp_Trsf()
trsf.SetTranslation(gp_Vec(length+trap_height,large_len, 0))  # Move to
     the top
trap_plate2 = BRepBuilderAPI_Transform(trap_plate2r, trsf, True).Shape
```

```
    ()
71
72
73
74
75  angle=create_angle_section(length, width, depth, flange_thickness,
        web_thickness)
76
77  trsf = gp_Trsf()
78  trsf.SetTranslation(gp_Vec(trap_height/2,(large_len-2*width)/2,
        plate_thickness))  # Move to the top
79  angle_transform = BRepBuilderAPI_Transform(angle, trsf, True).Shape()
80
81  """
82  p1=(0,0,0)
83  p2=(0,large_len,0)
84  p3=(trap_height,(large_len-small_len)/2+small_len, 0)
85  p4=(trap_height,(large_len-small_len)/2,0)
86
87
88  trap_plate1=create_trapizoid(p1,p2,p3,p4)
89
90  trsf = gp_Trsf()
91  trsf.SetTranslation(gp_Vec(500, 0, 0))  # Move to the top
92  trap_plate2= BRepBuilderAPI_Transform(trap_plate1, trsf, True).Shape()
93  """
94
95  # Bolt Parameters (Smaller bolts)
96  T=2
97  R, H, r = 3, (plate_thickness+flange_thickness+T)*1.25, 1
98
99  # Bolt positions
100 bolt_positions = [
101     numpy.array([(2*trap_height)/3., (large_len-2*width)/2+width/2.,
        plate_thickness+flange_thickness]),  # Bolt 1
102     numpy.array([(5*trap_height)/6., (large_len-2*width)/2+width/2.,
        plate_thickness+flange_thickness]),  # Bolt 2
103     numpy.array([(2*trap_height)/3., (large_len-2*width)/2+(width*3)
        /2.,  plate_thickness+flange_thickness]),  # Bolt 3
```

```
104     numpy.array([(5*trap_height)/6., (large_len-2*width)/2+(width*3)
            /2.,  plate_thickness+flange_thickness]),   # Bolt 4
105     numpy.array([length+trap_height-(2*trap_height)/3., (large_len-2*
            width)/2+width/2.,  plate_thickness+flange_thickness]),  # Bolt
            1
106     numpy.array([length+trap_height-(5*trap_height)/6., (large_len-2*
            width)/2+width/2.,  plate_thickness+flange_thickness]),  # Bolt
            2
107     numpy.array([length+trap_height-(2*trap_height)/3., (large_len-2*
            width)/2+(width*3)/2.,  plate_thickness+flange_thickness]),  #
            Bolt 3
108     numpy.array([length+trap_height-(5*trap_height)/6., (large_len-2*
            width)/2+(width*3)/2.,  plate_thickness+flange_thickness])   #
            Bolt 4
109 ]
110
111
112 # Bolt orientation
113 uDir = numpy.array([1., 0., 0.])       # X-direction
114 shaftDir = numpy.array([0., 0., -1.])  # Z-direction
115
116 # Generate and display bolts
117 for pos in bolt_positions:
118     bolt = Bolt(R, T, H, r)
119     bolt.place(pos, uDir, shaftDir)
120     display.DisplayShape(bolt.create_model(),color="blue", update=True)
121
122
123 #create nuts
124 innerR1 =1
125
126 # Nut positions
127 nut_positions = [
128
129     numpy.array([(2*trap_height)/3., (large_len-2*width)/2+width/2.,  -
            T]),  # Bolt 1
130     numpy.array([(5*trap_height)/6., (large_len-2*width)/2+width/2.,  -
            T]),  # Bolt 2
131     numpy.array([(2*trap_height)/3., (large_len-2*width)/2+(width*3)
```

```
        /2.,  -T]),  # Bolt 3
    numpy.array([(5*trap_height)/6., (large_len-2*width)/2+(width*3)
        /2.,  -T]),  # Bolt 4
    numpy.array([length+trap_height-(2*trap_height)/3., (large_len-2*
        width)/2+width/2.,  -T]),  # Bolt 1
    numpy.array([length+trap_height-(5*trap_height)/6., (large_len-2*
        width)/2+width/2.,  -T]),  # Bolt 2
    numpy.array([length+trap_height-(2*trap_height)/3., (large_len-2*
        width)/2+(width*3)/2.,  -T]),  # Bolt 3
    numpy.array([length+trap_height-(5*trap_height)/6., (large_len-2*
        width)/2+(width*3)/2.,  -T])  # Bolt 4

    ]

# Orientation
uDir = numpy.array([1., 0., 0.])
wDir = numpy.array([0., 0., 1.])

# Generate and display nuts
for pos in nut_positions:
    nut = Nut(R, T, H, innerR1)
    nut.place(pos, uDir, wDir)
    display.DisplayShape(nut.create_model(),color="blue", update=True)



# Show the I-section model
#display.DisplayShape(trap_plate1, update=True)
display.DisplayShape(trap_plate2,material=Graphic3d_NOM_ALUMINIUM,
    update=True)
display.DisplayShape(angle_transform, update=True)
display.DisplayShape(trap_plate1,material=Graphic3d_NOM_ALUMINIUM,
    update=True)
display.FitAll()
start_display()

%-------------------- end code ---------------
```

5.3.3 Explanation of the Code

- **Lines 1–5**: Imports core modules from the OpenCascade (OCC) library needed for geometric modeling, transformations, primitives, and visualization. Modules like `gp_Vec`, `BRepPrimAPI_MakeBox`, and `BRepAlgoAPI_Fuse` are fundamental for 3D modeling.

- **Lines 6–11**: Imports custom utility functions and classes like `create_angle_section`, `Bolt`, `Nut`, and `create_trapizoid`, which are assumed to be user-defined scripts to modularize geometry creation.

- **Lines 13–16**: Initializes the OCC 3D viewer using `init_display()`, enabling interactive visualization of the created CAD models.

- **Lines 18–23**: Defines geometry parameters for the angle section and trapezoidal gusset plates. This includes rod dimensions (`length`, `width`, `depth`), plate thickness, and gusset dimensions such as the longer and shorter bases and height.

- **Lines 26–33**: Sets up 2D corner points of a trapezoidal plate and constructs a wire polygon using these points. These define the base face for the gusset plate.

- **Lines 35–37**: Converts the polygon into a planar face, which is required for 3D extrusion.

- **Lines 39–41**: Extrudes the trapezoidal face along the Z-axis to form a solid 3D gusset plate.

- **Lines 44–47**: Rotates the first gusset plate 180 degrees about the Z-axis to mirror it for the opposite end.

- **Lines 49–51**: Translates the rotated plate to the other end of the rod to complete the symmetric gusset placement.

- **Line 54**: Creates the L-shaped angle section using a helper function `create_angle_section` with parametric inputs.

- **Lines 56–58**: Translates the angle section so that it fits precisely between the two gusset plates, accounting for the plate and flange thickness.

- **Lines 61–63**: Defines bolt dimensions such as shaft radius (`R`), height (`H`), head thickness (`T`), and fillet radius (`r`).
- **Lines 65–74**: Specifies eight bolt locations—four for each gusset—based on plate and gusset geometry. Coordinates are adjusted to place bolts symmetrically on both plates.
- **Lines 77–78**: Sets the orientation vectors for the bolts. The bolts are aligned in the Z-direction (shaftDir) and extend outward in the X-direction (uDir).
- **Lines 80–84**: Iterates over each bolt position, creates a bolt using the `Bolt` class, positions it, and renders it in blue using `display.DisplayShape()`.
- **Line 87**: Sets the inner radius for the nuts, used for visual or dimensional control.
- **Lines 89–98**: Defines nut positions by lowering each corresponding bolt location by `T` units in Z (opposite direction), positioning them beneath the gusset plates.
- **Lines 101–102**: Sets orientation vectors for nuts. The nuts face the opposite direction of bolts to simulate realistic tightening direction.
- **Lines 104–108**: Iterates through each nut position, creates the nut object, places it, and renders it similarly to bolts.
- **Lines 112–115**: Displays the CAD shapes in the viewer, assigning material appearances to the gusset plates and rendering the angle section. The view is adjusted to fit all objects before starting the OCC GUI loop.

5.4 Python Code(Welded gusset joint)

5.4.1 Description of the Script

The script is structured as follows:

- `length, width, depth`: Define the length and cross-sectional dimensions of the angle section (bracing member).
- `flange_thickness, web_thickness`: Control the thickness of the flanges and web in the angle section.

- `plate_width, plate_thickness`: Specify the size and thickness of the gusset plates.
- `large_len, small_len, trap_height`: Dimensions of the trapezoidal gusset plates—used to calculate the shape and extrusion length.
- `b, h`: Represent the leg lengths of the fillet weld cross-section (both equal here to flange thickness).
- `origins`: A list of 3D positions where each fillet weld is placed.
- `uDir, shaftDir`: Orientation vectors controlling the direction and face alignment of each fillet weld.
- `L`: The extrusion length of the fillet weld based on placement context (either half the gusset length or full flange width).

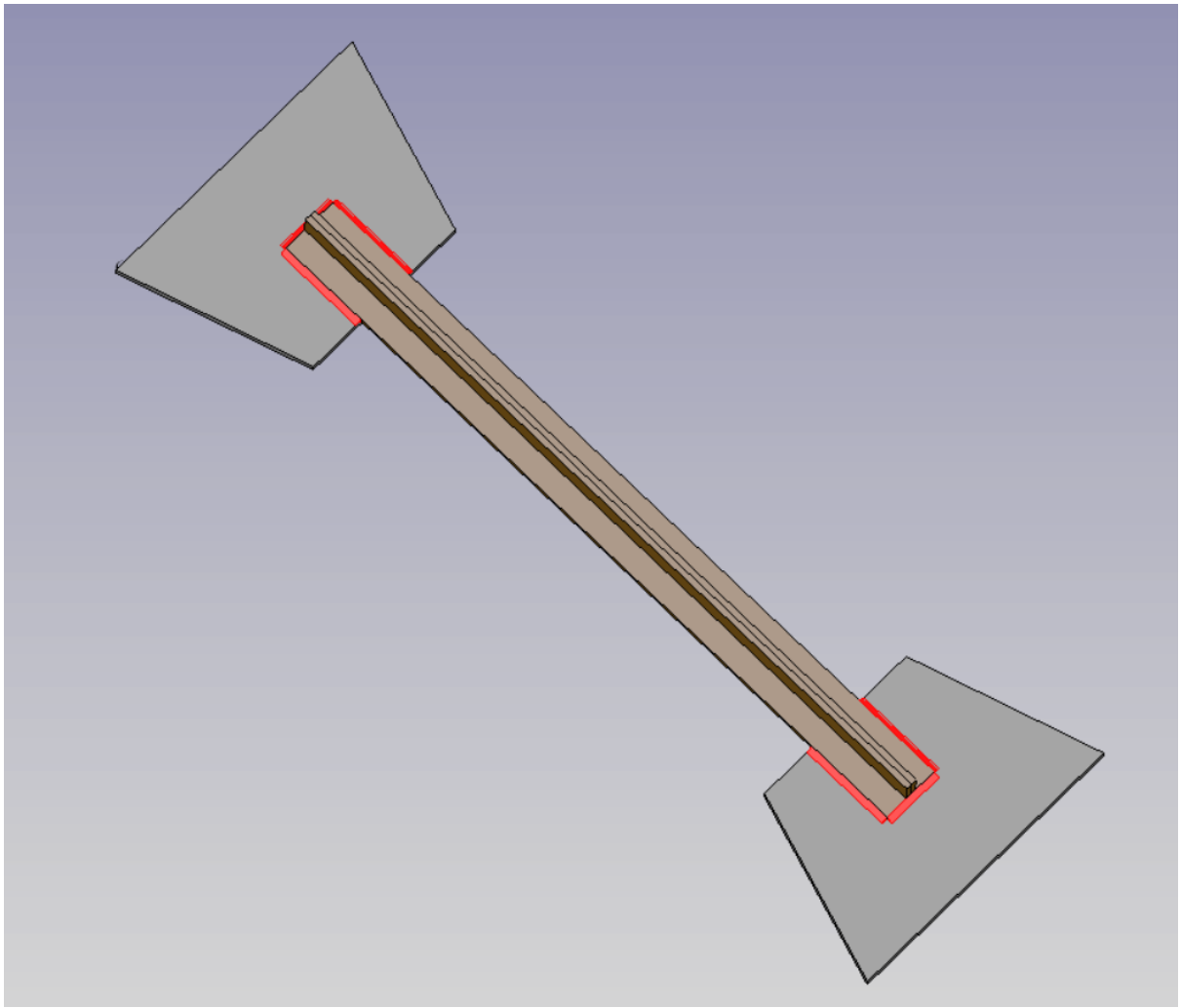

Figure 5.2: Double angle gusset weld joint

5.4.2 Python Script

The Python script is shown below. Each section is commented for clarity.

Listing 5.2: Double angle gusset weld joint

