FOSSEE Winter Internship Report

On

Development of CADs for Osdag

Submitted by

Aryan Gupta

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

Manipal Institute of Technology, Manipal

Manipal Academy of Higher Education

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

February 3, 2025

Acknowledgments

I am deeply grateful to everyone who supported and guided me throughout my internship project. This experience has been invaluable, and I would like to express my sincere appreciation to several individuals and organizations who made this journey possible.

First and foremost, I would like to thank the Osdag team, particularly Ajmal Babu M. S., Ajinkya Dahale, and Parth Karia, for their continuous guidance, patience, and technical support during my internship. Their expertise and collaborative spirit were instrumental in helping me navigate the complexities of the project.

I am particularly indebted to Prof. Siddhartha Ghosh from the Department of Civil Engineering at IIT Bombay, the Principal Investigator of the Osdag project, for providing me with this remarkable opportunity to contribute to meaningful research.

My gratitude also extends to Prof. Kannan M. Moudgalya, the FOSSEE Project Investigator from the Department of Chemical Engineering at IIT Bombay, whose leadership and vision have been crucial in driving innovative educational initiatives.

I am thankful to FOSSEE Managers Usha Viswanathan and Vineeta Parmar, along with their entire team, for creating a supportive and dynamic work environment that fostered learning and professional growth.

Special acknowledgement is due to the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India, for their financial support and commitment to advancing educational technology and research.

Lastly, I would like to thank my colleagues who worked alongside me during this internship, sharing knowledge, providing feedback, and creating a collaborative atmosphere that made this experience both educational and enjoyable.

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.

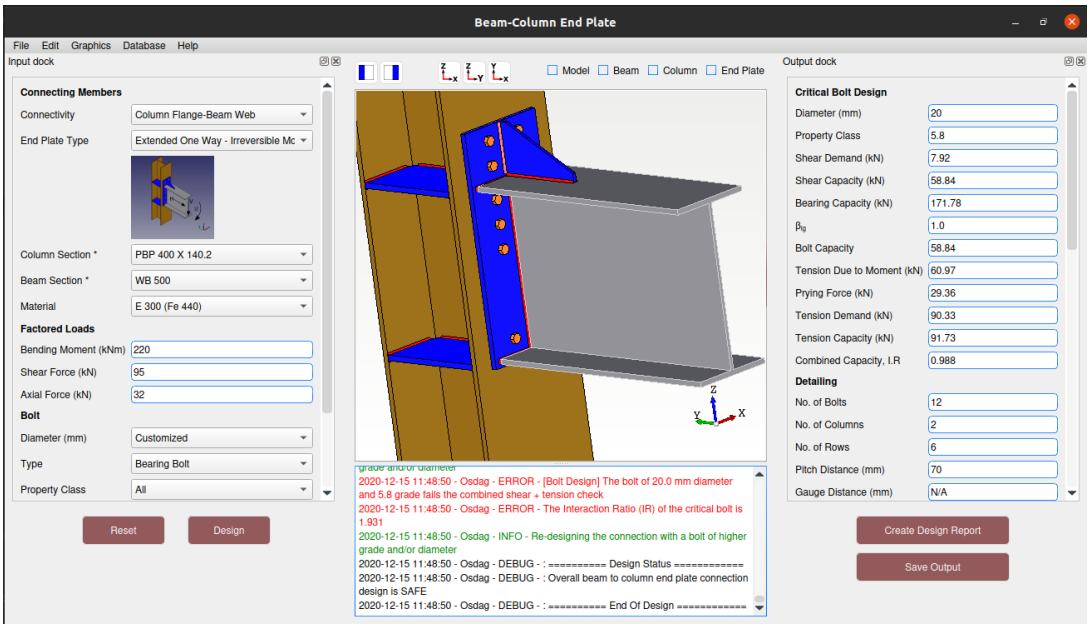

Figure 1.2: Osdag GUI

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 task was to create a 3D model of a steel portal frame based on the provided figure. An incomplete portal_frame.py script was given to generate a parametric model of the portal frame, where the dimensions of the frame components are adjustable. The final model was to be visualized using the OCC library.

