FOSSEE Summer Fellowship Report

On

CAD Development and Integration of Pyplots for Osdag

Submitted by

Harsh Gondal

3rd Year B.Tech Student, Department of Computer Science

Manipal Institute of Technology,Manipal

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 3, 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

1 Introduction **4**

1.1 National Mission in Education through ICT 4

1.1.1 ICT Initiatives of MoE 5

1.2 FOSSEE Project 6

1.2.1 Projects and Activities 6

1.2.2 Fellowships 6

1.3 Osdag Software 7

1.3.1 Osdag GUI 8

1.3.2 Features 8

2 Screening Task **9**

2.1 Problem Statement 9

2.1.1 Shear Force and Bending Moment Pyplots 9

2.1.2 Tasks Done 9

2.2 PythonOCC Code Analysis: Laced Compound Column 13

3 CAD Development **18**

3.1 Task 1: Problem Statement 18

3.2 Task 1: Tasks Done 18

3.3 Task 1: Python Code 20

3.3.1 Description of the Script: Laced Column and Double Laced Column 20

3.3.2 Python Code 20

3.3.3 Explanation of the Code 31

3.4 Task 1: Documentation 33

3.4.1 Directory Structure 33

4 Internship Task 2 Title **35**

4.1 Task 2: Problem Statement 35

4.2 Task 2: Tasks Done 35

4.2.1 Development and Integration of BMD and SFD Plotting Feature . 35

4.2.2 Python Code 38

4.3 Task 2: Documentation 44
4.3.1 Directory Structure 45

5 Conclusions **46**
5.1 Tasks Accomplished 46
5.1.1 Integration of BMD and SFD Plotting Using PyPlot 46
5.1.2 CAD Modeling of Laced and Double-Laced Welded Columns . . 47
5.2 Skills Developed 47

A Appendix **49**
A.1 Work Reports 49

Bibliography **51**

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.

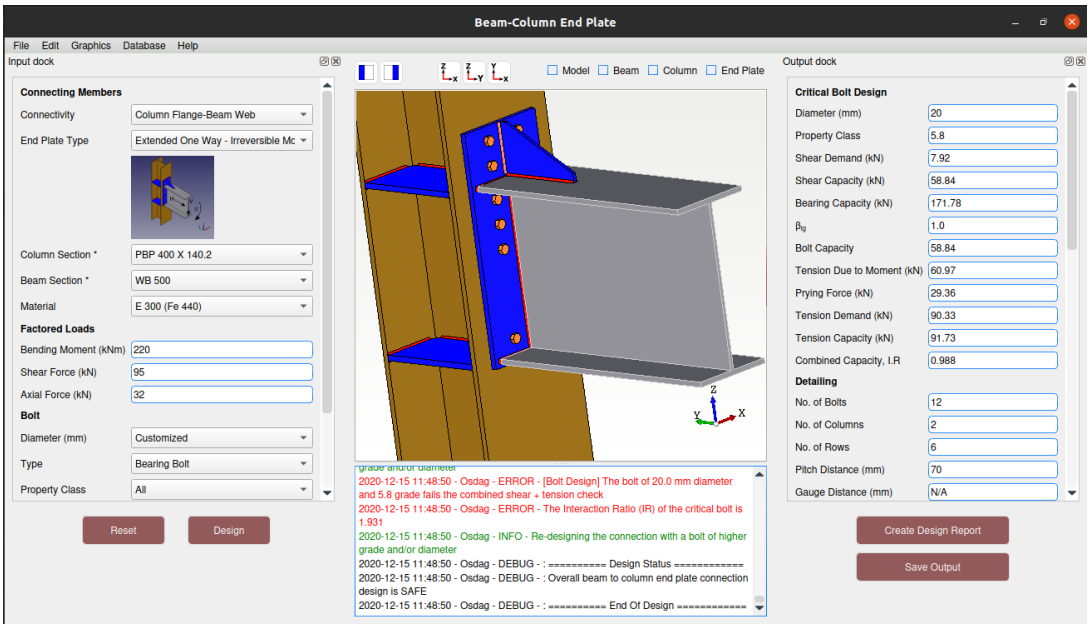


Figure 1.2: Osdag GUI

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

2.1.1 Shear Force and Bending Moment Pyplots

The objective is to create a Shear Force Diagram (SFD) and Bending Moment Diagram (BMD) from the values provided in an Excel sheet, using `PyPlot`. These diagrams help visualize how internal forces and moments vary along the length of a beam.

2.1.2 Tasks Done

Theoretical Background

Shear Force:

Shear force is the internal force acting perpendicular to the longitudinal axis of a structural member. It results from transverse loads and causes shearing stress in the member.

Bending Moment:

Bending moment is the internal moment that causes bending in structural members. It is calculated by multiplying the force by the perpendicular distance from the point of interest.

Code Implementation

The Python implementation uses the `pandas` library for data handling and `matplotlib.pyplot` for visualization. Below is a step-by-step explanation.

Data Preparation

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the data from the provided CSV file
file_path = r"C:\Users\Downloads\SFS_Screening_SFDBMD.xlsx -
   Sheet1.csv"
data = pd.read_csv(file_path)

# Extracting data for plotting
x = data['Distance (m)']
shear_force = data['SF (kN)']
bending_moment = data['BM (kN-m)']
```

The data is loaded from a CSV file using `pandas`, and the columns for distance, shear force, and bending moment are extracted for plotting.

Plot Configuration

```
fig, axes = plt.subplots(2, 1, figsize=(10, 8))
```

A figure with two vertically stacked subplots is created: one for the SFD and one for the BMD.

Shear Force Diagram Implementation

```
# --- Shear Force Diagram (SFD) ---
axes[0].plot(x, shear_force, drawstyle='steps-post', color='b',
   linewidth=1.5, label='Shear Force')

# Vertical lines showing step changes between points
for i in range(len(x) - 1):
    axes[0].vlines(x[i+1], shear_force[i], shear_force[i+1],
       colors='blue', linewidth=1.5, linestyles='solid')

# Add dashed vertical lines from each point to the x-axis
for xi, sf in zip(x, shear_force):
    axes[0].vlines(xi, 0, sf, colors='blue', linestyles='solid',
       linewidth=1)
```

The SFD is plotted using a step function and vertical lines to represent discontinuities and connect each shear value to the axis.

Maximum Value Annotation for SFD

```
dist_value = 6
sfd_value = data.loc[data['Distance (m)'] == dist_value, 'SF (kN)
    '].values[0]
axes[0].text(dist_value, sfd_value, f"Max: {sfd_value:.2f} kN",
    fontsize=10, color='black', ha='left', va='bottom', fontweight
    ='bold')
```

The maximum shear force is annotated at a distance of 6 meters.

SFD Formatting

```
axes[0].axhline(0, color='black', linewidth=0.8)
axes[0].set_xlabel("Distance (m)")
axes[0].set_ylabel("Shear Force (kN)")
axes[0].set_title("Shear Force Diagram (SFD)")
axes[0].legend()
axes[0].grid(True, linestyle='dashed', alpha=0.7)
```

Bending Moment Diagram Implementation

```
# --- Bending Moment Diagram (BMD) ---
axes[1].plot(x, bending_moment, color='r', linewidth=1.5, label='
    Bending Moment')

# Add vertical lines from each point to the x-axis
for xi, bm in zip(x, bending_moment):
    axes[1].vlines(xi, 0, bm, colors='red', linestyles='solid',
        linewidth=1)
```

The BMD is plotted as a continuous curve, and each point is connected to the x-axis with a vertical line.

BMD Formatting

```
axes[1].axhline(0, color='black', linewidth=0.8)
axes[1].set_xlabel("Distance (m)")
axes[1].set_ylabel("Bending Moment (kN-m)")
```

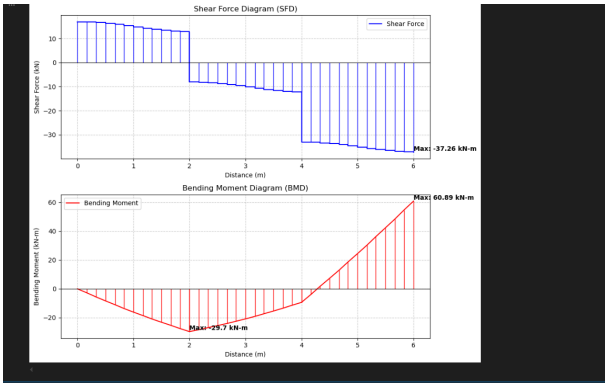


Figure 2.1: BMD and SFD Pyplots