```
%--------------------begin code-------------

from OCC.Core.gp import gp_Vec, gp_Trsf,gp_Pnt,gp_Ax1,gp_Dir
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_Transform
from OCC.Display.SimpleGui import init_display
from angle_create import create_angle_section
from filletweld import FilletWeld  # Import FilletWeld class
import numpy
from trapizoid_plate import create_trapizoid
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_MakePolygon,
    BRepBuilderAPI_MakeFace
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakePrism
from OCC.Core.gp import gp_Vec

from OCC.Core.Quantity import Quantity_Color, Quantity_TOC_RGB
from OCC.Core.Aspect import Aspect_GFM_VER  # Import the correct
    gradient mode

from OCC.Core.Quantity import Quantity_NOC_SADDLEBROWN #for bolt colors
from OCC.Core.Graphic3d import * #for gray colour
from OCC.Core.Graphic3d import Graphic3d_NOM_ALUMINIUM,
    Graphic3d_NOM_JADE,Graphic3d_NOM_OBSIDIAN,Graphic3d_NOM_NEON_PHC,
    Graphic3d_NOM_STEEL


# Visualization
display, start_display, add_menu, add_function_to_menu = init_display()

top_color = Quantity_Color(0.27, 0.27, 0.44, Quantity_TOC_RGB)  # Dark
    blue (#444470)
bottom_color = Quantity_Color(0.67, 0.67, 0.67, Quantity_TOC_RGB)  #
    Light gray (#AAAAAA)
```

```
29
30 # Set the gradient background (top-to-bottom)
31 display.View.SetBgGradientColors(top_color, bottom_color,
       Aspect_GFM_VER, True)
32 redd=Quantity_Color(1, 0, 0, Quantity_TOC_RGB)
33
34
35 length = 1000
36 width = 20 # Width of the angle
37 depth = 20  # Height of angle
38 flange_thickness = 4  # Thickness of the flanges
39 web_thickness =4 # Thickness of the web
40 plate_width=100
41 plate_thickness=6
42
43 large_len=200
44 small_len=40
45 trap_height=125
46
47
48 # Define trapezoid corner points (parallel sides along Y-axis)
49 p1 = gp_Pnt(0, 0, 0)    # Bottom-left (short side)
50 p2 = gp_Pnt(0, large_len, 0)  # Top-left (short side)
51 p3 = gp_Pnt(trap_height, 160, 0)  # Top-right (wide side)
52 p4 = gp_Pnt(trap_height, 40, 0)  # Bottom-right (wide side)
53
54 # Create a polygon (wire)
55 polygon = BRepBuilderAPI_MakePolygon()
56 polygon.Add(p1)
57 polygon.Add(p2)
58 polygon.Add(p3)
59 polygon.Add(p4)
60 polygon.Close()
61
62 # Create a face from the polygon
63 face = BRepBuilderAPI_MakeFace(polygon.Wire())
64
65 # Extrude to make a 3D plate (thickness of 5 units)
66
```

```
67 trap_plate1 = BRepPrimAPI_MakePrism(face.Face(), gp_Vec(0, 0,
       plate_thickness)).Shape()
68
69
70 # Define transformation for rotation
71 trsf = gp_Trsf()
72 rotation_axis = gp_Ax1(gp_Pnt(0, 0, 0), gp_Dir(0, 0, 1))  # Z-axis
73 angle = 180 * (3.14159265 / 180)  # Convert degrees to radians
74 trsf.SetRotation(rotation_axis, angle)
75 trap_plate2r = BRepBuilderAPI_Transform(trap_plate1, trsf, True).Shape
       ()
76
77 # transformation aftter rotation plate2
78 trsf = gp_Trsf()
79 trsf.SetTranslation(gp_Vec(length+trap_height,large_len, 0))  # Move to
       the top
80 trap_plate2 = BRepBuilderAPI_Transform(trap_plate2r, trsf, True).Shape
       ()
81
82
83
84
85 angle=create_angle_section(length, width, depth, flange_thickness,
       web_thickness)
86
87 trsf = gp_Trsf()
88 trsf.SetTranslation(gp_Vec(trap_height/2,(large_len-2*width)/2,
       plate_thickness))  # Move to the top
89 angle_transform = BRepBuilderAPI_Transform(angle, trsf, True).Shape()
90
91 """
92 p1=(0,0,0)
93 p2=(0,large_len,0)
94 p3=(trap_height,(large_len-small_len)/2+small_len, 0)
95 p4=(trap_height,(large_len-small_len)/2,0)
96
97
98 trap_plate1=create_trapizoid(p1,p2,p3,p4)
99
```

```
trsf = gp_Trsf()
trsf.SetTranslation(gp_Vec(500, 0, 0))  # Move to the top
trap_plate2= BRepBuilderAPI_Transform(trap_plate1, trsf, True).Shape()
"""


# Define parameters for the fillet weld
b = flange_thickness
h = flange_thickness

# Define multiple origins and directions for placing the fillet welds
origins = [
    numpy.array([trap_height/2,(large_len-2*width)/2, plate_thickness]),
    numpy.array([length,(large_len-2*width)/2, plate_thickness]),
    numpy.array([trap_height/2,(large_len-2*width)/2+2*width, plate_thickness]),
    numpy.array([length,(large_len-2*width)/2+2*width, plate_thickness]),
    numpy.array([trap_height/2,(large_len-2*width)/2, plate_thickness]),
    numpy.array([length+trap_height/2,(large_len-2*width)/2, plate_thickness]),# First placement,  # Second placement
]

#uDir = numpy.array([0.0, 0.0, 1.0])  # Common uDir for all placements
#shaftDir = numpy.array([0.0, 1.0, 0.0])  # Common wDir for all placements

# Create and display multiple fillet welds
for i, origin in enumerate(origins):
  if i<=3:
     shaftDir = numpy.array([1.0, 0.0, 0.0])
     L=trap_height/2
     if i<=1:
         uDir = numpy.array([0.0, 0.0, 1.0])
     else:
         uDir = numpy.array([0.0, 1.0, 0.0])

```

```
  else:
      shaftDir = numpy.array([0.0, 1.0, 0.0])
      L=width*2
      if i==4:
         uDir = numpy.array([-1.0, 0.0, 0.0])
      else:
         uDir = numpy.array([0.0, 0.0, 1.0])


  # Create a fillet weld object
  FWeld = FilletWeld(b, h, L)

  # Place the fillet weld at the specified origin and direction
  FWeld.place(origin, uDir, shaftDir)

  # Create the 3D model (prism) for the fillet weld
  prism = FWeld.create_model()  # Rotate the second weld
  display.DisplayShape(prism,color=redd, update=True)




# Add a message at the origin
Point = gp_Pnt(0.0, 0.0, 0.0)
display.DisplayMessage(Point, "Origin")

# Show the I-section model
#display.DisplayShape(trap_plate1, update=True)
display.DisplayShape(trap_plate2,material=Graphic3d_NOM_ALUMINIUM,
    update=True)
display.DisplayShape(angle_transform, update=True)
display.DisplayShape(trap_plate1,material=Graphic3d_NOM_ALUMINIUM,
    update=True)
display.FitAll()
start_display()


%-------------------- end code ---------------
```

5.4.3 Explanation of the Code

- **Lines 1–15**: Imports all necessary OCC modules and custom classes including `FilletWeld`, `create_angle_section`, and `create_trapizoid`. These enable 3D geometry creation and visualization.
- **Lines 17–22**: Initializes the OCC 3D display environment. Also sets a gradient background using top and bottom RGB values.
- **Lines 24–34**: Defines geometric parameters for the connection, including angle size and gusset plate dimensions.
- **Lines 37–46**: Constructs a 2D trapezoidal profile using four corner points, then forms a wire polygon for extrusion.
- **Lines 48–50**: Creates a planar face from the polygon and extrudes it to form the first 3D gusset plate (`trap_plate1`).
- **Lines 53–58**: Rotates the first plate 180° about the Z-axis and translates it to form the second gusset plate (`trap_plate2`).
- **Lines 61–64**: Generates an angle section using custom function `create_angle_section`, then positions it between the gusset plates.
- **Lines 70–73**: Defines the weld size parameters. Both `b` and `h` are set equal to flange thickness, forming a right triangular weld.
- **Lines 75–81**: Defines a list of 3D origins where fillet welds will be placed. These locations correspond to welds along the gusset and angle interfaces.
- **Lines 84–106**: Loops through each weld position. Based on index, it:
  - Sets the direction of extrusion (`shaftDir`),
  - Sets the face direction (`uDir`),
  - Defines the extrusion length (`L`) either as `trap_height/2` or `width*2`.

  Then creates and places a fillet weld at the correct orientation and position.

- **Line 109**: Adds a label “Origin” to mark the global coordinate system reference point in the 3D viewer.
- **Lines 112–115**: Displays the gusset plates and angle member using specified visual materials. Fits the view and starts the GUI loop to render the final 3D model.

Chapter 6

CAD Integration

6.1 Problem Statement

To integrate the CAD generation script into Osdag.

6.2 Task Done

Made changes in the Osdag main repository for integration of Bolted Butt Joint anf Welded Butt Joint CAD Integration. The changes made are as follow:

6.3 Python Code inside common_logic.py

- `common_logic.py`: Added the CAD generating method inside class CommonDesignLogic(object): for both Bolted Butt Joint and Welded Butt Joint.
- `ui_templete.py`:Added KEY of for both Bolted Butt Joint and Welded Butt Joint in ui_templete.py.
- `Original Code Placement`: Placed the orginal code for generating the Bolted Butt Joint and Welded Butt Joint under the "D:osdag" and "D:osdag" directory respectively.

6.3.1 Python Script

The Python script is shown below. Each section is commented for clarity.

Listing 6.1: Snippet of code for method for Bolted and Welded Butt Joint CAD generation insdie common_logic.py

```
1
2
3  class CommonDesignLogic(object):
4
5      def __init__(self, display, folder, connection, mainmodule):
6
7          self.display = display
8          self.mainmodule = mainmodule
9          self.connection = connection
10         print(self.connection)
11
12
13         self.connectivityObj = None
14         self.folder = folder
15
16     def createButtJointBoltedCAD(self):
17
18             # Get input values from the design object (i.e., instance
                   of ButtJointBolted)
19             Col = self.module_class
20
21             # Extract parameters from the ButtJointBolted object
22             self.plate1_thickness = float(Col.plate1.thickness[0])
23             self.plate2_thickness = float(Col.plate2.thickness[0])
24             self.cover_thickness = float(Col.
                   calculated_cover_plate_thickness)
25             self.plate_width = float(Col.width)
26             self.bolt_dia = float(Col.bolt.bolt_diameter_provided)
27             self.bolt_rows = int(Col.rows)
28             self.bolt_cols = int(Col.cols)
29             self.pitch = float(Col.final_pitch)
30             self.gauge = float(Col.final_gauge)
31             self.edge = float(Col.final_edge_dist)
32             self.end = float(Col.final_end_dist)
33             self.number_bolts = int(Col.number_bolts)
34
```