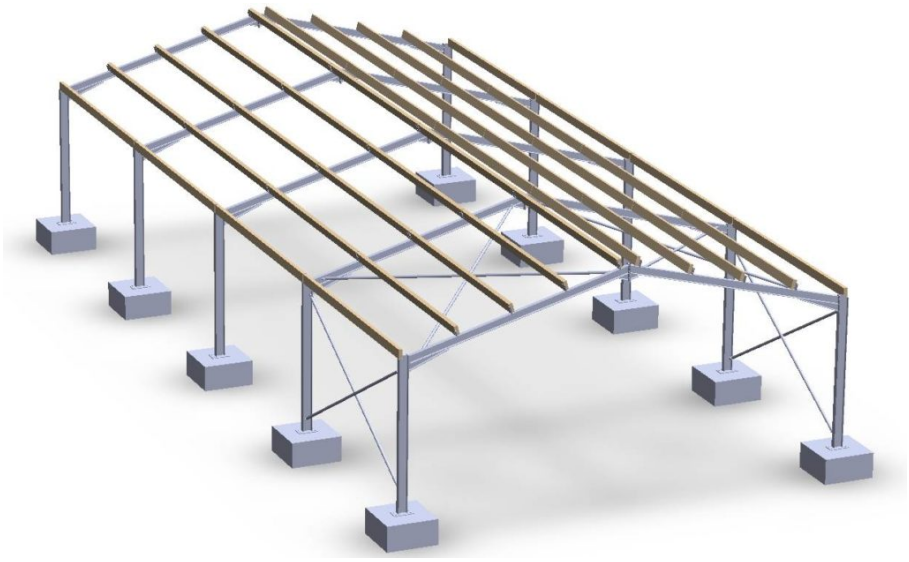

Figure 2.1: Structure of a simple portal frame with vertical columns, inclined rafters, and horizontal purlins.

2.2 Tasks Done

- **Libraries used:** The script uses the following libraries:
  - **Open Cascade Technology (OCCT):** A powerful CAD kernel for 3D modeling and computation.
  - **Geometric primitives and transformations:** `gp_Vec`, `gp_Trsf`, `gp_Ax1`, `gp_Pnt`, `gp_Dir`
  - **Shape creation:** `BRepPrimAPI_MakeBox` (To create box shapes)
  - **Boolean operations:** `BRepAlgoAPI_Fuse` (To perform Boolean operations, e.g., fusing shapes)
  - **Transformations:** `BRepBuilderAPI_Transform` (To apply geometric transformations to shapes)
  - **STEP file export:** `STEPControl_Writer`, `STEPControl_AsIs` (To export shapes to STEP files)
  - **Visualization:** `init_display` (For displaying shapes using a simple GUI)
  - **Mathematical operations:** `math` (For mathematical operations such as trigonometric functions)

Listing 2.1: Library imports

```
%--------------------begin code-------------
from OCC.Core.gp import gp_Vec, gp_Trsf, gp_Ax1, gp_Pnt, 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.Core.STEPControl import STEPControl_Writer,
    STEPControl_AsIs
from OCC.Display.SimpleGui import init_display
import math
%-------------------- end code ---------------
```

- **Columns:** Create vertical columns using the create_i_section function.
- **Rafters:** Create inclined rafters using the create_i_section function to connect the tops of the columns.

Listing 2.2: Complete create_i_section function for columns and rafters

```
%--------------------begin code-------------
def create_i_section(length, width, depth, flange_thickness,
    web_thickness):
    # Calculate web height
    web_height = depth - 2 * flange_thickness

    # Create bottom flange
    bottom_flange = BRepPrimAPI_MakeBox(length, width,
        flange_thickness).Shape()

    # Create top flange
    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()

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

    # Fuse flanges and web to create I-section
    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
    %-------------------- end code ---------------
```

- **Purlins:** Add horizontal purlins at intervals along the length of the rafters using the create_rectangular_prism function.

- **Inclination Angle:** The angle of inclination for the rafters should be a parametric input.

Listing 2.3: Complete create_purlin_layout function for creation and proper placement of purlins

```
%--------------------begin code-------------
def create_purlin_layout(num_purlins, purlin_width, purlin_height,
    purlin_depth, rafter_angle):
    # Create purlin shape
    purlin = BRepPrimAPI_MakeBox(purlin_width, purlin_depth,
        purlin_height).Shape()

    # Calculate roof rise based on rafter angle
    roof_rise = (8400 / 2) * math.tan(math.radians(rafter_angle)) #
        8400/2
    purlins = None

    # Loop to position each purlin
    for i in range(num_purlins):
        x = i * (8000 - purlin_width) / (num_purlins - 1)
        if i < num_purlins / 2:
            z = column_height + roof_rise - (num_purlins / 2 - i) *
                (roof_rise / (num_purlins / 2))
        else:
            z = column_height + roof_rise - (i - num_purlins / 2 +
                1) * (roof_rise / (num_purlins / 2))

        # Translate purlin to correct position
        trsf = gp_Trsf()
        trsf.SetTranslation(gp_Vec(x, 0, z))
        purlin_instance = BRepBuilderAPI_Transform(purlin, trsf,
            True).Shape()

        # Fuse purlins together
        if purlins is None:
            purlins = purlin_instance
        else:
            purlins = BRepAlgoAPI_Fuse(purlins, purlin_instance).
                Shape()
```

```
28
29      return purlins
30  %-------------------- end code ---------------
```