```
axes[1].set_title("Bending Moment Diagram (BMD)")
axes[1].legend()
axes[1].grid(True, linestyle='dashed', alpha=0.7)
```

Final Display

```
plt.tight_layout()
plt.show()
```

Interpretation of SFD

The Shear Force Diagram illustrates:

- The location of maximum shear force.
- Points where the shear force changes sign (zero crossing).
- Segments where the beam undergoes high shear stress.

Interpretation of BMD

The Bending Moment Diagram reveals:

- Maximum positive and negative bending moments.
- Points of maximum bending stress.
- Points of contraflexure (locations where the bending moment is zero).

2.2 PythonOCC Code Analysis: Laced Compound Column

Introduction

This task involves analyzing a Python script designed to model a **laced compound column** using the Open Cascade Technology (OCCT) CAD kernel via the Python wrapper, `PythonOCC`.

A laced compound column is a structural system composed of two parallel I-sections connected by end battens and diagonal lacing plates. It is widely used in steel construction for its enhanced load-bearing capacity and structural stability.

Libraries and Imports

The script utilizes various components from the OCCT library:

- `gp_Vec, gp_Trsf, gp_Pnt, gp_Dir, gp_Ax1` – geometric primitives and transformations
- `BRepPrimAPI_MakeBox` – for creating box-shaped components
- `BRepAlgoAPI_Fuse` – for Boolean fusing operations
- `BRepBuilderAPI_Transform` – for shape transformations
- `BRepBuilderAPI_MakeEdge`, `MakeWire`, `MakeFace` – for 2D geometry construction
- `BRepPrimAPI_MakePrism` – for extruding 2D faces into 3D solids
- `init_display` – for visualizing the 3D model

Function Overview

- **create_i_section:** Constructs a single I-section by fusing three rectangular boxes representing the flanges and web.
- **create_end_batten:** Builds rectangular plates (battens) to connect the top and bottom of the I-sections.

- **create_straight_lace:** Creates horizontal lace plates using a rectangular box.
- **create_parallelogram_face:** Generates diagonal lacing plates in a parallelogram form by defining three corner points and computing the fourth. The face is then extruded into a solid.

Key Parameters

The script defines several parameters to control geometry:

- `column_length = 6100.0 mm`
- `i_section_width = 100.0 mm`, `i_section_height = 200.0 mm`
- `flange_thickness = 10.0 mm`, `web_thickness = 5.0 mm`
- `column_distance = 450.0 mm`
- `batten_depth = 300.0 mm`, `batten_thickness = 10.0 mm`
- `lace_width = 100.0 mm`, `lace_thickness = 8.0 mm`

Construction Process

1. **Create I-sections:** Two I-beams are generated, and one is translated to form a parallel column system.
2. **Add End Battens:** Four battens are placed between the columns - top / bottom and left / right.
3. **Fuse Components:** The I-sections and battens are fused into a single structure.
4. **Create Diagonal Laces:**
    - Loops generate forward-leaning (/) and backward-leaning (\) laces at multiple heights.
    - Two horizontal laces are added between diagonals.
5. **Final Touches:** Extra diagonal laces are placed at the ends to complete the structure.

6. **Visualization:** The entire composite structure is rendered using `init_display()` from PythonOCC.

Output Image

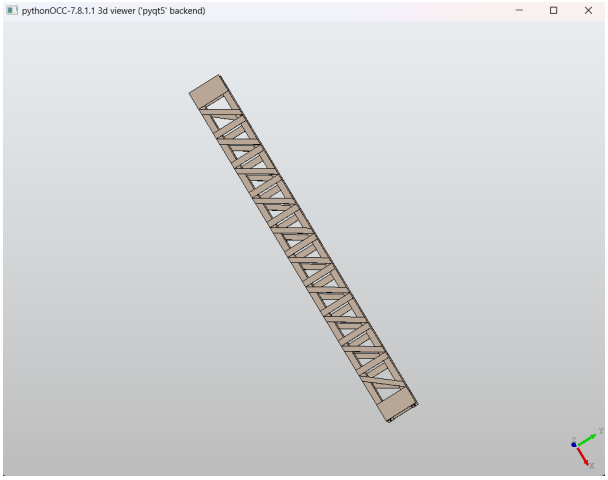


Figure 2.2: Laced Column Compound

Code Screenshot

The implementation is shown in the following figure:

```
from OCC.Core.gp import gp_Vec, gp_Trsf, gp_Pnt, gp_Dir, gp_Ax1
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_Transform,
   BRepBuilderAPI_MakeEdge, BRepBuilderAPI_MakeWire,
   BRepBuilderAPI_MakeFace
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakePrism
from OCC.Display.SimpleGui import init_display
import math

def create_i_section(length, width, depth, flange_thickness,
   web_thickness):
    web_height = depth - 2 * flange_thickness

    bottom_flange = BRepPrimAPI_MakeBox(length, width,
       flange_thickness).Shape()
```

```
    top_flange = BRepPrimAPI_MakeBox(length, width,
        flange_thickness).Shape()
    trsf = gp_Trsf()
    trsf.SetTranslation(gp_Vec(0, 0, depth - flange_thickness))
    top_flange_transform = BRepBuilderAPI_Transform(top_flange,
        trsf, True).Shape()

    web = BRepPrimAPI_MakeBox(length, web_thickness, web_height).
        Shape()
    trsf = gp_Trsf()
    trsf.SetTranslation(gp_Vec(0, (width - web_thickness) / 2,
        flange_thickness))
    web_transform = BRepBuilderAPI_Transform(web, trsf, True).
        Shape()

    i_section_solid = BRepAlgoAPI_Fuse(bottom_flange,
        top_flange_transform).Shape()
    i_section_solid = BRepAlgoAPI_Fuse(i_section_solid,
        web_transform).Shape()

    return i_section_solid

def create_end_batten(length, width, depth, column_distance):
    batten = BRepPrimAPI_MakeBox(length, column_distance + width,
         depth).Shape()
    return batten

def create_parallelogram_face(p1, p2, p3, thickness):
    # Calculate the fourth point (Top-right)
    vec1 = gp_Vec(p1, p2)  # Base vector
    p4 = gp_Pnt(p3.XYZ() + vec1.XYZ())  # Top-right = Top-left +
        base vector

    # Create edges
```

```
    edge1 = BRepBuilderAPI_MakeEdge(p1, p2).Edge()
    edge2 = BRepBuilderAPI_MakeEdge(p2, p4).Edge()
    edge3 = BRepBuilderAPI_MakeEdge(p4, p3).Edge()
    edge4 = BRepBuilderAPI_MakeEdge(p3, p1).Edge()