```
35              butt_joint, plate1, plate2, platec, bolts, nuts =
                    create_bolted_butt_joint(self.plate1_thickness, self.
                    plate2_thickness, self.cover_thickness, self.plate_width
                    , self.bolt_dia,
36                                  self.bolt_rows, self.bolt_cols, self.pitch,
                                        self.gauge, self.edge, self.end, self.
                                    number_bolts)
37              return butt_joint, plate1, plate2, platec, bolts, nuts
38
39      def createWeldedButtJoint(self):
40          Conn = self.module_class
41          self.plate1_thickness = float(Conn.plate1thk)
42          self.plate2_thickness = float(Conn.plate2thk)
43          self.plate_width = float(Conn.width)
44          self.cover_thickness = float(Conn.
                calculated_cover_plate_thickness)
45          self.weld_size = float(Conn.weld_size)
46          self.weld_length = float(Conn.weld_length_provided)
47
48          # Call function to create the welded joint
49          plate1_model, plate2_model, cover_plate_model, welds_models =
                create_welded_butt_joint(
50              plate_width=self.plate_width,
51              plate1_thickness=self.plate1_thickness,
52              plate2_thickness=self.plate2_thickness,
53              cover_thickness=self.cover_thickness,
54              weld_size=self.weld_size,
55              weld_length=self.weld_length
56          )
57          self.nuts_models = []    #         prevents crash for welded
58          self.bolt_models = []    #         optional
59          return plate1_model, plate2_model, cover_plate_model,
                welds_models
```

6.3.2 Explanation of the Code

- **Lines 1–2**: Define the method `createButtJointBoltedCAD`, intended to create a 3D model of a bolted butt joint.

- **Line 3**: Retrieves an instance of the data class `ButtJointBolted` through `self.module_class`.
- **Lines 5–14**: Extracts all required geometric and design parameters such as plate thicknesses, bolt size, bolt layout (rows/columns), and spacing details (pitch, gauge, edge, end distance) from the input object.
- **Line 16**: Calls the function `create_bolted_butt_joint` with extracted values, which returns CAD models of the full joint including plates, bolts, and nuts.
- **Line 17**: Returns all generated 3D shapes to be used for visualization or further processing.
- **Line 19**: Begins the definition of `createWeldedButtJoint`, used for modeling a welded butt joint variation.
- **Line 20**: Retrieves an instance of the corresponding input object (`self.module_class`).
- **Lines 21–25**: Extracts relevant dimensions such as plate thicknesses, plate width, weld size, and weld length.
- **Lines 27–34**: Passes the extracted values into `create_welded_butt_joint`, which generates models for two plates, a cover plate, and fillet welds.
- **Lines 35–36**: Initializes empty lists for bolts and nuts to prevent runtime errors—since welded joints do not contain these components.
- **Line 37**: Returns the generated models for the welded configuration, including the plates and welds.

Listing 6.2: Snippet of code for method for Bolted and Welded Butt Joint CAD generation insdie common_logic.py

```
1
2
3       elif self.mainmodule == 'Butt Joint Bolted Connection':
4            self.col = self.module_class()
5            self.assembly,self.plate1_model,self.plate2_model,self.
                platec_model,self.bolt_models,self.nuts_models = self.
                createButtJointBoltedCAD()
6
```

```
7             if self.component == "Model":
8                 osdag_display_shape(self.display, self.plate1_model,
                      update=True, material=Graphic3d_NOM_ALUMINIUM)
9                 osdag_display_shape(self.display, self.plate2_model,
                      update=True)
10                osdag_display_shape(self.display, self.platec_model,
                      update=True)
11                if hasattr(self, 'bolt_models'):
12                    for bolt in self.bolt_models:
13                        osdag_display_shape(self.display, bolt, update=
                              True, color=Quantity_NOC_SADDLEBROWN)
14
15                    if hasattr(self, 'nuts_models'):
16                        for nut in self.nuts_models:
17                            osdag_display_shape(self.display, nut,
                                  update=True, color=
                                  Quantity_NOC_SADDLEBROWN)
18
19        elif self.mainmodule == 'Butt Joint Welded Connection':
20            self.col = self.module_class()
21            self.plate1_model, self.plate2_model, self.
                  cover_plate_model, self.welds_models = self.
                  createWeldedButtJoint()
22
23            if self.component == "Model":
24                osdag_display_shape(self.display, self.plate1_model,
                      update=True, material=Graphic3d_NOM_ALUMINIUM)
25                osdag_display_shape(self.display, self.plate2_model,
                      update=True)
26                osdag_display_shape(self.display, self.
                      cover_plate_model, update=True)
27                if hasattr(self, 'welds_models'):
28                    for weld in self.welds_models:
29                        osdag_display_shape(self.display, weld, update=
                              True, color=Quantity_NOC_SADDLEBROWN)
30                for nut in self.nuts_models:
31                    osdag_display_shape(self.display, nut, update=True,
                          color=Quantity_NOC_SADDLEBROWN)
```

6.3.3 Full code

Listing 6.3: Snippet of code for method for KEY specification for Bolted and Weled Butt Joint inside ui_templete,py

```
1
2       def display_3DModel(self, component, bgcolor):
3
4           self.component = component
5
6           self.display.EraseAll()
7
8           self.display.View_Iso()
9
10          self.display.FitAll()
11
12          self.display.DisableAntiAliasing()
13
14          if bgcolor == "gradient_bg":
15
16              self.display.set_bg_gradient_color([51, 51, 102], [150,
                    150, 170])
17          else:
18              self.display.set_bg_gradient_color([255, 255, 255], [255,
                    255, 255])
19
20          if self.mainmodule  == "Shear Connection":
21
22              A = self.module_class()
23
24              self.loc = A.connectivity
25
26
27              if self.loc == "Column Flange-Beam Web" and self.connection
                    == KEY_DISP_FINPLATE:
28                  # pass
29                  # print("hghghghg")
30                  self.display.View.SetProj(OCC.Core.V3d.V3d_XnegYnegZpos
                        )
```

```
31              elif self.loc == "Column Flange-Beam Web" and self.
                    connection == KEY_DISP_SEATED_ANGLE:
32                  self.display.View.SetProj(OCC.Core.V3d.V3d_XnegYnegZpos
                        )
33              elif self.loc == "Column Flange-Beam Web" and self.
                    connection == KEY_DISP_SEATED_ANGLE:
34                  self.display.View.SetProj(OCC.Core.V3d.V3d_XposYnegZpos
                        )
35
36              if self.component == "Column":
37                  osdag_display_shape(self.display, self.connectivityObj.
                        get_columnModel(), update=True)
38              elif self.component == "Beam":
39                  osdag_display_shape(self.display, self.connectivityObj.
                        get_beamModel(), material=Graphic3d_NOM_ALUMINIUM,
40                                      update=True)
41              elif component == "cleatAngle":
42
43                  osdag_display_shape(self.display, self.connectivityObj.
                        angleModel, color=Quantity_NOC_BLUE1, update=True)
44                  osdag_display_shape(self.display, self.connectivityObj.
                        angleLeftModel, color=Quantity_NOC_BLUE1,
45                                      update=True)
46                  nutboltlist = self.connectivityObj.nut_bolt_array.
                        get_models()
47                  for nutbolt in nutboltlist:
48                      osdag_display_shape(self.display, nutbolt, color=
                            Quantity_NOC_SADDLEBROWN, update=True)
49
50              elif component == "SeatAngle":
51                  osdag_display_shape(self.display, self.connectivityObj.
                        topclipangleModel, color=Quantity_NOC_BLUE1,
52                                      update=True)
53                  osdag_display_shape(self.display, self.connectivityObj.
                        angleModel, color=Quantity_NOC_BLUE1, update=True)
54                  nutboltlist = self.connectivityObj.nut_bolt_array.
                        get_models()
55                  for nutbolt in nutboltlist:
```

```
56                      osdag_display_shape(self.display, nutbolt, color=
                            Quantity_NOC_SADDLEBROWN, update=True)
57
58              elif self.component == "Plate":
59                  osdag_display_shape(self.display, self.connectivityObj.
                        weldModelLeft, color=Quantity_NOC_RED, update=True)
60                  osdag_display_shape(self.display, self.connectivityObj.
                        weldModelRight, color=Quantity_NOC_RED, update=True)
61                  osdag_display_shape(self.display, self.connectivityObj.
                        plateModel, color=Quantity_NOC_BLUE4, update=True)
62                  nutboltlist = self.connectivityObj.nut_bolt_array.
                        get_models()
63                  for nutbolt in nutboltlist:
64                      osdag_display_shape(self.display, nutbolt, color=
                            Quantity_NOC_SADDLEBROWN, update=True)
65
66              elif self.component == "Model":
67
68                  osdag_display_shape(self.display, self.connectivityObj.
                        columnModel, update=True)
69                  osdag_display_shape(self.display, self.connectivityObj.
                        beamModel, material=Graphic3d_NOM_ALUMINIUM,
70                                      update=True)
71                  if self.connection == KEY_DISP_FINPLATE or self.
                        connection == KEY_DISP_ENDPLATE:
72                      osdag_display_shape(self.display, self.
                            connectivityObj.weldModelLeft, color=
                            Quantity_NOC_RED, update=True)
73                      osdag_display_shape(self.display, self.
                            connectivityObj.weldModelRight, color=
                            Quantity_NOC_RED, update=True)
74                      osdag_display_shape(self.display, self.
                            connectivityObj.plateModel, color=
                            Quantity_NOC_BLUE1,
75                                          update=True)
76
77                  elif self.connection == KEY_DISP_CLEATANGLE:
78                      osdag_display_shape(self.display, self.
                            connectivityObj.angleModel, color=
```

```
                        Quantity_NOC_BLUE1,
79                                      update=True)
80                  osdag_display_shape(self.display, self.
                        connectivityObj.angleLeftModel, color=
                        Quantity_NOC_BLUE1,
81                                      update=True)
82              else:
83                  osdag_display_shape(self.display, self.
                        connectivityObj.topclipangleModel, color=
                        Quantity_NOC_BLUE1,
84                                      update=True)
85                  osdag_display_shape(self.display, self.
                        connectivityObj.angleModel, color=
                        Quantity_NOC_BLUE1,
86                                      update=True)
87              nutboltlist = self.connectivityObj.nut_bolt_array.
                    get_models()
88              for nutbolt in nutboltlist:
89                  osdag_display_shape(self.display, nutbolt, color=
                        Quantity_NOC_SADDLEBROWN, update=True)
90
91
92
93
94      if self.mainmodule == "Moment Connection":
95          if self.connection == KEY_DISP_BEAMCOVERPLATE:
96
97              self.B = self.module_class()
98              # else:
99              #     pass
100             #
101             # self.loc = A.connectivity
102             self.CPObj = self.createBBCoverPlateCAD()  #
                    CPBoltedObj is an object which gets all the
                    calculated values of CAD models
103             if self.component == "Beam":
104                 # Displays both beams
105                 osdag_display_shape(self.display, self.CPObj.
                        get_only_beams_Models(), update=True)
```