- **Assumptions:** Standard dimensions for the components are assumed, but they should be adjustable through variables.

Listing 2.4: create_portal_frame function for creating the complete portal frame using the above functions

```
1  %--------------------begin code-------------
2  def create_portal_frame(column_length, column_width, column_height,
       column_flange_thickness, column_web_thickness,
3                          rafter_width, rafter_depth,
                               rafter_flange_thickness,
                               rafter_web_thickness, rafter_angle,
                               num_rafters,
4                          purlin_width, purlin_height, purlin_depth):
5      # Create column I-sections
6      column = create_i_section(column_length, column_width,
           column_height, column_flange_thickness, column_web_thickness
           )
7      columns = None
8      column_spacing = purlin_depth / (num_columns_per_side - 1)
9
10     # Loop to position columns
11     for i in range(num_columns_per_side):
12         y = i * column_spacing
13         x_left = 0
14         trsf_left = gp_Trsf()
15         trsf_left.SetTranslation(gp_Vec(x_left, y, 0))
16         column_instance_left = BRepBuilderAPI_Transform(column,
               trsf_left, True).Shape()
17
18         x_right = 8000
19         trsf_right = gp_Trsf()
20         trsf_right.SetTranslation(gp_Vec(x_right, y, 0))
21         column_instance_right = BRepBuilderAPI_Transform(column,
               trsf_right, True).Shape()
22
```

```
        # Fuse columns together
        if columns is None:
            columns = BRepAlgoAPI_Fuse(column_instance_left,
                column_instance_right).Shape()
        else:
            columns = BRepAlgoAPI_Fuse(columns,
                column_instance_left).Shape()
            columns = BRepAlgoAPI_Fuse(columns,
                column_instance_right).Shape()

    # Create purlins layout
    purlins = create_purlin_layout(num_purlins, purlin_width,
        purlin_height, purlin_depth, rafter_angle)

    # Calculate rafter length and create I-section
    rafter_length = 8000 / 2 / math.cos(math.radians(rafter_angle))
    rafter = create_i_section(rafter_length, rafter_width,
        rafter_depth, rafter_flange_thickness, rafter_web_thickness)
    rafters = None
    rafter_spacing = purlin_depth / (num_rafters - 1)

    # Loop to position rafters
    for i in range(num_rafters):
        y = i * rafter_spacing

        # Left rafter
        trsf_left = gp_Trsf()
        trsf_left.SetTranslation(gp_Vec(0, y, column_height))
        rafter_instance_left = BRepBuilderAPI_Transform(rafter,
            trsf_left, True).Shape()

        # Rotate left rafter upward
        rotation_left = gp_Trsf()
        rotation_left.SetRotation(gp_Ax1(gp_Pnt(0, y, column_height
            ), gp_Dir(0, -1, 0)), math.radians(rafter_angle))
        rafter_instance_left = BRepBuilderAPI_Transform(
            rafter_instance_left, rotation_left, True).Shape()

        # Right rafter
```

```
        trsf_right = gp_Trsf()
        trsf_right.SetTranslation(gp_Vec(4000, y, column_height))
        rafter_instance_right = BRepBuilderAPI_Transform(rafter,
            trsf_right, True).Shape()

        # Rotate right rafter downward
        rotation_right = gp_Trsf()
        rotation_right.SetRotation(gp_Ax1(gp_Pnt(4000, y,
            column_height), gp_Dir(0, -1, 0)), math.radians(-
            rafter_angle))
        rafter_instance_right = BRepBuilderAPI_Transform(
            rafter_instance_right, rotation_right, True).Shape()

        # Shift right rafter to align with purlins
        alignment_shift = gp_Trsf()
        alignment_shift.SetTranslation(gp_Vec(0, 0, (8000 / 2) *
            math.tan(math.radians(rafter_angle))))
        rafter_instance_right = BRepBuilderAPI_Transform(
            rafter_instance_right, alignment_shift, True).Shape()

        # Fuse rafters together
        if rafters is None:
            rafters = BRepAlgoAPI_Fuse(rafter_instance_left,
                rafter_instance_right).Shape()
        else:
            rafters = BRepAlgoAPI_Fuse(rafters,
                rafter_instance_left).Shape()
            rafters = BRepAlgoAPI_Fuse(rafters,
                rafter_instance_right).Shape()

    # Combine all components into the portal frame
    portal_frame_solid = BRepAlgoAPI_Fuse(columns, purlins).Shape()
    portal_frame_solid = BRepAlgoAPI_Fuse(portal_frame_solid,
        rafters).Shape()

    return portal_frame_solid
%-------------------- end code ---------------
```