    # Make wire
    wire = BRepBuilderAPI_MakeWire(edge1, edge2, edge3, edge4).
       Wire()

    # Make face
    face = BRepBuilderAPI_MakeFace(wire).Face()

    # Extrude the face into a solid
    vec = gp_Vec(0, 0, thickness)
    prism = BRepPrimAPI_MakePrism(face, vec).Shape()

    return prism
```

Chapter 3

CAD Development

3.1 Task 1: Problem Statement

CAD Development of Laced and Double Laced Welded Column using PythonOCC and its libraries.

3.2 Task 1: Tasks Done

Developed complete CAD models for both laced and double-laced welded column assemblies using PythonOCC. These columns consist of two parallel I-sections connected by diagonal lace plates and transverse end battens for structural stability and load transfer.

To realistically represent fabrication details, triangular fillet welds were implemented at strategic junctions:

At the interfaces where diagonal laces meet the I-sections, ensuring secure transfer of shear forces.

At the points where end battens connect to the flanges of the I-sections, providing rigidity against out-of-plane deformations.

These triangular welds were modeled as extruded triangular prisms, oriented and placed based on the local geometry and connection type (XZ, YZ, or XY plane). The design ensures each component is physically joined and ready for visualization, finite element meshing, or further structural analysis.

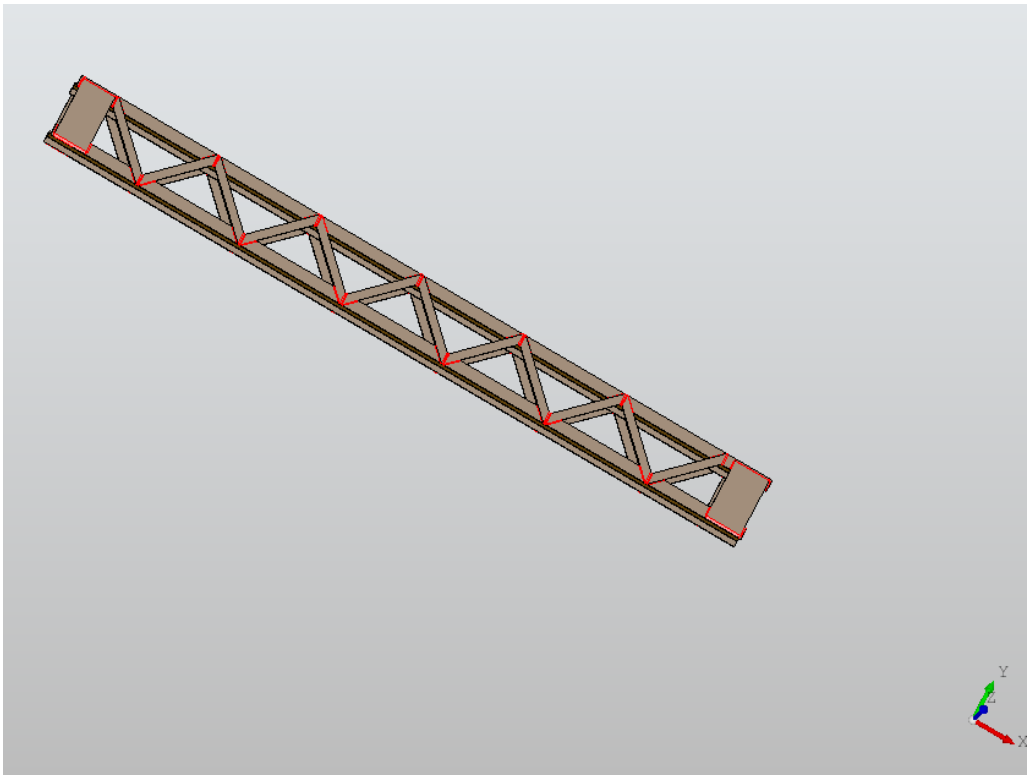


Figure 3.1: CAD model of a laced welded column with triangular fillet welds at diagonal lace and batten junctions.

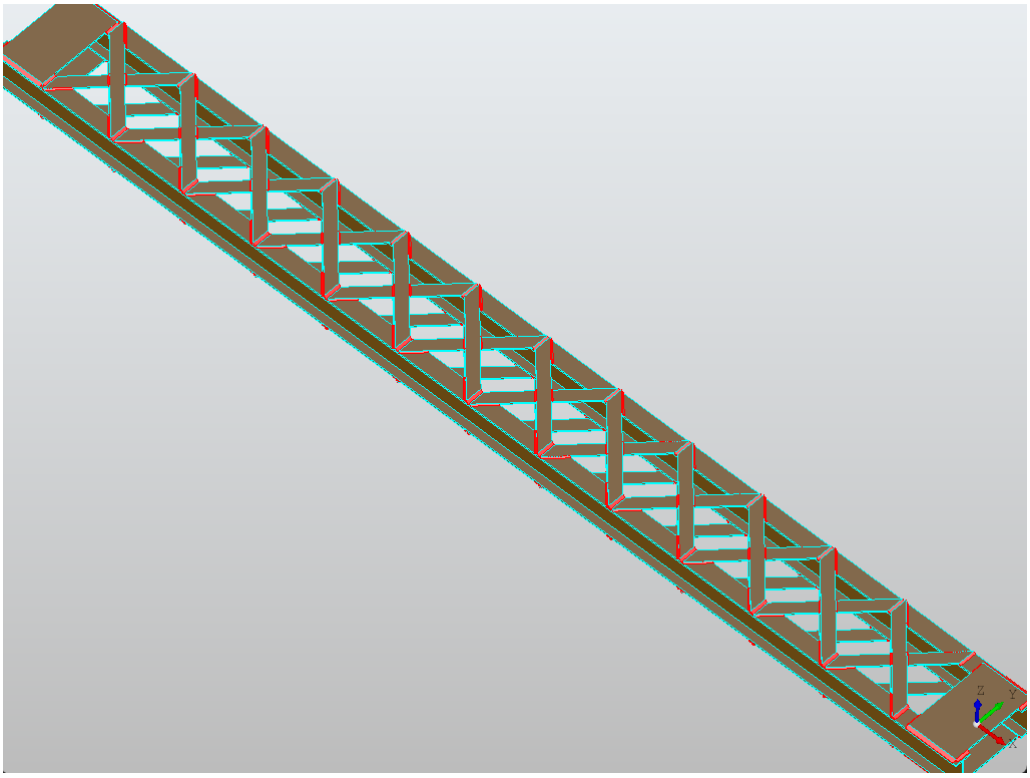


Figure 3.2: CAD model of a double-laced welded column showing diagonal laces in both directions and weld detailing.

3.3 Task 1: Python Code

Include codes developed during your internship. Give a proper description. Example is given here. This section presents a Python script for designing a beam-to-column connection using the Osdag framework. The script performs calculations based on user-defined parameters like beam section, column section, and applied shear load. It automates the design process, ensuring compliance with Indian Standards (IS) codes.

3.3.1 Description of the Script: Laced Column and Double Laced Column

The script is structured as follows:

- **Input Parameters**: The user defines geometric properties such as the column length, I-section dimensions (width, depth, flange thickness, web thickness), batten dimensions, lace width and thickness, and spacing between I-sections.
- **Geometry Creation**: The script builds two parallel I-sections, places end battens at the top and bottom, and creates diagonal lace plates connecting the I-sections. Weld details are generated using triangular prism solids placed at connection junctions.
- **Weld Modeling**: Triangular fillet welds are placed at corners where laces and battens join the I-sections, and along edges where applicable. Functions like `create_corner_triangul` and `create_edge_weld` are used for this purpose. In double laced column there are extra welds at the intersection of the two diagonal laces to support them.
- **Assembly**: All parts — I-sections, laces, battens, and welds — are transformed and fused using `BRepAlgoAPI_Fuse` to create a complete, manufacturable solid model.

3.3.2 Python Code

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

Listing 3.1: Laced and Double Column Design

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

from OCC.Core.gp import gp_Vec, gp_Trsf, gp_Pnt, gp_Dir, gp_Ax1
from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox,
    BRepPrimAPI_MakePrism
from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_Transform,
    BRepBuilderAPI_MakeEdge, BRepBuilderAPI_MakeWire,
    BRepBuilderAPI_MakeFace
from OCC.Display.SimpleGui import init_display
from OCC.Core.Quantity import Quantity_Color, Quantity_NOC_RED
import math
import numpy


def make_edge(*args):
    edge = BRepBuilderAPI_MakeEdge(*args)
    result = edge.Edge()
    return result

def make_wire(*args):
    if isinstance(args[0], list) or isinstance(args[0], tuple):
        wire = BRepBuilderAPI_MakeWire()
        for i in args[0]:
            wire.Add(i)
        wire.Build()
        return wire.Wire()
    wire = BRepBuilderAPI_MakeWire(*args)
    return wire.Wire()

def makeWireFromEdges(edges):
    wire = None
    for edge in edges:
        if wire:
            wire = make_wire(wire, edge)
        else:
            wire = make_wire(edge)
```

```
35      return wire
36
37  def makeFaceFromWire(wire):
38      return BRepBuilderAPI_MakeFace(wire).Face()
39
40  def getGpPt(point):
41      return gp_Pnt(point[0], point[1], point[2])
42
43  def makeEdgesFromPoints(points):
44      edges = []
45      num = len(points)
46      for i in range(num - 1):
47          edge = make_edge(getGpPt(points[i]), getGpPt(points[i + 1]))
48          edges.append(edge)
49      # Close the loop
50      cycleEdge = make_edge(getGpPt(points[num - 1]), getGpPt(points[0]))
51      edges.append(cycleEdge)
52      return edges
53
54  def makePrismFromFace(aFace, eDir):
55      return BRepPrimAPI_MakePrism(aFace, gp_Vec(gp_Pnt(0., 0., 0.),
            gp_Pnt(eDir[0], eDir[1], eDir[2]))).Shape()
56
57  def create_corner_triangular_weld(corner_point, size, length,
        orientation='xy',flag='positive',end=False):
58      """Create a triangular fillet weld at corner intersection"""
59      x, y, z = corner_point
60
61      if orientation == 'xy':  # Horizontal corner weld
62          # Right triangle in XY plane
63          p1 = gp_Pnt(x, y, z)              # Corner point
64          p2 = gp_Pnt(x + size, y, z)       # Along X axis
65          p3 = gp_Pnt(x, y + size, z)       # Along Y axis
66
67          # Create triangular face
68          edge1 = BRepBuilderAPI_MakeEdge(p1, p2).Edge()
69          edge2 = BRepBuilderAPI_MakeEdge(p2, p3).Edge()
70          edge3 = BRepBuilderAPI_MakeEdge(p3, p1).Edge()
71
```

```
        wire = BRepBuilderAPI_MakeWire(edge1, edge2, edge3).Wire()
        face = BRepBuilderAPI_MakeFace(wire).Face()

        # Extrude along Z axis
        vec = gp_Vec(0, 0, length)

    elif orientation == 'xz':  # Vertical corner weld in XZ plane
        if end==True:
            p1 = gp_Pnt(x, y, z)              # Corner point
            p2 = gp_Pnt(x, y-20, z)       # Along X axis
            p3 = gp_Pnt(x, y, z + size)       # Along Z axis
        if flag=='negative':