```
106
107                 elif self.component == "Connector":
108                     osdag_display_shape(self.display, self.CPObj.
                            get_flangewebplatesModel(), update=True,
109                                         color=Quantity_NOC_BLUE1)
110                     if self.B.preference != 'Outside':
111                         osdag_display_shape(self.display, self.CPObj.
                                get_innetplatesModels(), update=True,
112                                             color=Quantity_NOC_BLUE1)
113
114                     osdag_display_shape(self.display, self.CPObj.
                            get_nut_bolt_arrayModels(), update=True,
115                                         color=Quantity_NOC_YELLOW)
116
117                 elif self.component == "Model":
118                     osdag_display_shape(self.display, self.CPObj.
                            get_beamsModel(), update=True)
119                     osdag_display_shape(self.display, self.CPObj.
                            get_flangewebplatesModel(), update=True,
120                                         color=Quantity_NOC_BLUE1)
121
122                     # Todo: remove velove commented lines
123
124                     if self.B.preference != 'Outside':
125                         osdag_display_shape(self.display, self.CPObj.
                                get_innetplatesModels(), update=True,
126                                             color=Quantity_NOC_BLUE1)
127
128                     osdag_display_shape(self.display, self.CPObj.
                            get_nut_bolt_arrayModels(), update=True,
129                                         color=Quantity_NOC_YELLOW)
130             elif self.connection == KEY_DISP_BB_EP_SPLICE:
131                 self.B = self.module_class()
132
133                 self.ExtObj = self.createBBEndPlateCAD()
134
135                 if component == "Beam":
136                     osdag_display_shape(self.display, self.ExtObj.
                            get_beam_models(), update=True)
```

```
137
138                 elif component == "Connector":
139                     osdag_display_shape(self.display, self.ExtObj.
                            get_plate_connector_models(), update=True,
140                                         color='Blue')
141                     osdag_display_shape(self.display, self.ExtObj.
                            get_welded_models(), update=True, color='Red')
142                     osdag_display_shape(self.display, self.ExtObj.
                            get_nut_bolt_array_models(), update=True,
143                                         color=Quantity_NOC_SADDLEBROWN)
144
145                 elif component == "Model":
146
147                     # osdag_display_shape(self.display, self.ExtObj.
                            get_models(), update=True)
148                     osdag_display_shape(self.display, self.ExtObj.
                            get_beam_models(), update=True)
149                     osdag_display_shape(self.display, self.ExtObj.
                            get_plate_connector_models(), update=True,
150                                         color='Blue')
151                     osdag_display_shape(self.display, self.ExtObj.
                            get_welded_models(), update=True, color='Red')
152                     osdag_display_shape(self.display, self.ExtObj.
                            get_nut_bolt_array_models(), update=True,
153                                         color=Quantity_NOC_SADDLEBROWN)
154
155
156
157             elif self.connection == KEY_DISP_BEAMCOVERPLATEWELD:
158                 self.B = self.module_class()
159                 self.CPObj = self.createBBCoverPlateCAD()
160                 beams = self.CPObj.get_beam_models()
161                 plates = self.CPObj.get_plate_models()
162                 welds = self.CPObj.get_welded_modules()
163
164                 if self.component == "Beam":
165                     # Displays both beams
166                     osdag_display_shape(self.display, beams, update=
                            True)
```

```
167                 elif self.component == "Connector":
168                     osdag_display_shape(self.display, plates, update=
                            True, color=Quantity_NOC_BLUE1)
169                     osdag_display_shape(self.display, welds, update=
                            True, color=Quantity_NOC_RED)
170                 elif self.component == "Model":
171                     osdag_display_shape(self.display, beams, update=
                            True)
172                     osdag_display_shape(self.display, plates, update=
                            True, color=Quantity_NOC_BLUE1)
173                     osdag_display_shape(self.display, welds, update=
                            True, color=Quantity_NOC_RED)
174
175             elif self.connection == KEY_DISP_COLUMNCOVERPLATE:
176                 self.C = self.module_class()
177                 self.CPObj = self.createCCCoverPlateCAD()
178                 columns = self.CPObj.get_column_models()
179                 plates = self.CPObj.get_plate_models()
180                 nutbolt = self.CPObj.get_nut_bolt_models()
181                 onlycolumn = self.CPObj.get_only_column_models()
182
183                 if self.component == "Column":
184                     # Displays both beams
185                     osdag_display_shape(self.display, onlycolumn,
                            update=True)
186                 elif self.component == "Cover Plate":
187                     osdag_display_shape(self.display, plates, update=
                            True, color=Quantity_NOC_BLUE1)
188                     osdag_display_shape(self.display, nutbolt, update=
                            True, color=Quantity_NOC_YELLOW)
189                 elif self.component == "Model":
190                     osdag_display_shape(self.display, columns, update=
                            True)
191                     osdag_display_shape(self.display, plates, update=
                            True, color=Quantity_NOC_BLUE1)
192                     osdag_display_shape(self.display, nutbolt, update=
                            True, color=Quantity_NOC_YELLOW)
193
194
```

```
195             elif self.connection == KEY_DISP_BCENDPLATE:
196                 self.Bc = self.module_class()
197                 self.ExtObj = self.createBCEndPlateCAD()
198 
199                 self.display.View.SetProj(OCC.Core.V3d.V3d_XnegYnegZpos
                        )
200                 c_length = self.column_length
201                 # Point1 = gp_Pnt(0.0, 0.0, c_length)
202                 # DisplayMsg(self.display, Point1, self.Bc.
                        supporting_section.designation)
203                 b_length = self.beam_length + self.Bc.
                        supporting_section.depth/2+100
204                 # Point2 = gp_Pnt(0.0,-b_length, c_length/2)
205                 # DisplayMsg(self.display, Point2, self.Bc.
                        supported_section.designation)
206                 # Displays the beams #TODO ANAND
207                 if component == "Column":
208                     self.display.View_Iso()
209                     osdag_display_shape(self.display, self.ExtObj.
                            columnModel, update=True)
210                     # Point1 = gp_Pnt(-self.Bc.supporting_section.
                            flange_width/2, 0, c_length)
211                     # DisplayMsg(self.display, Point1, self.Bc.
                            supporting_section.designation)
212                     # Point = gp_Pnt(0.0, 0.0, 10)
213                     # DisplayMsg(self.display,Point, "Column")
214 
215                 elif component == "Beam":
216                     self.display.View_Iso()
217                     osdag_display_shape(self.display, self.ExtObj.
                            beamModel, update=True,
218                                         material=
                                                Graphic3d_NOM_ALUMINIUM)
219                     # Point2 = gp_Pnt(0.0, -b_length, c_length / 2)
220                     # DisplayMsg(self.display, Point2, self.Bc.
                            supported_section.designation)
221                     # , color = 'Dark Gray'
222 
223                 elif component == "Connector":
```

```
224                     osdag_display_shape(self.display, self.ExtObj.
                            get_plate_connector_models(), update=True,
225                                         color='Blue')
226                     osdag_display_shape(self.display, self.ExtObj.
                            get_welded_models(), update=True, color='Red')
227                     osdag_display_shape(self.display, self.ExtObj.
                            get_nut_bolt_array_models(), update=True,
228                                         color=Quantity_NOC_SADDLEBROWN)
229
230
231                 elif component == "Model":
232
233                     osdag_display_shape(self.display, self.ExtObj.
                            get_column_models(), update=True)
234                     osdag_display_shape(self.display, self.ExtObj.
                            get_beam_models(), update=True,
235                                         material=
                                                Graphic3d_NOM_ALUMINIUM)
236                     osdag_display_shape(self.display, self.ExtObj.
                            get_plate_connector_models(), update=True,
237                                         color='Blue')
238                     osdag_display_shape(self.display, self.ExtObj.
                            get_welded_models(), update=True, color='Red')
239                     osdag_display_shape(self.display, self.ExtObj.
                            get_nut_bolt_array_models(), update=True,
240                                         color=Quantity_NOC_SADDLEBROWN)
241                     # Point1 = gp_Pnt(self.Bc.supporting_section.
                            flange_width/2, -self.Bc.supporting_section.
                            depth/2, c_length*0.75)
242                     # DisplayMsg(self.display, Point1, self.Bc.
                            supporting_section.designation)
243                     # Point2 = gp_Pnt(self.Bc.supporting_section.
                            flange_width/2, -b_length, c_length / 2)
244                     # DisplayMsg(self.display, Point2, self.Bc.
                            supported_section.designation)
245                     # Erase(DisplayMsg(self.display, Point2, self.Bc.
                            supported_section.designation))
246
247
```

```
248             elif self.connection == KEY_DISP_COLUMNCOVERPLATEWELD:
249                 self.C = self.module_class()
250                 self.CPObj = self.createCCCoverPlateCAD()
251                 columns = self.CPObj.get_column_models()
252                 plates = self.CPObj.get_plate_models()
253                 welds = self.CPObj.get_welded_modules()
254 
255                 if self.component == "Column":
256                     # Displays both beams
257                     osdag_display_shape(self.display, columns, update=
                            True)
258                 elif self.component == "Cover Plate":
259                     osdag_display_shape(self.display, plates, update=
                            True, color=Quantity_NOC_BLUE1)
260                     osdag_display_shape(self.display, welds, update=
                            True, color=Quantity_NOC_RED)
261                 elif self.component == "Model":
262                     osdag_display_shape(self.display, columns, update=
                            True)
263                     osdag_display_shape(self.display, plates, update=
                            True, color=Quantity_NOC_BLUE1)
264                     osdag_display_shape(self.display, welds, update=
                            True, color=Quantity_NOC_RED)
265 
266             elif self.connection == KEY_DISP_COLUMNENDPLATE:
267                 self.CEP = self.module_class()
268                 self.CEPObj = self.createCCEndPlateCAD()
269                 columns = self.CEPObj.get_column_models()
270                 plates = self.CEPObj.get_plate_models()
271                 welds = self.CEPObj.get_weld_models()
272                 nutBolts = self.CEPObj.get_nut_bolt_models()
273 
274                 if self.component == "Column":
275                     osdag_display_shape(self.display, columns, update=
                            True)
276 
277                 elif self.component == "Connector":
278                     osdag_display_shape(self.display, plates, update=
                            True, color=Quantity_NOC_BLUE1)
```

```
279                     osdag_display_shape(self.display, welds, update=
                            True, color=Quantity_NOC_RED)
280                     osdag_display_shape(self.display, nutBolts, update=
                            True, color=Quantity_NOC_YELLOW)
281
282                 elif self.component == "Model":
283                     osdag_display_shape(self.display, columns, update=
                            True)
284                     osdag_display_shape(self.display, plates, update=
                            True, color=Quantity_NOC_BLUE1)
285                     osdag_display_shape(self.display, welds, update=
                            True, color=Quantity_NOC_RED)
286                     osdag_display_shape(self.display, nutBolts, update=
                            True, color=Quantity_NOC_YELLOW)
287
288             elif self.connection == KEY_DISP_BASE_PLATE:
289                 self.Bp = self.module_class
290
291                 self.BPObj = self.createBasePlateCAD()
292
293                 column = self.BPObj.get_column_model()
294                 plate = self.BPObj.get_plate_connector_models()
295                 weld = self.BPObj.get_welded_models()
296                 nut_bolt = self.BPObj.get_nut_bolt_array_models()
297                 conc = self.BPObj.get_concrete_models()
298                 grout = self.BPObj.get_grout_models()
299
300                 if self.component == "Model":  # Todo: change this into
                        key
301                     osdag_display_shape(self.display, column, update=
                            True)
302                     osdag_display_shape(self.display, plate, color=
                            Quantity_NOC_BLUE1, update=True)
303                     osdag_display_shape(self.display, weld, color=
                            Quantity_NOC_RED, update=True)
304                     osdag_display_shape(self.display, nut_bolt, color=
                            Quantity_NOC_YELLOW, update=True)
305                     osdag_display_shape(self.display, conc, color=GRAY,
                             transparency=0.5, update=True)
```

```
306                     osdag_display_shape(self.display, grout, color=GRAY
                            , transparency=0.5, update=True)
307
308                 elif self.component == "Column":
309                     osdag_display_shape(self.display, column, update=
                            True)
310
311                 elif self.component == "Connector":
312                     osdag_display_shape(self.display, plate, color=
                            Quantity_NOC_BLUE1, update=True)
313                     osdag_display_shape(self.display, weld, color=
                            Quantity_NOC_RED, update=True)
314                     osdag_display_shape(self.display, nut_bolt, color=
                            Quantity_NOC_YELLOW, update=True)
315
316         elif self.mainmodule == 'Columns with known support conditions'
                :
317             self.col = self.module_class()
318             self.ColObj = self.createColumnInFrameCAD()
319
320             if self.component == "Model":
321                 osdag_display_shape(self.display, self.ColObj, update=
                        True)
322
323         elif self.mainmodule == 'Lap Joint Bolted Connection':
324             self.col = self.module_class()
325             self.assembly,self.plate1_model,self.plate2_model,self.
                    bolt_models,self.nuts_models = self.createBoltedLapJoint
                    ()
326
327             if self.component == "Model":
328                 osdag_display_shape(self.display, self.plate1_model,
                        update=True, material=Graphic3d_NOM_ALUMINIUM)
329                 osdag_display_shape(self.display, self.plate2_model,
                        update=True)
330                 if hasattr(self, 'bolt_models'):
331                     for bolt in self.bolt_models:
332                         osdag_display_shape(self.display, bolt, update=
                                True, color=Quantity_NOC_SADDLEBROWN)
```

```
333
334                     if hasattr(self, 'nuts_models'):
335                         for nut in self.nuts_models:
336                             osdag_display_shape(self.display, nut,
                                    update=True, color=
                                    Quantity_NOC_SADDLEBROWN)
337
338         elif self.mainmodule == 'Butt Joint Bolted Connection':
339             self.col = self.module_class()
340             self.assembly,self.plate1_model,self.plate2_model,self.
                platec_model,self.bolt_models,self.nuts_models = self.
                createButtJointBoltedCAD()
341
342             if self.component == "Model":
343                 osdag_display_shape(self.display, self.plate1_model,
                    update=True, material=Graphic3d_NOM_ALUMINIUM)
344                 osdag_display_shape(self.display, self.plate2_model,
                    update=True)
345                 osdag_display_shape(self.display, self.platec_model,
                    update=True)
346                 if hasattr(self, 'bolt_models'):
347                     for bolt in self.bolt_models:
348                         osdag_display_shape(self.display, bolt, update=
                            True, color=Quantity_NOC_SADDLEBROWN)
349
350                     if hasattr(self, 'nuts_models'):
351                         for nut in self.nuts_models:
352                             osdag_display_shape(self.display, nut,
                                update=True, color=
                                Quantity_NOC_SADDLEBROWN)
353
354         elif self.mainmodule == 'Butt Joint Welded Connection':
355             self.col = self.module_class()
356             self.plate1_model, self.plate2_model, self.
                cover_plate_model, self.welds_models = self.
                createWeldedButtJoint()
357
358             if self.component == "Model":
```

```
359                 osdag_display_shape(self.display, self.plate1_model,
                        update=True, material=Graphic3d_NOM_ALUMINIUM)
360                 osdag_display_shape(self.display, self.plate2_model,
                        update=True)
361                 osdag_display_shape(self.display, self.
                        cover_plate_model, update=True)
362                 if hasattr(self, 'welds_models'):
363                     for weld in self.welds_models:
364                         osdag_display_shape(self.display, weld, update=
                            True, color=Quantity_NOC_SADDLEBROWN)
365                 for nut in self.nuts_models:
366                     osdag_display_shape(self.display, nut, update=True,
                        color=Quantity_NOC_SADDLEBROWN)
367
368         elif self.mainmodule == 'Flexure Member':
369             self.flex = self.module_class()
370             self.FObj = self.createSimplySupportedBeam()
371
372             if self.component == "Model":
373                 osdag_display_shape(self.display, self.FObj, update=
                        True)
374
375         elif self.mainmodule == 'Flexural Members - Cantilever':
376             self.flex = self.module_class()
377             self.FObj = self.createCantileverBeam()
378
379             if self.component == "Model":
380                 osdag_display_shape(self.display, self.FObj, update=
                        True)
381
382         elif self.mainmodule == 'Struts in Trusses':
383             self.col = self.module_class()
384             self.ColObj = self.createStrutsInTrusses()
385
386             if self.component == "Model":
387                 osdag_display_shape(self.display, self.ColObj, update=
                        True)
388
389         else:
```

```
390             if self.connection == KEY_DISP_TENSION_BOLTED:
391                 self.T = self.module_class()
392                 self.TObj = self.createTensionCAD()
393 
394                 member = self.TObj.get_members_models()
395                 plate = self.TObj.get_plates_models()
396 
397                 nutbolt = self.TObj.get_nut_bolt_array_models()
398 
399                 onlymember = self.TObj.get_only_members_models()
400                 # distance = self.T.length/2 - (2* self.T.plate.
                        end_dist_provided + (self.T.plate.bolt_line - 1 ) *
                        self.T.plate.pitch_provided)
401                 # Point = gp_Pnt(distance, 0.0, 300)
402                 # DisplayMsg(self.display, Point, self.T.section_size_1
                        .designation)
403 
404 
405                 if self.component == "Member":  # Todo: change this
                        into key
406                     osdag_display_shape(self.display, onlymember,
                            update=True)
407                 elif self.component == "Plate":
408                     osdag_display_shape(self.display, plate, color=
                            Quantity_NOC_BLUE1, update=True)
409                     osdag_display_shape(self.display, nutbolt, color=
                            Quantity_NOC_YELLOW, update=True)
410                 elif self.component == "Endplate":
411                     endplate = self.TObj.get_end_plates_models()
412                     end_nutbolt = self.TObj.
                            get_end_nut_bolt_array_models()
413                     osdag_display_shape(self.display, endplate, color=
                            Quantity_NOC_BLUE1, update=True)
414                     osdag_display_shape(self.display, end_nutbolt,
                            color=Quantity_NOC_YELLOW, update=True)
415                 else:
416                     connector = BRepAlgoAPI_Fuse(nutbolt, plate).Shape
                            ()
417                     shape = BRepAlgoAPI_Fuse(connector, member).Shape()
```

```
418                     self.TObj.shape = shape
419                     osdag_display_shape(self.display, member, update=
                            True)
420                     osdag_display_shape(self.display, plate, color=
                            Quantity_NOC_BLUE1, update=True)
421                     osdag_display_shape(self.display, nutbolt, color=
                            Quantity_NOC_YELLOW, update=True)
422
423
424             elif self.connection == KEY_DISP_TENSION_WELDED:
425                 self.T = self.module_class()
426                 self.TObj = self.createTensionCAD()
427
428                 member = self.TObj.get_members_models()
429                 plate = self.TObj.get_plates_models()
430                 welds = self.TObj.get_welded_models()
431                 if self.component == "Member":  # Todo: change this
                        into key
432                     osdag_display_shape(self.display, member, update=
                            True)
433                 elif self.component == "Plate":
434                     osdag_display_shape(self.display, plate, color=
                            Quantity_NOC_BLUE1, update=True)
435                     osdag_display_shape(self.display, welds, color=
                            Quantity_NOC_RED, update=True)
436                 elif self.component == "Endplate":
437                     endplate = self.TObj.get_end_plates_models()
438                     osdag_display_shape(self.display, endplate, color=
                            Quantity_NOC_BLUE1, update=True)
439                 else:
440                     connector = BRepAlgoAPI_Fuse(welds, plate).Shape()
441                     shape = BRepAlgoAPI_Fuse(connector, member).Shape()
442                     self.TObj.shape = shape
443                     osdag_display_shape(self.display, member, update=
                            True)
444                     osdag_display_shape(self.display, plate, color=
                            Quantity_NOC_BLUE1, update=True)
```

Listing 6.4: Snippet of code for method for KEY specification for Bolted and Weled Butt Joint inside ui_templete,py