            p1 = gp_Pnt(x, y, z)              # Corner point
            p2 = gp_Pnt(x + 12, y, z)       # Along X axis
            p3 = gp_Pnt(x, y, z - size)       # Along Z axis
        else:
        # Right triangle in XZ plane
            p1 = gp_Pnt(x, y, z)              # Corner point
            p2 = gp_Pnt(x + 12, y, z)       # Along X axis
            p3 = gp_Pnt(x, y, z + size)       # Along Z axis

        # Create triangular face
        edge1 = BRepBuilderAPI_MakeEdge(p1, p2).Edge()
        edge2 = BRepBuilderAPI_MakeEdge(p2, p3).Edge()
        edge3 = BRepBuilderAPI_MakeEdge(p3, p1).Edge()

        wire = BRepBuilderAPI_MakeWire(edge1, edge2, edge3).Wire()
        face = BRepBuilderAPI_MakeFace(wire).Face()

        # Extrude along Y axis
        vec = gp_Vec(0, length, 0)

    elif orientation == 'yz':  # Vertical corner weld in YZ plane
        # Right triangle in YZ plane
        if flag=="negative":

            p1 = gp_Pnt(x, y, z)              # Corner point
            p2 = gp_Pnt(x, y - 20, z)       # Along Y axis
            p3 = gp_Pnt(x, y, z - size)       # Along Z axis
```

```
        else:

            p1 = gp_Pnt(x, y, z)              # Corner point
            p2 = gp_Pnt(x, y - 20, z)        # Along Y axis
            p3 = gp_Pnt(x, y, z + size)       # Along Z axis

        # Create triangular face
        edge1 = BRepBuilderAPI_MakeEdge(p1, p2).Edge()
        edge2 = BRepBuilderAPI_MakeEdge(p2, p3).Edge()
        edge3 = BRepBuilderAPI_MakeEdge(p3, p1).Edge()

        wire = BRepBuilderAPI_MakeWire(edge1, edge2, edge3).Wire()
        face = BRepBuilderAPI_MakeFace(wire).Face()

        # Extrude along X axis
        vec = gp_Vec(length, 0, 0)

    prism = BRepPrimAPI_MakePrism(face, vec).Shape()
    return prism


def create_corner_triangular_weldo(corner_point, size, length,
    orientation='xy',flag="positive"):
    """Create a triangular fillet weld at corner intersection"""
    x, y, z = corner_point

    if orientation == 'xy':  # Horizontal corner weld
        # Right triangle in XY plane
        p1 = gp_Pnt(x, y, z)              # Corner point
        p2 = gp_Pnt(x + size, y, z)       # Along X axis
        p3 = gp_Pnt(x, y + size, z)       # Along Y axis

        # Create triangular face
        edge1 = BRepBuilderAPI_MakeEdge(p1, p2).Edge()
        edge2 = BRepBuilderAPI_MakeEdge(p2, p3).Edge()
        edge3 = BRepBuilderAPI_MakeEdge(p3, p1).Edge()

        wire = BRepBuilderAPI_MakeWire(edge1, edge2, edge3).Wire()
        face = BRepBuilderAPI_MakeFace(wire).Face()
```

```
149
150
151         # Extrude along Z axis
152         vec = gp_Vec(0, 0, length)
153
154     elif orientation == 'xz':  # Vertical corner weld in XZ plane
155         # if flag==False:
156         #     p1 = gp_Pnt(x, y, z)              # Corner point
157         #     p2 = gp_Pnt(x - size-10, y, z)       # Along X axis
158         #     p3 = gp_Pnt(x, y, z + size)
159         # elif flag2==False:
160         #     p1 = gp_Pnt(x, y, z)              # Corner point
161         #     p2 = gp_Pnt(x - size, y, z)       # Along X axis
162         #     p3 = gp_Pnt(x, y, z - size)
163         # else:
164         # Right triangle in XZ plane
165         if flag=="negative":
166             p1 = gp_Pnt(x, y, z)              # Corner point
167             p2 = gp_Pnt(x - 8, y, z)       # Along X axis
168             p3 = gp_Pnt(x, y, z - size)       # Along Z axis
169         else:
170             p1 = gp_Pnt(x, y, z)              # Corner point
171             p2 = gp_Pnt(x - 8, y, z)       # Along X axis
172             p3 = gp_Pnt(x, y, z + size)       # Along Z axis
173
174         # Create triangular face
175         edge1 = BRepBuilderAPI_MakeEdge(p1, p2).Edge()
176         edge2 = BRepBuilderAPI_MakeEdge(p2, p3).Edge()
177         edge3 = BRepBuilderAPI_MakeEdge(p3, p1).Edge()
178
179         wire = BRepBuilderAPI_MakeWire(edge1, edge2, edge3).Wire()
180         face = BRepBuilderAPI_MakeFace(wire).Face()
181
182         # Extrude along Y axis
183         vec = gp_Vec(0, length, 0)
184
185     elif orientation == 'yz':  # Vertical corner weld in YZ plane
186         # Right triangle in YZ plane
187         if flag=="negative":
```

```
            p1 = gp_Pnt(x, y, z)                # Corner point
            p2 = gp_Pnt(x, y + 20, z)         # Along Y axis
            p3 = gp_Pnt(x, y, z - size)         # Along Z axis
        else:
            p1 = gp_Pnt(x, y, z)                # Corner point
            p2 = gp_Pnt(x, y + 20, z)         # Along Y axis
            p3 = gp_Pnt(x, y, z + size)         # Along Z axis

        # Create triangular face
        edge1 = BRepBuilderAPI_MakeEdge(p1, p2).Edge()
        edge2 = BRepBuilderAPI_MakeEdge(p2, p3).Edge()
        edge3 = BRepBuilderAPI_MakeEdge(p3, p1).Edge()

        wire = BRepBuilderAPI_MakeWire(edge1, edge2, edge3).Wire()
        face = BRepBuilderAPI_MakeFace(wire).Face()

        # Extrude along X axis
        vec = gp_Vec(length, 0, 0)

    prism = BRepPrimAPI_MakePrism(face, vec).Shape()
    return prism

def create_edge_weld(start_point, end_point, weld_size,
    thickness_direction='z',weld_size_x=0,theta=0,flag="positive"):
    """Create fillet weld along an edge"""
    x1, y1, z1 = start_point
    x2, y2, z2 = end_point

    if thickness_direction == 'z':
        if weld_size_x!=0:
                if flag=="negative":
                    p1 = gp_Pnt(x1, y1, z1)
                    p2 = gp_Pnt(x1 + weld_size_x*math.cos(theta), y1-
                        weld_size_x*math.sin(theta), z1)
                    p3 = gp_Pnt(x1, y1, z1 - weld_size)
                else:
                    p1 = gp_Pnt(x1, y1, z1)
                    p2 = gp_Pnt(x1 + weld_size_x*math.cos(theta), y1+
                        weld_size_x*math.sin(theta), z1)
```

```
                    p3 = gp_Pnt(x1, y1, z1 + weld_size)
        # Create triangular cross-section weld along Z direction
        # Base points for triangle
        else:
            if flag=="negative":
                p1 = gp_Pnt(x1, y1, z1)
                p2 = gp_Pnt(x1 + weld_size, y1, z1)
                p3 = gp_Pnt(x1, y1, z1 - weld_size)
            else:
                p1 = gp_Pnt(x1, y1, z1)
                p2 = gp_Pnt(x1 + weld_size, y1, z1)
                p3 = gp_Pnt(x1, y1, z1 + weld_size)

        # Create triangular face
        edge1 = BRepBuilderAPI_MakeEdge(p2, p1).Edge()
        edge2 = BRepBuilderAPI_MakeEdge(p3, p2).Edge()