```
1
2  (Line 1843- 1889)
3      def return_class(self,name):
4
5          if name == KEY_DISP_FINPLATE:
6              return FinPlateConnection
7          elif name == KEY_DISP_ENDPLATE:
8          elif name == KEY_DISP_BUTTJOINTBOLTED:
9              return ButtJointBolted
10         elif name == KEY_DISP_BUTTJOINTWELDED:
11             return ButtJointWelded
12 \subsection{Output of the CAD Integration:}
```

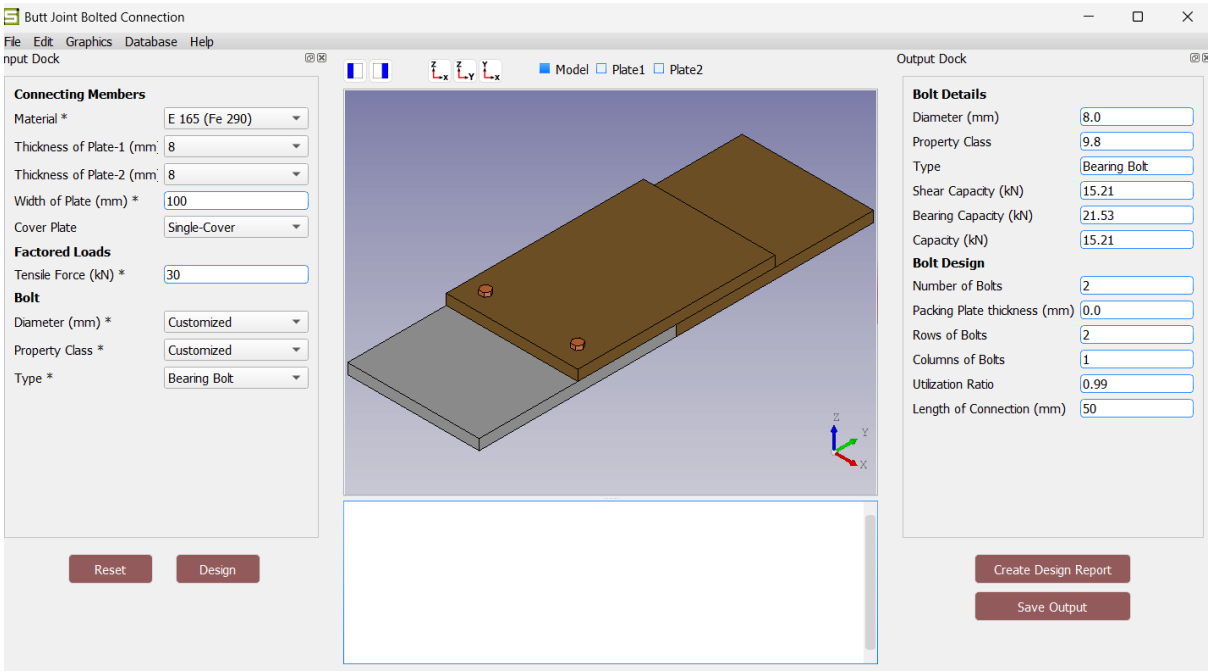

Figure 6.1: CAD Integration of Bolted Butt Joint Connection

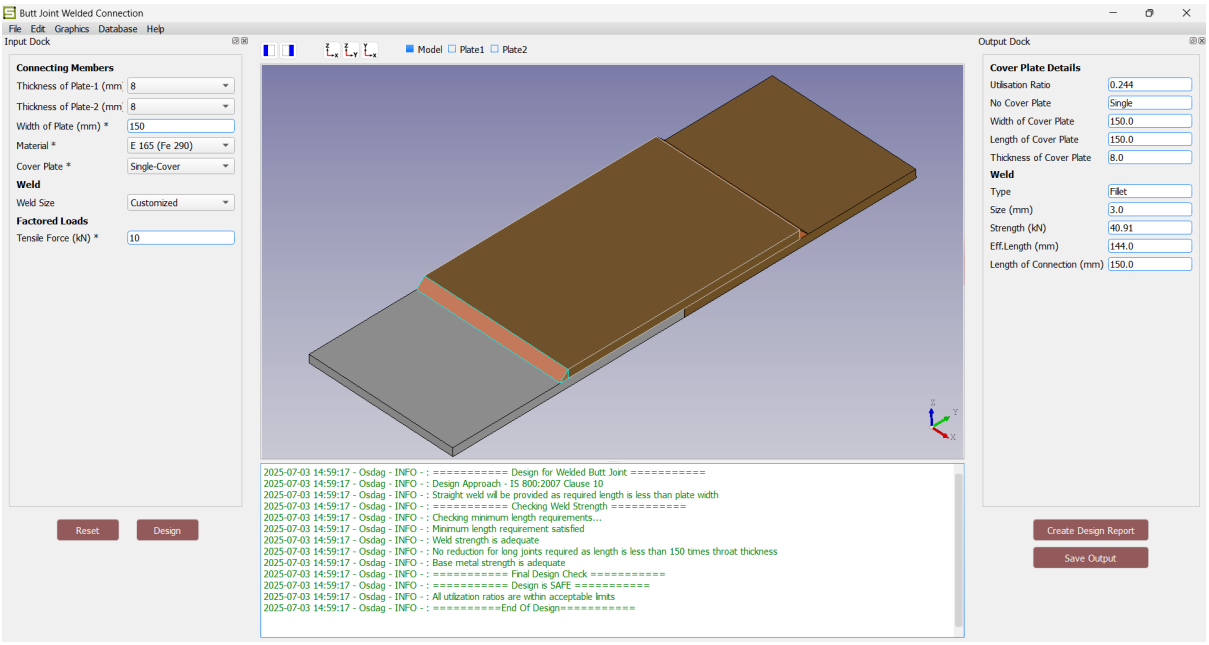

Figure 6.2: CAD Integration of Welded Butt Joint Connection

Chapter 7

Gantry Girder

7.1 Problem Statement

To write a python script to generate the CAD for Gantry Girder beam.

7.2 Task Done

Gantry Girder

A gantry girder is a horizontal steel beam that supports an overhead crane running on rails, typically used in industrial buildings, workshops, and warehouses. It is designed to carry the loads from the crane, including the weight of the crane itself and any materials being lifted.

In this task, I created a Python script that generates and displays the CAD of Gantry Girder beam where there is the I section beam under the C channel beam.

7.3 Python Code

7.3.1 Description of the Script

The script is structured as follows:

- **Input Parameters**: The user defines dimensions such as the total length of the girder, I-section parameters (width, depth, flange thickness, and web thickness), and

C-section parameters (width, depth, flange thickness, and web thickness). These inputs are used to construct the composite gantry girder geometry.

- **Geometry Construction**: The script generates the I-section using a custom `create_i_section` function and the C-section using `create_c_section`. The C-section is then rotated 90° about the X-axis to simulate its vertical orientation in real structures. A translation transformation is applied to correctly position the C-section on top of the I-section.

- **Output**: The final output is a 3D CAD model of a gantry girder visualized using the Open Cascade (OCC) viewer. It includes the base I-section and a vertically oriented C-section mounted above it. The components are colored with different materials (aluminum and copper) and a gradient background enhances visual clarity.

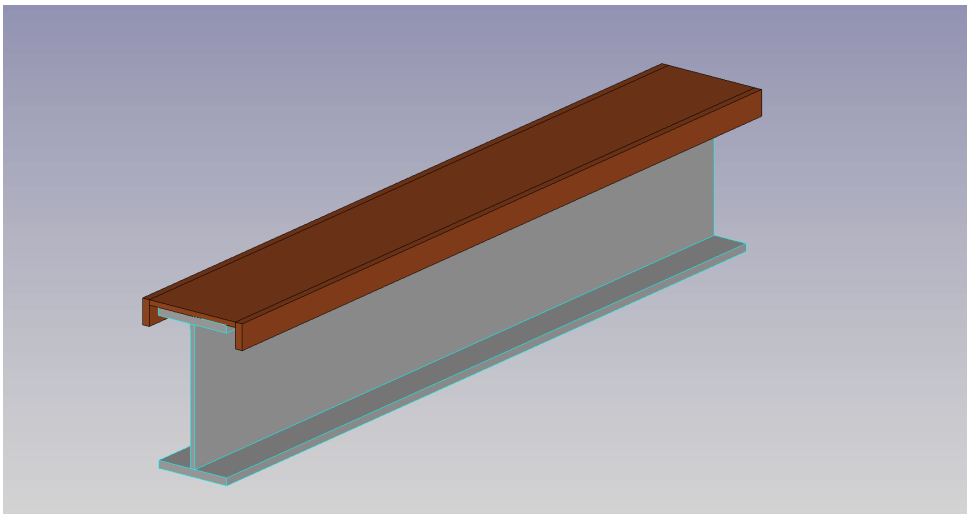

Figure 7.1: Gantry Girder Beam

7.3.2 Python Script

The Python script is shown below. Each section is commented for clarity.

Listing 7.1: Gantry Girder Beam

```
%--------------------begin code-------------

from OCC.Core.gp import gp_Pnt, gp_Dir, gp_Vec, gp_Trsf, gp_Ax1
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_Transform
```

```
7  from OCC.Display.SimpleGui import init_display
8  from OCC.Core.Quantity import Quantity_Color, Quantity_TOC_RGB
9  from OCC.Core.Quantity import Quantity_Color, Quantity_TOC_RGB
10 from OCC.Core.Aspect import Aspect_GFM_VER  # Import the correct
       gradient mode
11 from draw_i_section import create_i_section
12 from draw_c_section import create_c_section
13
14 def create_gantry_girder(length,
15                          i_flange_thickness, i_web_thickness, i_width,
                               i_depth,
16                          c_flange_thickness, c_web_thickness, c_width,
                               c_depth):
17
18     # Create I-section
19     i_section = create_i_section(length, i_width, i_depth,
           i_flange_thickness, i_web_thickness)
20
21     # Create C-section
22     c_section = create_c_section(length, c_width, c_depth,
           c_flange_thickness, c_web_thickness)
23
24     # Rotate the C-section 90 degrees about its length (X-axis)
25     rotation_axis = gp_Ax1(gp_Pnt(0, 0, 0), gp_Dir(1, 0, 0))  #
           Corrected: using gp_Pnt and gp_Dir
26     rotation = gp_Trsf()
27     rotation.SetRotation(rotation_axis, 3.14159/2)  # 90 degrees in
           radians (pi/2)
28     rotated_c_section = BRepBuilderAPI_Transform(c_section, rotation,
           True).Shape()
29
30     # Position the rotated C-section on top of the I-section
31
32     trsf = gp_Trsf()
33     trsf.SetTranslation(gp_Vec(0, i_width/2+c_depth/2, ( i_depth-
           c_width+c_web_thickness)))
34     transformed_c_section = BRepBuilderAPI_Transform(rotated_c_section,
            trsf, True).Shape()
35
```

```
    # Combine the sections
    girder = BRepAlgoAPI_Fuse(i_section, transformed_c_section).Shape()

    return girder

if __name__ == "__main__":
    length = 1500.0

    # I-section parameters
    i_width = 150.0
    i_depth = 300.0
    i_flange_thickness = 15.0
    i_web_thickness = 10.0

    # C-section parameters
    c_width = 50.0
    c_depth = 220.0
    c_flange_thickness = 15.0
    c_web_thickness = 10.0

    gantry_girder = create_gantry_girder(length,
                                         i_flange_thickness,
                                            i_web_thickness, i_width,
                                            i_depth,
                                         c_flange_thickness,
                                            c_web_thickness, c_width,
                                            c_depth)

    # Visualization
    display, start_display, add_menu, add_function_to_menu =
        init_display()

    top_color = Quantity_Color(0.27, 0.27, 0.44, Quantity_TOC_RGB)  #
        Dark blue (#444470)
    bottom_color = Quantity_Color(0.67, 0.67, 0.67, Quantity_TOC_RGB)
        # Light gray (#AAAAAA)
    # Set the gradient background (top-to-bottom)
    display.View.SetBgGradientColors(top_color, bottom_color,
        Aspect_GFM_VER, True)
```

```
67      redd=Quantity_Color(1, 0, 0, Quantity_TOC_RGB)
68
69      # Show the gantry girder model
70      display.DisplayShape(gantry_girder, update=True)
71      display.FitAll()
72      start_display()
73  %-------------------- end code ---------------
```

7.3.3 Explanation of the Code

- **Lines 1–8**: Imports essential Open Cascade (OCC) geometry, transformation, and visualization modules, along with custom functions `create_i_section` and `create_c_section` to build I and C cross-sections.
- **Lines 10–12**: Defines the total length of the gantry girder as `1500 mm`.
- **Lines 14–18**: Sets the geometric parameters for the I-section, including flange and web dimensions.
- **Lines 20–24**: Defines the dimensions of the C-section, which will act as a rail for the crane wheel.
- **Line 27**: Calls the custom function `create_i_section()` to generate a 3D I-section model using the specified dimensions.
- **Line 30**: Generates the C-section model using the defined parameters through the `create_c_section()` function.
- **Lines 33–36**: Applies a 90-degree rotation (about the X-axis) to the C-section so that its flanges face outward, simulating its typical orientation on top of the I-beam.
- **Lines 38–40**: Translates the rotated C-section to position it correctly on top of the I-section's top flange, aligning it both vertically and laterally.
- **Lines 43–46**: Initializes the OCC 3D viewer and sets a vertical gradient background for improved visual clarity.
- **Lines 49–51**: Displays the I-section using an aluminium material and the transformed C-section in a copper-like appearance.

- **Lines 52–53**: Fits the entire model in the viewer and launches the 3D GUI window for interactive visualization.

Chapter 8

Castellated Beam

8.1 Problem Statement

To write a python script to generate the CAD for Castellated Beam.

8.2 Task Done

Castellated Beam

A castellated beam is a specially fabricated steel beam that features a series of regularly spaced holes (usually hexagonal, circular, or rectangular) cut through the web (the vertical part) of the beam. These holes are created to increase the depth and moment of inertia of the beam without adding additional material, thereby improving its load-carrying capacity while keeping the self-weight low.

In this task, I created a Python script that generates and displays the CAD of Castellated Beam where there is the I section beam with hexagonal holes pattern.

8.3 Python Code

8.3.1 Description of the Script

The script is structured as follows:

- **Input Parameters**: The user defines the beam geometry parameters including

flange width (`B`), flange thickness (`T`), overall beam depth (`D`), web thickness (`t`), notch radii (`R1`, `R2`), beam length (`length`), and notch dimensions (`width`, `hight`). These parameters specify the shape and size of the I-section and the notches cut in the web.

- **I-Section and Notch Creation**: The `Notch` object is instantiated with notch parameters, and an `ISection` object is created using all geometric inputs. The I-section beam model is generated through methods such as `place`, `compute_params`, and `create_model` to obtain a 3D prism representing the beam with notches.

- **Transformation**: A translation transformation is applied to the I-section prism to shift it by +20 units in the X-direction and +25 units in the Z-direction, positioning the beam appropriately in the coordinate space.

- **Hexagonal Pattern Parameters**: Variables like `a`, `b`, `c`, `e`, and `j` control the geometry and spacing of the hexagonal openings cut into the beam web to create the castellated effect. The horizontal (`hex_v`) and vertical (`hex_h`) dimensions of the hexagon are computed based on beam and pattern parameters.

- **Hexagonal Prisms Creation**: A loop calculates the number of hexagonal openings (`n`) along the beam length, then generates hexagonal wires and extrudes them into prisms at calculated positions along the beam.

- **Extrude Cutting**: Each hexagonal prism is subtracted from the translated I-section prism using Boolean cut operations, creating the characteristic castellated web openings.

- **Visualization**: The final castellated beam shape is displayed using the OCC visualization module, fitting the view and starting the GUI event loop for interactive inspection.

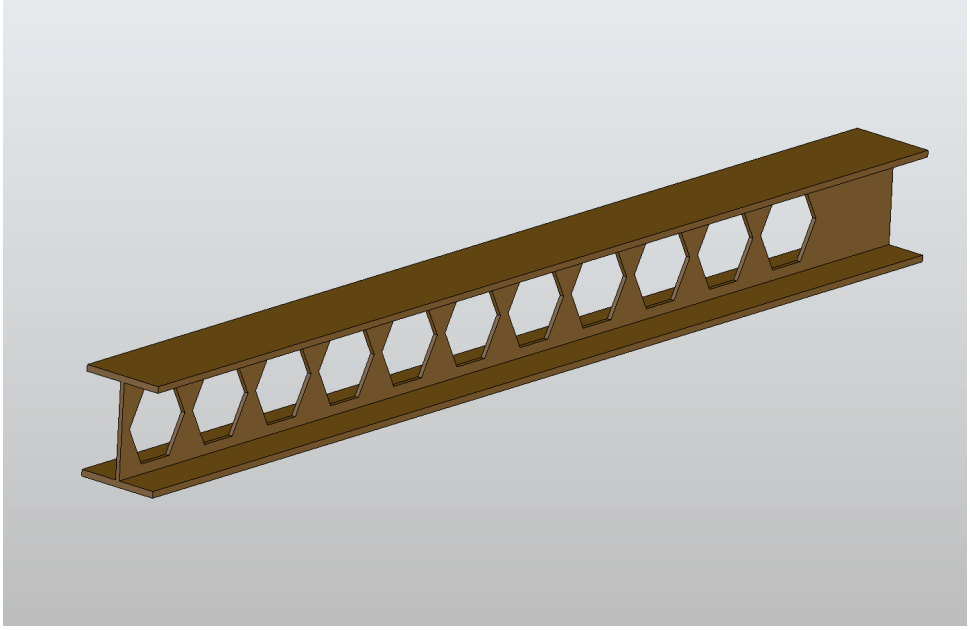

Figure 8.1: Castellated Beam

8.3.2 Python Script

The Python script is shown below. Each section is commented for clarity.

Listing 8.1: Hexagonal Prism For extrude cut on final code

```
%--------------------begin code-------------

import math
from OCC.Core.gp import gp_Pnt, gp_Vec
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_MakePolygon
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_MakeFace
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakePrism
from OCC.Core.gp import gp_Pnt


def hexagon(center, hex_v, hex_h):
    polygon = BRepBuilderAPI_MakePolygon()
    for i in range(6):
        angle = math.radians(i * 60)
        y = center.Y() + (hex_h / 2) * math.cos(angle)
        z = center.Z() + (hex_v / 2) * math.sin(angle)
        polygon.Add(gp_Pnt(center.X(), y, z))  # Keep X constant
    polygon.Close()
```

```
    return polygon.Wire()

if __name__ == '__main__':
    from OCC.Display.SimpleGui import init_display
    display, start_display, add_menu, add_function_to_menu =
        init_display()

    center = gp_Pnt(0, 0, 0)
    hex_w = 5
    hex_h = 7
    hex_wire = hexagon(center, hex_w, hex_h)

    # Create hexagon face
    hex_face = BRepBuilderAPI_MakeFace(hex_wire).Face()

    # Extrude the face along X-axis
    length = 10
    extrude_vec = gp_Vec(length, 0, 0)
    hex_prism = BRepPrimAPI_MakePrism(hex_face, extrude_vec).Shape()

    display.DisplayShape(hex_prism, update=True)
    start_display()
```

Listing 8.2: Castillated Beam

```
%--------------------begin code-------------

import math
import numpy
from OCC.Core.gp import gp_Pnt, gp_Vec,gp_Trsf
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_Transform
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_MakePolygon
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_MakeFace
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakePrism
from OCC.Display.SimpleGui import init_display
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_MakeFace
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Cut
from Hexagon_beam2 import hexagon
from ISection import ISection
from notch import Notch
```

```
from OCC.Display.SimpleGui import init_display
display, start_display, add_menu, add_function_to_menu = init_display()


# Creating I section
B = 40  # width of flanges
T = 3  # flanges sthickness
D = 50 # hright of beam ( falnege thickness +web height)
t = 2  #web thickness
R1 = 5 # notch radius
R2 = 5
alpha = 1
length = 500 # length of beam
width = 10  #notch
hight = 10  #notch
notchObj = Notch(R1, hight, width, length)

origin = numpy.array([0.,0.,0.])
uDir = numpy.array([1.,0.,0.])
shaftDir = numpy.array([0.,1.,0.])

ISec = ISection(B, T, D, t, R1, R2, alpha, length, notchObj)
place = ISec.place(origin, uDir, shaftDir)
point = ISec.compute_params()
prism = ISec.create_model()


# Define the translation vector (shift by +20 in X and +25 in Z) to
    keep it in origin
translation_vector = gp_Vec(20, 0, 25)

# Create a transformation
trsf = gp_Trsf()
trsf.SetTranslation(translation_vector)

# Apply the transformation to the prism
translated_prism = BRepBuilderAPI_Transform(prism, trsf, True).Shape()


```

```
#display.DisplayShape(translated_prism, update=True)


#creating hexagon pattern:

a=5 # edge gap
b=30 # length of flat horizontal edge
c=5 # gap between two hexagon
e=3 # distance between flat edge and middle corner
j=5 # gap between flast edge and flange of i section
hex_v = D-2*j  #horizontal length of hexagon
hex_h = 2*e+b  #vertical length of hexagon
tw = 30  # extrtrude cut length

n=round((length-2*b+c)/(2*e+b+c))

# List to hold all hexagonal prisms
hex_prisms = []
# Create 10 hexagons along X-axis
for i in range(n):
    center = gp_Pnt(0,hex_h/2+a + i * (hex_h+c), D/2)  # shift X each
        time
    hex_wire = hexagon(center, hex_v, hex_h)
    hex_face = BRepBuilderAPI_MakeFace(hex_wire).Face()
    extrude_vec = gp_Vec(tw, 0, 0)
    hex_prism = BRepPrimAPI_MakePrism(hex_face, extrude_vec).Shape()
    hex_prisms.append(hex_prism)

# Display all prisms
#for prism in hex_prisms:
#    display.DisplayShape(prism, update=False)

#extrude cutting
beam=translated_prism
for cutter in hex_prisms:
    beam = BRepAlgoAPI_Cut(beam, cutter).Shape()