        edge3 = BRepBuilderAPI_MakeEdge(p3, p1).Edge()

        wire = BRepBuilderAPI_MakeWire(edge1, edge2, edge3).Wire()
        face = BRepBuilderAPI_MakeFace(wire).Face()

        # Extrude along the edge direction
        edge_length = math.sqrt((x2-x1)**2 + (y2-y1)**2 + (z2-z1)**2)
        direction = gp_Vec(x2-x1, y2-y1, z2-z1)
        direction.Normalize()
        direction.Scale(edge_length)

    elif thickness_direction == 'y':
        # Create triangular cross-section weld along Y direction
        p1 = gp_Pnt(x1, y1, z1)
        p2 = gp_Pnt(x1 + weld_size, y1, z1)
        p3 = gp_Pnt(x1, y1 + weld_size, z1)

        edge1 = BRepBuilderAPI_MakeEdge(p1, p2).Edge()
        edge2 = BRepBuilderAPI_MakeEdge(p2, p3).Edge()
        edge3 = BRepBuilderAPI_MakeEdge(p3, p1).Edge()

        wire = BRepBuilderAPI_MakeWire(edge1, edge2, edge3).Wire()
        face = BRepBuilderAPI_MakeFace(wire).Face()
```

```
263
264         edge_length = math.sqrt((x2-x1)**2 + (y2-y1)**2 + (z2-z1)**2)
265         direction = gp_Vec(x2-x1, y2-y1, z2-z1)
266         direction.Normalize()
267         direction.Scale(edge_length)
268
269     prism = BRepPrimAPI_MakePrism(face, direction).Shape()
270     return prism
271
272
273 def create_edge_weldo(start_point, end_point, weld_size,
    thickness_direction='z',weld_size_x=0,theta=0,flag="positive"):
274     """Create fillet weld along an edge"""
275     x1, y1, z1 = start_point
276     x2, y2, z2 = end_point
277
278     if thickness_direction == 'z':
279         if weld_size_x!=0:
280                 if flag=="negative":
281                     p1 = gp_Pnt(x1, y1, z1)
282                     p2 = gp_Pnt(x1 - weld_size_x*math.cos(theta), y1+
                        weld_size_x*math.sin(theta), z1)
283                     p3 = gp_Pnt(x1, y1, z1 - weld_size)
284                 else:
285                     p1 = gp_Pnt(x1, y1, z1)
286                     p2 = gp_Pnt(x1 - weld_size_x*math.cos(theta), y1-
                        weld_size_x*math.sin(theta), z1)
287                     p3 = gp_Pnt(x1, y1, z1 + weld_size)
288         # Create triangular cross-section weld along Z direction
289         # Base points for triangle
290         else:
291             if flag=="negative":
292                 p1 = gp_Pnt(x1, y1, z1)
293                 p2 = gp_Pnt(x1 - weld_size, y1, z1)
294                 p3 = gp_Pnt(x1, y1, z1 - weld_size)
295             else:
296                 p1 = gp_Pnt(x1, y1, z1)
297                 p2 = gp_Pnt(x1 - weld_size, y1, z1)
298                 p3 = gp_Pnt(x1, y1, z1 + weld_size)
```

```
299
300         # Create triangular face
301         edge1 = BRepBuilderAPI_MakeEdge(p1, p2).Edge()
302         edge2 = BRepBuilderAPI_MakeEdge(p2, p3).Edge()
303         edge3 = BRepBuilderAPI_MakeEdge(p3, p1).Edge()
304
305         wire = BRepBuilderAPI_MakeWire(edge1, edge2, edge3).Wire()
306         face = BRepBuilderAPI_MakeFace(wire).Face()
307
308         # Extrude along the edge direction
309         edge_length = math.sqrt((x2-x1)**2 + (y2-y1)**2 + (z2-z1)**2)
310         direction = gp_Vec(x2-x1, y2-y1, z2-z1)
311         direction.Normalize()
312         direction.Scale(edge_length)
313
314     elif thickness_direction == 'y':
315         # Create triangular cross-section weld along Y direction
316         p1 = gp_Pnt(x1, y1, z1)
317         p2 = gp_Pnt(x1 + weld_size, y1, z1)
318         p3 = gp_Pnt(x1, y1 +weld_size, z1)
319
320         edge1 = BRepBuilderAPI_MakeEdge(p1, p2).Edge()
321         edge2 = BRepBuilderAPI_MakeEdge(p2, p3).Edge()
322         edge3 = BRepBuilderAPI_MakeEdge(p3, p1).Edge()
323
324         wire = BRepBuilderAPI_MakeWire(edge1, edge2, edge3).Wire()
325         face = BRepBuilderAPI_MakeFace(wire).Face()
326
327         edge_length = math.sqrt((x2-x1)**2 + (y2-y1)**2 + (z2-z1)**2)
328         direction = gp_Vec(x2-x1, y2-y1, z2-z1)
329         direction.Normalize()
330         direction.Scale(edge_length)
331
332     prism = BRepPrimAPI_MakePrism(face, direction).Shape()
333     return prism
334
335
336
337
```

```
338 def create_i_section(length, width, depth, flange_thickness,
        web_thickness):
339     web_height = depth - 2 * flange_thickness
340 
341     bottom_flange = BRepPrimAPI_MakeBox(length, width, flange_thickness
            ).Shape()
342     top_flange = BRepPrimAPI_MakeBox(length, width, flange_thickness).
            Shape()
343     trsf = gp_Trsf()
344     trsf.SetTranslation(gp_Vec(0, 0, depth - flange_thickness))
345     top_flange_transform = BRepBuilderAPI_Transform(top_flange, trsf,
            True).Shape()
346 
347     web = BRepPrimAPI_MakeBox(length, web_thickness, web_height).Shape
            ()
348     trsf = gp_Trsf()
349     trsf.SetTranslation(gp_Vec(0, (width - web_thickness) / 2,
            flange_thickness))
350     web_transform = BRepBuilderAPI_Transform(web, trsf, True).Shape()
351 
352     i_section_solid = BRepAlgoAPI_Fuse(bottom_flange,
            top_flange_transform).Shape()
353     i_section_solid = BRepAlgoAPI_Fuse(i_section_solid, web_transform).
            Shape()
354 
355     return i_section_solid
356 
357 def create_end_batten(length, width, depth, column_distance):
358     batten = BRepPrimAPI_MakeBox(length, column_distance + width, depth
            ).Shape()
359     return batten
360 
361 def create_straight_lace(width, thickness, length):
362     # Create a horizontal lace plate
363     lace = BRepPrimAPI_MakeBox(length, width, thickness).Shape()
364     return lace
365 
366 def create_parallelogram_face(p1, p2, p3, thickness):
367     # Calculate the fourth point (Top-right)
```

```
368     vec1 = gp_Vec(p1, p2)  # Base vector
369     p4 = gp_Pnt(p3.XYZ() + vec1.XYZ())  # Top-right = Top-left + base
            vector
370
371     # Create edges
372     edge1 = BRepBuilderAPI_MakeEdge(p1, p2).Edge()
373     edge2 = BRepBuilderAPI_MakeEdge(p2, p4).Edge()
374     edge3 = BRepBuilderAPI_MakeEdge(p4, p3).Edge()
375     edge4 = BRepBuilderAPI_MakeEdge(p3, p1).Edge()
376
377     # Make wire
378     wire = BRepBuilderAPI_MakeWire(edge1, edge2, edge3, edge4).Wire()
379
380     # Make face
381     face = BRepBuilderAPI_MakeFace(wire).Face()
382
383     # Extrude the face into a solid
384     vec = gp_Vec(0, 0, thickness)
385     prism = BRepPrimAPI_MakePrism(face, vec).Shape()
386
387     return prism
388
389 %-------------------- end code ---------------
```

3.3.3 Explanation of the Code

- **Line 1–10: Library Imports**

  The script begins by importing necessary classes from the `OCC.Core` module. These include:

  - `gp_Pnt`, `gp_Vec`, `gp_Trsf` for defining geometry and applying transformations.
  - `BRepPrimAPI_MakeBox`, `BRepAlgoAPI_Fuse`, and `BRepBuilderAPI_Transform` for creating primitive shapes, performing boolean operations, and manipulating geometry.
  - `BRepBuilderAPI_MakeEdge`, `MakeWire`, and `MakeFace` for creating topological entities used in generating solid shapes like laces and welds.

- **Line 12–70: Utility Functions**

  A set of reusable utility functions are defined to handle geometric modeling operations:

  - `create_i_section()` generates a standard I-section using three box primitives (top flange, web, and bottom flange).
  - `create_corner_triangular_weld()` and `create_edge_weld()` define weld geometries using triangular faces extruded along a specified direction.
  - `create_parallelogram_face()` constructs diagonal lace plates by calculating four-point parallelogram faces and extruding them into 3D solids.

- **Line 72–130: Initialization of Parameters**

  This section sets up the core design parameters, including:

  - Total column height, width, and flange/web thickness for I-sections.
  - Batten depth and thickness for top and bottom transverse plates.
  - Lace dimensions (width, thickness), column spacing, and number of laces based on geometric spacing logic.

- **Line 132–190: Base Geometry Construction**

  Two I-sections are generated and spaced apart using geometric translation. Batten plates are placed at the top and bottom of the structure using `create_parallelogram_face()` and positioned using translation vectors. Triangular fillet welds are added at batten-column interfaces to simulate realistic welded connections.

- **Line 192–290: Diagonal Lace Generation**

  The diagonal lacing members are added between the two I-sections. In the case of:

  - **Laced Column:** Laces are alternated in a zig-zag pattern using a loop.
  - **Double-Laced Column:** Two lace plates are added per bay, forming an X-pattern. This increases stability and simulates real-world double lacing in steel structures.

  Each lace is generated by defining three corner points of a parallelogram and extruding it to form a solid.

- **Line 292–350: Weld Detailing**

  For every lace and batten connection, triangular welds are modeled using the previously defined weld functions. Each weld is placed and oriented properly depending on the plane (XZ, YZ, or XY) and location (top or bottom, left or right). This ensures the visual and structural completeness of the model.

- **Line 352–End: Final Assembly**

  All individual components — I-sections, battens, laces, and welds — are combined using the boolean fuse operation (`BRepAlgoAPI_Fuse`) to generate a single solid object. This object can be exported, rendered, or used for further operations like meshing or structural validation.

3.4 Task 1: Documentation

Insert your contribution towards Osdag Developer/ User manual. A sample is given below

3.4.1 Directory Structure

```
Osdag
├─ osdagMainPage.py
├─ Common.py
├─ ResourceFiles
│  ├─ images
│  ├─ last_designs
│  └─ ....
├─ design_type
├─ design_report
├─ cad
└─ gui
```

Program Start Calls The main entry point for the program is `osdagMainPage.py`. To start the program, open the Osdag folder and start the terminal with that path, execute the following command from the project root:

```
$ python osdagMainPage.py
```

Chapter 4

Internship Task 2 Title

4.1 Task 2: Problem Statement

Pyplots Development and Integration of Bending Moment Diagram(BMD) and Shear Force Diagram(SFD) in Flexural Member Module in Osdag

4.2 Task 2: Tasks Done

4.2.1 Development and Integration of BMD and SFD Plotting Feature

A major enhancement was implemented in the flexural design module of Osdag to allow users to visualize Bending Moment Diagrams (BMD) and Shear Force Diagrams (SFD) based on manually provided input data. This feature significantly improves user interaction and understanding of internal forces acting on a beam.

Objective: To provide an interactive and intuitive user interface within Osdag where users can enter tabular values of distance and corresponding bending moment or shear force, and generate clean, customizable plots directly from the UI.

Key Features Implemented

- **Modular Design of Plotting Window:** A custom PyQt-based dialog window, `PlotInputWidget`, was created using `QDialog`. It dynamically adapts based on the

plot type: "BMD" or "SFD". This modular approach makes the widget reusable in other modules if needed.

- **Data Input via Table Widget:** Users can input or paste data into a `QTableWidget` structured with two columns — Distance (m) and either BM (kNm) or SF (kN). The widget supports:
  - Manual entry of values,
  - Copy-paste functionality from spreadsheet tools like Microsoft Excel,
  - Dynamic row addition.
- **Real-Time Plotting using Matplotlib:** Upon clicking the "Plot" button, the diagram is rendered using `matplotlib`. Key aspects include:
  - A clean line plot with data markers and vertical dashed lines connecting each point to the X-axis.
  - Auto-labeling of the axis and consistent styling based on the diagram type.
  - Grid lines and horizontal zero-line for reference.
- **Interactive Hover with Coordinate Display:** A hover feature was implemented using `mpl_connect` which shows an annotation box near the cursor, displaying the interpolated X and Y values at the cursor's location on the line. This allows users to inspect values even between discrete data points.

Integration into Flexure Module

The plotting feature was integrated into the flexural member design interface by adding "Plot BMD" and "Plot SFD" buttons below the respective numerical input fields. These buttons launch the modal plot input dialogs. The design leverages Osdag's existing `options_list` architecture to inject these buttons cleanly into the layout.

Outcome

The final implementation provides users with a smooth, professional-grade experience for visualizing internal force diagrams, supports fast iteration via table-based input, and aligns with the UI/UX standards expected in engineering software.