display.DisplayShape(beam, update=True)

```

```
92 display.FitAll()
93 start_display()
94
95 %-------------------- end code ---------------
```

8.3.3 Explanation of the Code

- **Lines 1–3**: Imports necessary libraries including `math`, `numpy`, and core OpenCASCADE modules for geometric and shape operations.
- **Lines 4–13**: Imports functions for transformations, face creation, prism extrusion, polygon building, and Boolean operations. Also includes display modules and custom geometry classes: `Hexagon_beam2`, `ISection`, and `Notch`.
- **Line 14**: Initializes the OCC display window and rendering environment.
- **Lines 17–24**: Defines key input parameters for the I-section geometry such as flange width (`B`), flange thickness (`T`), beam depth (`D`), web thickness (`t`), notch dimensions, and beam length.
- **Line 25**: Creates a `Notch` object using specified radius and notch dimensions.
- **Lines 27–29**: Sets the origin and directional vectors `uDir` (beam axis) and `shaftDir` (cross-axis) for the I-section placement.
- **Lines 31–33**: Initializes an `ISection` object, places it in 3D space using direction vectors, computes geometry parameters, and creates the solid prism model.
- **Lines 36–41**: Defines a translation vector to shift the I-beam slightly in the X and Z directions. Applies this transformation to the prism to better center it in the viewer.
- **Lines 48–54**: Defines geometric parameters for the hexagonal cut pattern: flat edge length, spacing, height, and distance from beam surfaces. Calculates how many hexagons fit along the beam length.
- **Lines 56–63**: Creates a series of hexagon-shaped prisms along the beam's length. Each prism is created by forming a hexagonal face and extruding it along the X-axis. These are stored in a list for further use.

- **Lines 67–69**: Iteratively applies Boolean cut operations using the hexagonal prisms to remove material from the I-section beam, forming decorative or functional holes.
- **Line 71**: Displays the final beam model with hexagonal cutouts in the OCC viewer.
- **Lines 73–74**: Fits the entire model in the display window and launches the GUI for user interaction.

Chapter 9

Conclusions

9.1 Tasks Accomplished

CAD Generation Scripts Developed

The following CAD generation scripts were developed and integrated as part of the structural connection modeling tasks:

1. **Butt Joint Bolted CAD Generation Script**
   Developed a script to generate 3D CAD models of bolted butt joints using user-defined parameters such as plate thickness, bolt arrangement, and pitch-gauge distances.

2. **Welded Butt Joint CAD Generation Script**
   Created a script that generates a fully parametric welded butt joint model with two plates and weld geometry using the Open Cascade framework.

3. **CAD Integration of Both Bolted and Welded Butt Joint Connections in Osdag**
   Successfully integrated both CAD scripts into the Osdag application to allow real-time 3D visualization of either connection based on user selection.

4. **Double Angle Gusset Plate Connection**
   Implemented the CAD model for a double-angle gusset plate connection, including angle section placement and weld modeling at each interface.

5. **Gantry Girder Beam CAD Generation Script**
   Developed a 3D model combining an I-section and a rotated C-section to represent a gantry girder beam system with accurate geometric alignment and placement.

6. **Castellated Beam CAD Generation Script**
   Created a script to generate castellated I-beams with hexagonal web openings using a pattern of subtractive extrusion, demonstrating geometry manipulation and cut operations in PythonOCC.

9.2 Skills Developed

Skills Gained During Internship

During the course of a 4-month internship, the following technical and professional skills were acquired:

- Proficient in generating 3D CAD models using **PythonOCC** and Open Cascade libraries.
- Gained a deep understanding of **object-oriented programming** concepts, including the effective use of classes and objects in Python.
- Strengthened **debugging** skills through hands-on troubleshooting of complex codebases.
- Experienced in collaborative **teamwork** within a development environment, including participation in design discussions and code reviews.
- Developed proficiency with **Git** and **GitHub** for version control and collaborative development.
- Contributed to an **open-source project**, adhering to community guidelines, modular coding practices, and documentation standards.

Chapter A

Appendix

A.1 Work Reports

Internship Work Report

Name:	Sachin Saud
Project:	Osdag
Internship:	FOSSEE Semester long Internship 2025

DATE	DAY	TASK	Hours Worked
10-Feb-2025	Monday	Orientation meeting on osdag internship/ Beginning of Internship	4
11-Feb-2025	Tuesday	Tried installing osdag, Introduction session on osdag by the mentors team	4
12-Feb-2025	Wednesday	Investigated issue with erros while installing osdag on my machine | Read installation mannual	2
13-Feb-2025	Thursday	Tried fixing the issues and Learned about steel structures and joints | Task assigned by the mentors- To get familarize with the code of cover plate welded.py and report	2
14-Feb-2025	Friday	fixed the error of installation of osdag with the help of Mehendi Hassan and familarized with the osdag interface| Tried different features of Osdag and ran exaple problems	5
15-Feb-2025	Saturday	Holiday | Started to do task with the help and gudence of Harsan	2
16-Feb-2025	Sunday	Holiday | Work continue | Work submitted	2
17-Feb-2025	Monday	Discussion Meeting on task | Feedback from mentor |	4
18-Feb-2025	Tuesday	Meeting of CAD Team | Assigned task on creating I section perlin	4
19-Feb-2025	Wednesday	Git hub Discipline meeting | Learned about github and exploered it	4
20-Feb-2025	Thursday	Learned About github and its terminologies | Tried possible features with repositories	4
21-Feb-2025	Friday	intsalled and created environment fo python occ library	4
22-Feb-2025	Saturday	Holiday |	
23-Feb-2025	Sunday	Holiday |	
24-Feb-2025	Monday	CAD Meeting purlin module CAD	4
25-Feb-2025	Tuesday	CAD Meeting to discuss the bolted joint | Assigned task-1 for single plated butt joint generation	4
26-Feb-2025	Wednesday	created a single plated plated butt joint coded hard	5
27-Feb-2025	Thursday	Created and placed bolt on the CAD|Task update meeting for task 1	4
28-Feb-2025	Friday	task 3 continue	
1-Mar-2025	Saturday	Holiday	
2-Mar-2025	Sunday	Holiday| Individual CAD Meetng with parth about task 1 update	1
3-Mar-2025	Monday	Worked on palcing bolt from osdag src. | Took leave for exam this day only	
4-Mar-2025	Tuesday	Completed task3- SIngle plated butt joint cad model || Meeting and **assigned task-2**: double plated butt joint and single and double plated weled	4
5-Mar-2025	Wednesday	Worked on double plated butt joint and byplated butt joint CAD	4
6-Mar-2025	Thursday	Worked on double plated welded joint and single plated welded jont	4

Internship Work Report

Name:		Sachin Saud	
Project:		Osdag	
Internship:		FOSSEE Semester long Internship 2025	
7-Mar-2025	Friday	Worked on byplate weld| completed task 3| Cad meeting for update|	4
8-Mar-2025	Saturday	Holiday	
9-Mar-2025	Sunday	Holiday	
10-Mar-2025	Monday	Worked on task 4| multiple bolt placement | Faced and tried to solve errors	4
11-Mar-2025	Tuesday	Leave and could not attend meeting	4
12-Mar-2025	Wednesday	Leave due to final year project defense	4
13-Mar-2025	Thursday	Worked on multiple bolt on sigle plate bolted	4
14-Mar-2025	Friday	Worked on multiple bolt on sigle plate bolted error. Hel from aryan | **Got assigned task 5 - Gusset plate**	4
15-Mar-2025	Saturday	Double angle same side of gusset plate |	4
16-Mar-2025	Sunday	Double angle same side of gusset plate | completed but error	4
17-Mar-2025	Monday	meeting with parth to recify the task work	4
18-Mar-2025	Tuesday	Double angle same side of gusset plate | Wokring	4
19-Mar-2025	Wednesday	Double angle same side of gusset plate | completed	4
20-Mar-2025	Thursday	leave Exams Boards	
21-Mar-2025	Friday	leave Exams Boards	
22-Mar-2025	Saturday	leave Exams Boards	
23-Mar-2025	Sunday	leave Exams Boards	
24-Mar-2025	Monday	leave Exams Boards	
25-Mar-2025	Tuesday	leave Exams Boards	
26-Mar-2025	Wednesday	leave Exams Boards	
27-Mar-2025	Thursday	leave Exams Boards	
28-Mar-2025	Friday	leave Exams Boards	
29-Mar-2025	Saturday	leave Exams Boards	
30-Mar-2025	Sunday	leave Exams Boards	
31-Mar-2025	Monday	leave Exams Boards	
1-Apr-2025	Tuesday	leave Exams Boards	
2-Apr-2025	Wednesday	leave Exams Boards	

Internship Work Report

Name:	Sachin Saud
Project:	Osdag
Internship:	FOSSEE Semester long Internship 2025

3-Apr-2025	Thursday	leave Exams Boards	
4-Apr-2025	Friday	leave Exams Boards	
5-Apr-2025	Saturday	leave Exams Boards	
6-Apr-2025	Sunday	leave Exams Boards	
7-Apr-2025	Monday	leave Exams Boards	
8-Apr-2025	Tuesday	leave Exams Boards	
9-Apr-2025	Wednesday	leave Exams Boards	
10-Apr-2025	Thursday	leave Exams Boards	
11-Apr-2025	Friday	leave Exams Boards	
12-Apr-2025	Saturday	leave Exams Boards	
13-Apr-2025	Sunday	leave Exams Boards	
14-Apr-2025	Monday	leave Exams Boards	
15-Apr-2025	Tuesday	leave Exams Boards	
16-Apr-2025	Wednesday	leave Exams Boards	
17-Apr-2025	Thursday	leave Exams Boards	
18-Apr-2025	Friday	leave Exams Boards	
19-Apr-2025	Saturday	leave Exams Boards	
20-Apr-2025	Sunday	leave Exams Boards	
21-Apr-2025	Monday	leave Exams Boards	4
22-Apr-2025	Tuesday	leave Exams Boards | CAD team meeting	4
23-Apr-2025	Wednesday	CAD team meeting | continuatiion of gusset plate	4
24-Apr-2025	Thursday	Worked on gusset plate weld joint	4
25-Apr-2025	Friday	Worked on gusset plate weld joint	4
26-Apr-2025	Saturday	Worked on gusset plate weld joint placements	4
27-Apr-2025	Sunday	Updated the script with modification	4
28-Apr-2025	Monday	Completed gusset plate weld joints	4
29-Apr-2025	Tuesday	Task update meeting | CAD team meeting|New task : Gantry Girder	4

Internship Work Report

Name:		Sachin Saud	
Project:		Osdag	
Internship:		FOSSEE Semester long Internship 2025	
30-Apr-2025	Wednesday	Worked on Gantry Girder beam	4
1-May-2025	Thursday	Worked on Gantry Girder beam | faced error with I section beam	4
2-May-2025	Friday	Worked on Gantry Girder beam | Resolved the issue using the i beam of osdag items	4
3-May-2025	Saturday	Holiday	
4-May-2025	Sunday	Holiday	
5-May-2025	Monday	Continued Gantry Girder task | Meeting update	4
6-May-2025	Tuesday	Continued Gantry Girder task	4
7-May-2025	Wednesday	Gantry Girder Beam task finalizationn and Update meeting	4
8-May-2025	Thursday	Completed Gantry Girder Beam task	4
9-May-2025	Friday	Reviewed and refined the girder beam	4
10-May-2025	Saturday	Holiday	
11-May-2025	Sunday	Holiday	
12-May-2025	Monday	Meeting | New task assigned: Castillated beam	4
13-May-2025	Tuesday	Worked on Castillated Beam | Created python script for hexagon prism	4
14-May-2025	Wednesday	Worked on Castillated Beam | Created python script for I section beam cutting logic	4
15-May-2025	Thursday	Worked on Castillated Beam | Ran experimentation of extrude cut and learned about cutting in C	4
16-May-2025	Friday	Worked on Castillated Beam | Created python script for final pattern for heaxagonal cutting	4
17-May-2025	Saturday	Holiday	
18-May-2025	Sunday	Holiday	
19-May-2025	Monday	Worked on Castillated Beam |Task Update Meeting	4
20-May-2025	Tuesday	Worked on Castillated Beam | Issues with direction of the extrude cut	4
21-May-2025	Wednesday	Continued working on the Castillated beam	4
22-May-2025	Thursday	Continued working on the Castillated beam	4
23-May-2025	Friday	Continued working on the Castillated beam	4
24-May-2025	Saturday	Holiday	
25-May-2025	Sunday	Holiday	
26-May-2025	Monday	Continued working on the Castillated beam | Update meeting	4

Internship Work Report

Name:		Sachin Saud	
Project:		Osdag	
Internship:		FOSSEE Semester long Internship 2025	
27-May-2025	Tuesday	Final working on the Castillated beam	4
28-May-2025	Wednesday	the Castillated beam Completed	4
29-May-2025	Thursday	Meeting with Parth about update and new task assigned	4
30-May-2025	Friday	Learned about gui integration in osdag	4
31-May-2025	Saturday	Holiday	
1-Jun-2025	Sunday	Holiday	
2-Jun-2025	Monday	CAD Integration of Butt Joint Bolted Connection with Osdag Task Assigned	4
3-Jun-2025	Tuesday	Worked on CAD integration| Read and reviewed Aryan's Mannual on CAD Integration	4
4-Jun-2025	Wednesday	Continued working on CAD integration| Tried to make changes on necessary files	4
5-Jun-2025	Thursday	Continued working on CAD integration|Faced Error with my code	4
6-Jun-2025	Friday	Continued working on CAD integration| Re written the code for butt joint bolted according to the c	4
7-Jun-2025	Saturday	Holiday	
8-Jun-2025	Sunday	Holiday	
9-Jun-2025	Monday	Continued working on CAD integration| Faced Error in running Main page of osdag | Task update	4
10-Jun-2025	Tuesday	Continued working on CAD integration| Solved the error of pyqt5	4
11-Jun-2025	Wednesday	Continued working on CAD integration| Meeting with Harsan about discussion	4
12-Jun-2025	Thursday	Completed the CAD integration for butt joint bolted connection |Pused the code to github	4
13-Jun-2025	Friday	CAD integration for butt joint bolted connection update and meeting | new task assigned	4
14-Jun-2025	Saturday	Holiday	4
15-Jun-2025	Sunday	Holiday	4
16-Jun-2025	Monday	New task: Cad integration of butt joint welded connection | worked on it	4
17-Jun-2025	Tuesday	Worked on Cad integration of butt joint welded connection	4
18-Jun-2025	Wednesday	Worked on Cad integration of butt joint welded connection | File of Tanu	4
19-Jun-2025	Thursday	Continued working on Cad integration of butt joint welded connection	4
20-Jun-2025	Friday	Continued working on Cad integration of butt joint welded connection	4
21-Jun-2025	Saturday	Holiday	4
22-Jun-2025	Sunday	Holiday	4

Internship Work Report

Name:		Sachin Saud	
Project:		Osdag	
Internship:		FOSSEE Semester long Internship 2025	
23-Jun-2025	Monday	Continued working on Cad integration of butt joint welded connection | Task update meeting	4
24-Jun-2025	Tuesday	Continued working on Cad integration of butt joint welded connection Error with weld parameters	4
25-Jun-2025	Wednesday	Continued working on Cad integration of butt joint welded connection | Meeting with tanu	4
26-Jun-2025	Thursday	Continued working on Cad integration of butt joint welded connection | Fixed error by discussion	4
27-Jun-2025	Friday	Cad integration of butt joint welded connection | Pused the code to github	4
28-Jun-2025	Saturday	Holiday	4
29-Jun-2025	Sunday	Holiday	
30-Jun-2025	Monday	End of the Internship	

Bibliography

[1] Siddhartha Ghosh, Danish Ansari, Ajmal Babu Mahasrankintakam, Dharma Teja Nuli, Reshma Konjari, M. Swathi, and Subhrajit Dutta. Osdag: A Software for Structural Steel Design Using IS 800:2007. In Sondipon Adhikari, Anjan Dutta, and Satyabrata Choudhury, editors, *Advances in Structural Technologies*, volume 81 of *Lecture Notes in Civil Engineering*, pages 219–231, Singapore, 2021. Springer Singapore.

[2] FOSSEE Project. FOSSEE News - January 2018, vol 1 issue 3. Accessed: 2024-12-05.

[3] FOSSEE Project. Osdag website. Accessed: 2024-12-05.