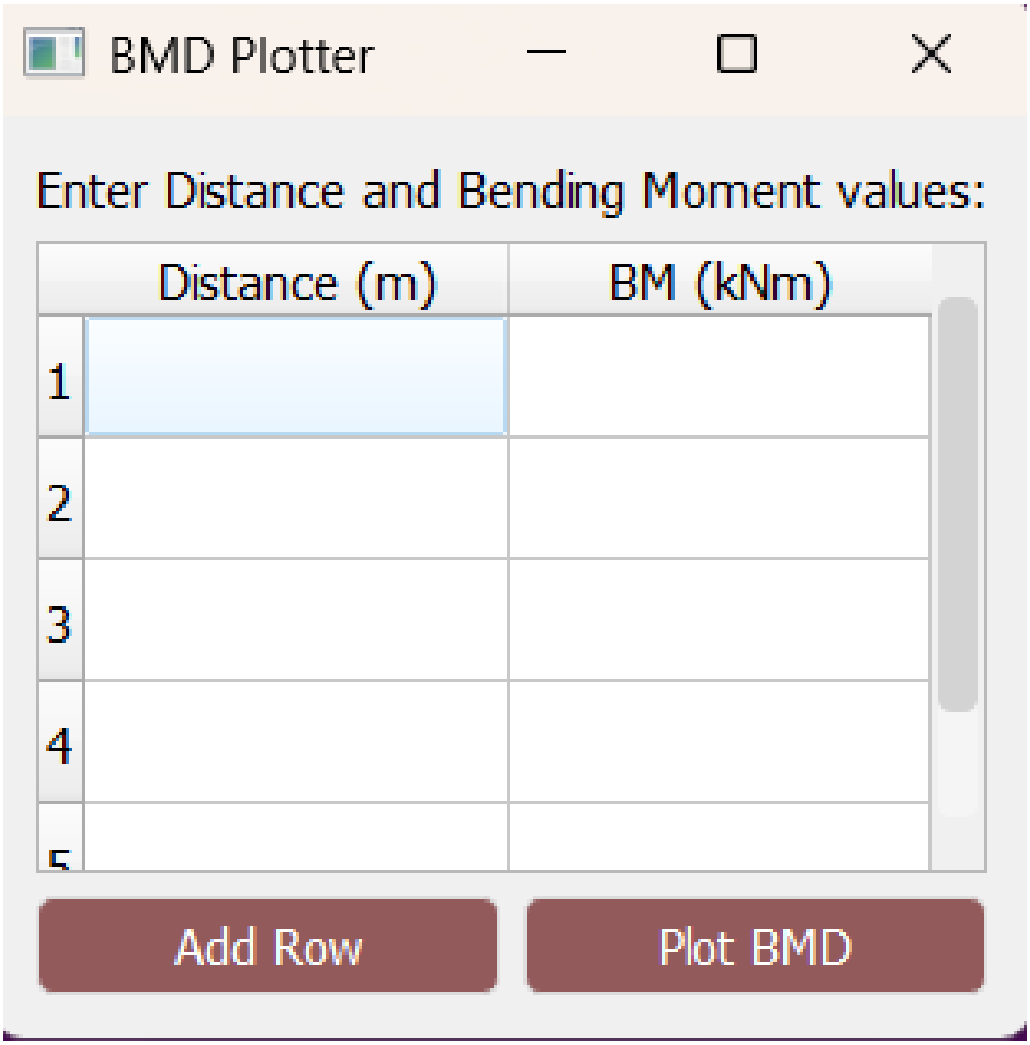


Figure 4.1: Bending Moment Diagram input dialog in Osdag

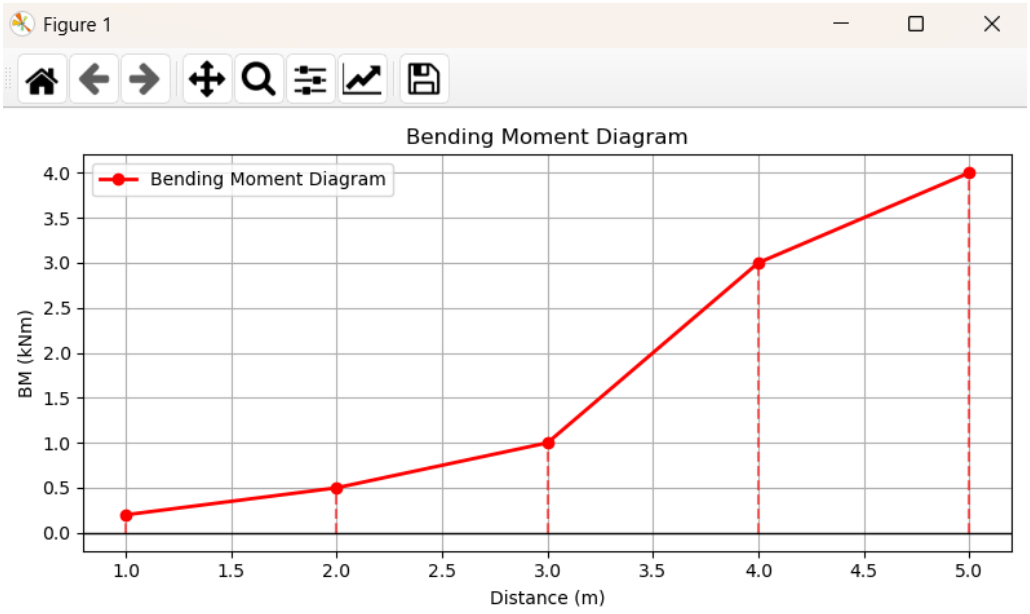


Figure 4.2: Plotted Bending Moment Diagram with interactive coordinate display

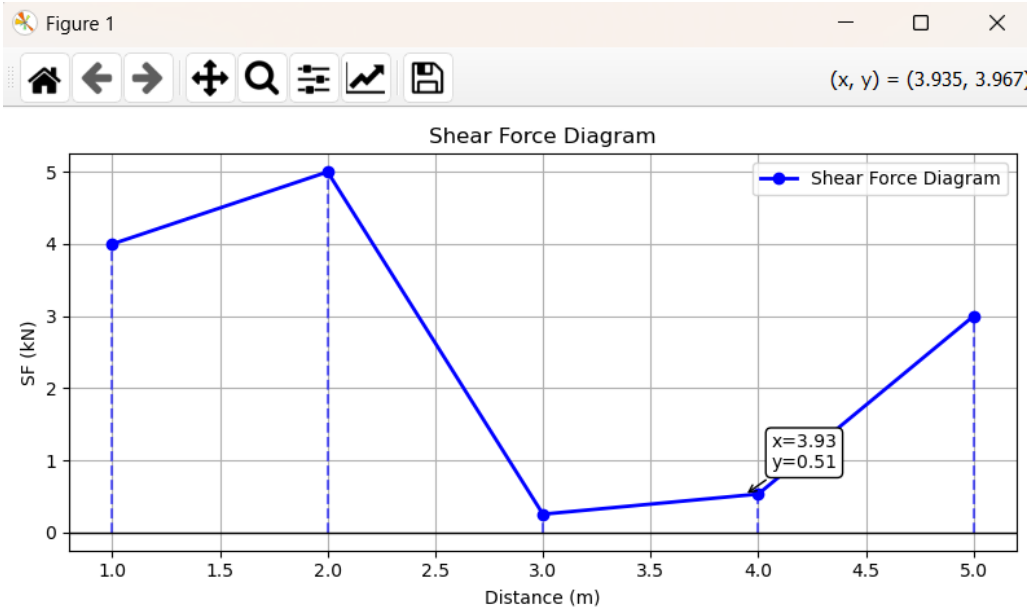


Figure 4.3: Plotted Shear Force Diagram showing vertical dashed lines

4.2.2 Python Code

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

```
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QApplication
from PyQt5.QtGui import QKeySequence
from PyQt5.QtWidgets import QDialog

from PyQt5.QtWidgets import (
    QWidget, QLabel, QPushButton, QVBoxLayout, QTableWidget,
       QTableWidgetItem,
    QHBoxLayout, QFileDialog, QHeaderView, QMessageBox
)
import matplotlib.pyplot as plt


class PlotInputWidget(QWidget):
    def __init__(self, plot_type="BM"):  # "BM" or "SF"
        super().__init__()
```

```
        self.plot_type = plot_type
        self.setWindowTitle(f"{'BMD' if plot_type == 'BM' else '
            SFD'} Plotter")

        self.layout = QVBoxLayout(self)

        # Setup Table
        self.table = QTableWidget(5, 2)
        self.table.setHorizontalHeaderLabels(["Distance (m)", "BM
            (kNm)" if plot_type == "BM" else "SF (kN)"])
        self.table.horizontalHeader().setSectionResizeMode(
            QHeaderView.Stretch)

        self.layout.addWidget(QLabel(f"Enter Distance and {'
            Bending Moment' if plot_type == 'BM' else 'Shear Force
            '} values:"))
        self.layout.addWidget(self.table)

        # Buttons
        btn_layout = QHBoxLayout()
        self.plot_btn = QPushButton(f"Plot {'BMD' if plot_type ==
            'BM' else 'SFD'}")
        # self.save_btn = QPushButton("Save as PNG")
        self.add_row_btn = QPushButton("Add Row")
        btn_layout.addWidget(self.add_row_btn)
        btn_layout.addWidget(self.plot_btn)
        # btn_layout.addWidget(self.save_btn)
        self.layout.addLayout(btn_layout)

        # Connections
        self.add_row_btn.clicked.connect(self.add_row)
        self.plot_btn.clicked.connect(self.plot_graph)
        # self.save_btn.clicked.connect(self.save_plot)
```

```
    def add_row(self):
        self.table.insertRow(self.table.rowCount())

    def get_data(self):
        x_vals = []
        y_vals = []

        for row in range(self.table.rowCount()):
            try:
                xi = float(self.table.item(row, 0).text())
                yi = float(self.table.item(row, 1).text())
                x_vals.append(xi)
                y_vals.append(yi)
            except (ValueError, AttributeError):
                continue

        if len(x_vals) < 2:
            QMessageBox.warning(self, "Invalid Input", "Please
               enter at least two valid data points.")
            return None, None

        # Sort by distance
        return zip(*sorted(zip(x_vals, y_vals)))

    def plot_graph(self):
        x, y = self.get_data()
        if x is None:
            return

        color = 'r' if self.plot_type == "BM" else 'b'
        title = "Bending Moment Diagram" if self.plot_type == "BM
           " else "Shear Force Diagram"
        ylabel = "BM (kNm)" if self.plot_type == "BM" else "SF (
           kN)"
```

```
        # plt.rcParams['home'] = 'none'
        fig, ax = plt.subplots(figsize=(8, 4))
        line, = ax.plot(x, y, color=color, linewidth=2, marker='o
            ', label=title)

        # Draw vertical dashed lines from each point to X-axis
        for xi, yi in zip(x, y):
            ax.vlines(xi, 0, yi, colors=color, linestyles='dashed
                ', alpha=0.6)

        ax.set_title(title)
        ax.set_xlabel("Distance (m)")
        ax.set_ylabel(ylabel)
        ax.grid(True)
        ax.axhline(0, color='black', linewidth=1)

        # Hover annotation
        annot = ax.annotate("", xy=(0,0), xytext=(15,15),
            textcoords="offset points",
                            bbox=dict(boxstyle="round", fc="w"),
                            arrowprops=dict(arrowstyle="->"))
        annot.set_visible(False)

        def update_annot(ind):
            index = ind["ind"][0]
            x_val = x[index]
            y_val = y[index]
            annot.xy = (x_val, y_val)
            annot.set_text(f"x={x_val:.2f}\ny={y_val:.2f}")
            annot.get_bbox_patch().set_alpha(0.9)

        def hover(event):
            if event.inaxes == ax and event.xdata is not None:
                x_mouse = event.xdata
```

```
                # Find two x-values around the mouse (for
                    interpolation)
                for i in range(len(x) - 1):
                    if x[i] <= x_mouse <= x[i + 1]:
                        # Linear interpolation
                        x1, x2 = x[i], x[i + 1]
                        y1, y2 = y[i], y[i + 1]

                        # Interpolated y value
                        slope = (y2 - y1) / (x2 - x1)
                        y_interp = y1 + slope * (x_mouse - x1)

                        # Update annotation
                        annot.xy = (x_mouse, y_interp)
                        annot.set_text(f"x={x_mouse:.2f}\ny={
                            y_interp:.2f}")
                        annot.set_visible(True)
                        fig.canvas.draw_idle()
                        return

                annot.set_visible(False)
                fig.canvas.draw_idle()


    fig.canvas.mpl_connect("motion_notify_event", hover)
    plt.tight_layout()
    plt.legend()
    plt.show()


def save_plot(self):
    x, y = self.get_data()
    if x is None:
```

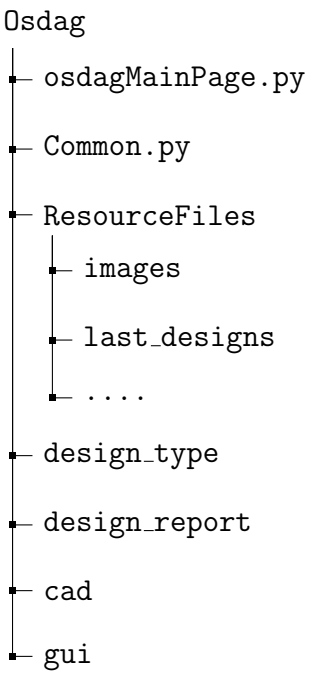
```
            return

        fname, _ = QFileDialog.getSaveFileName(self, "Save Plot",
            "", "PNG Files (*.png)")
        if fname:
            color = 'r' if self.plot_type == "BM" else 'b'
            title = "Bending Moment Diagram" if self.plot_type ==
                "BM" else "Shear Force Diagram"
            ylabel = "BM (kNm)" if self.plot_type == "BM" else "
               SF (kN)"

            plt.figure(figsize=(8, 4))
            plt.plot(x, y, color=color, linewidth=2, marker='o')
            for xi, yi in zip(x, y):
                plt.vlines(xi, 0, yi, colors=color, linestyles='
                   dashed', alpha=0.7)
            plt.axhline(0, color='black', linewidth=1)
            plt.title(title)
            plt.xlabel("Distance (m)")
            plt.ylabel(ylabel)
            plt.grid(True)
            plt.tight_layout()
            plt.savefig(fname)
            plt.close()

    def keyPressEvent(self, event):
        if event.matches(QKeySequence.Paste):
            clipboard = QApplication.clipboard()
            text = clipboard.text()
            rows = text.strip().split("\n")
            current_row = self.table.currentRow()
            current_col = self.table.currentColumn()

            for r_idx, row_data in enumerate(rows):
```

```
                columns = row_data.split("\t")
                for c_idx, cell in enumerate(columns):
                    target_row = current_row + r_idx
                    target_col = current_col + c_idx
                    # Expand if necessary
                    if target_row >= self.table.rowCount():
                        self.table.insertRow(self.table.rowCount
                            ())
                    item = QTableWidgetItem(cell.strip())
                    self.table.setItem(target_row, target_col,
                        item)
        else:
            super().keyPressEvent(event)
    def showEvent(self, event):
    # Clear table contents every time the widget is shown
        self.table.clearContents()
        self.table.setRowCount(5)  # Reset to default number of
            rows
        super().showEvent(event)
```

4.3 Task 2: Documentation

Insert your contribution towards Osdag Developer/ User manual. A sample is given below

4.3.1 Directory Structure



Program Start Calls The main entry point for the program is `osdagMainPage.py`. To start the program, open the Osdag folder and start the terminal with that path, execute the following command from the project root:

```
$ python osdagMainPage.py
```

Chapter 5

Conclusions

5.1 Tasks Accomplished

This section provides a concise summary of the major technical tasks completed during the project, focusing on both user interface enhancements and CAD development within the Osdag structural design software.

5.1.1 Integration of BMD and SFD Plotting Using PyPlot

- Designed and implemented a custom PyQt-based modal dialog for plotting **Bending Moment Diagrams (BMD)** and **Shear Force Diagrams (SFD)**.
- Enabled user input through an editable table interface, supporting both manual typing and direct copy-paste from Excel.
- Integrated `matplotlib` to dynamically plot the diagrams with:
  - Vertical dashed lines from each data point to the X-axis,
  - Interactive (x, y) tooltips on hover using real-time interpolation.
- Provided functionality to save the plotted diagrams as images.
- Embedded the feature cleanly into Osdag's flexural module by adding "Plot BMD" and "Plot SFD" buttons under respective input fields.

5.1.2 CAD Modeling of Laced and Double-Laced Welded Columns

- Developed parametric CAD generation logic for **laced** and **double-laced welded steel columns** using the PythonOCC 3D modeling library.
- Programmatically placed lacing bars between main column elements based on user-defined spacing, angle, and cross-sectional data.
- Supported both single-laced and double-laced configurations with customizable end connections and batten options.
- To support the Diagonal laces, triangular welding was done at the intersection with the i-section. Additonally for double laced column, welding was done at the intersection of the two laces as well.

5.2 Skills Developed

During the course of this fellowship, several key technical and professional skills were acquired and strengthened. These skills span across programming, software design, UI/UX development, and collaborative engineering practices.

Technical Skills

- **PyQt5 GUI Development:** Designed and integrated custom dialogs, widgets, buttons, and table interfaces into Osdag's user interface using PyQt5, enhancing the interactivity and usability of structural design tools.
- **CAD Automation with PythonOCC:** Developed detailed 3D models of laced and double-laced welded columns, implementing geometry creation, parametric control, and welding details using the PythonOCC (OpenCASCADE) library.
- **Software Integration and Debugging:** Integrated new features into Osdag's existing architecture while resolving compatibility issues, UI layout bugs, and logic conflicts during runtime.

- **Version Control and Codebase Navigation:** Became proficient in navigating a large open-source codebase, understanding module dependencies, and using Git for incremental development.

Professional Skills

- **Project Planning and Execution:** Understood feature requirements, translated them into actionable development tasks, and delivered them within specified timelines.
- **Collaboration and Communication:** Worked effectively as part of a development team, seeking feedback from mentors, coordinating UI design decisions, and presenting work in an organized and clear manner.
- **Attention to User Experience:** Focused on enhancing the usability of engineering tools through thoughtful interface design, error handling, and user-friendly workflows.

Chapter A

Appendix

A.1 Work Reports

Work Report: 19 May 2025 – 30 June 2025

Name: Harsh Gondal Summer Fellowship

Date	Day	Task	Hours
19th May 2025	Mon	- Installed OSDAG on Windows 10 x64 machine and did initial testing	8
20th May 2025-23rd May 2025	Tue-Fri	Got assigned with first task for laced column welded, started to develop the cad.	32
24th May 2025-30th May 2025	Sat-Fri	Worked on understanding the osdag architecture with help manual given and analysed files which were related to the triangular welds required for the task. Started building the cad according to the given schema diagram given with proper positioning of the diagonal laces .	48
31st May 2025-5th June 2025	Sat-Thr	Completed the single laced welded column, the welds were same as previously used in osdag. Did some minor changes as instructed by the mentor. Got assigned the next task for double laced welded column	40
6th June – 12th Jun 2025	Fri–Thr	Started developing the double laced welded column, there were few thing s which were different form single laced, so started developing cad according to that.	48
13th June- 18th June 2025	Fri-Wed	Completed the double laced welded column according to the schema diagram given and got assigned with pyplots task.	40
19th June- 25th June 2025	Thr-Wed	Started working on integrating pyplot feature of bmd and sfd in flexural module. First implemented code locally to print the pyplots. After that started integrating it into the module.	48
26th June – 28th June 2025	Thru-Sat	Worked on the integration of pyplot feature in flexural module along with creating suitable ui button for it and integrating many options like saving the image, scaling the image, manually inputting multiple values at a time.	24
28th June 2025-30th June 2025	Sat- Mon	Created Pull requests for both the tasks and resolved issues raised in that.	8

Total hours – 296 hrs

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.