- **Save to STEP file:** This function saves the generated shape to a STEP file. The

shape is transferred to a STEP writer and written to the specified file.

Listing 2.5: save_to_step function for saving the created shape to a STEP file

```
%--------------------begin code-------------
 def save_to_step(shape, filename):
    # Save the shape to a STEP file
    step_writer = STEPControl_Writer()
    step_writer.Transfer(shape, STEPControl_AsIs)
    status = step_writer.Write(filename)
    return status == 1
%-------------------- end code ---------------
```

Listing 2.6: main function which takes parameters set by the user and displays the result

```
%--------------------begin code-------------
if __name__ == "__main__":
    # Define the parameters for the portal frame
    column_height = 4000
    column_length = 100
    column_width = 200
    column_flange_thickness = 15
    column_web_thickness = 5
    num_columns_per_side = 6

    num_rafters = 10
    rafter_width = 200
    rafter_depth = 100
    rafter_flange_thickness = 15.01
    rafter_web_thickness = 4.67
    rafter_angle = 30


    num_purlins = 9
    purlin_width = 125  #x-axis
    purlin_height = 175
    purlin_depth = 6000.0

    # Create portal frame
    portal_frame = create_portal_frame(column_length, column_width,
        column_height, column_flange_thickness,
```

```
        column_web_thickness,
                                            rafter_width, rafter_depth,
                                                rafter_flange_thickness
                                                , rafter_web_thickness,
                                                rafter_angle,
                                                num_rafters,
                                            purlin_width, purlin_height
                                                , purlin_depth)

    display, start_display, add_menu, add_function_to_menu =
        init_display()
    display.DisplayShape(portal_frame, update=True)
    display.FitAll()

    # Save as a STEP file

    filename = "portal_frame.stp"
    if save_to_step(portal_frame, filename):
        print(f"Successfully saved the portal frame to {filename}")
    else:
        print(f"Failed to save the portal frame to {filename}")

    start_display()

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

2.3 Calculations and Formulas

- **Roof Rise:** This calculates the vertical rise of the roof at the midpoint.

$$\text{roof\_rise} = \left(\frac{8400}{2}\right) \times \tan(\text{radians}(\text{rafter\_angle}))$$

- **Purlin Positioning:** Purlins are evenly spaced along the width of the roof. The z-coordinate of each purlin is calculated based on its position relative to the roof rise.

$$x = i \times \frac{8000 - \text{purlin\_width}}{\text{num\_purlins} - 1}$$

$$\text{if } i < \frac{\text{num\_purlins}}{2} :$$

$$z = \text{column\_height} + \text{roof\_rise} - \left(\frac{\text{num\_purlins}}{2} - i\right) \times \left(\frac{\text{roof\_rise}}{\text{num\_purlins}/2}\right)$$

$$\text{else :}$$

$$z = \text{column\_height} + \text{roof\_rise} - \left(i - \frac{\text{num\_purlins}}{2} + 1\right) \times \left(\frac{\text{roof\_rise}}{\text{num\_purlins}/2}\right)$$

where i=0 refers to the first purlin, i=1 refers to the second purlin and so on.

- **Rafter Length:** This calculates the length of each rafter based on the roof angle.

$$\text{rafter\_length} = \frac{8000}{2\cos(\text{radians}(\text{rafter\_angle}))}$$

- **Column Spacing:** This calculates the space between two consecutive columns based on the depth of the purlin and the number of columns entered by the user.

$$\text{column\_spacing} = \frac{\text{purlin\_depth}}{\text{num\_columns\_per\_side} - 1}$$

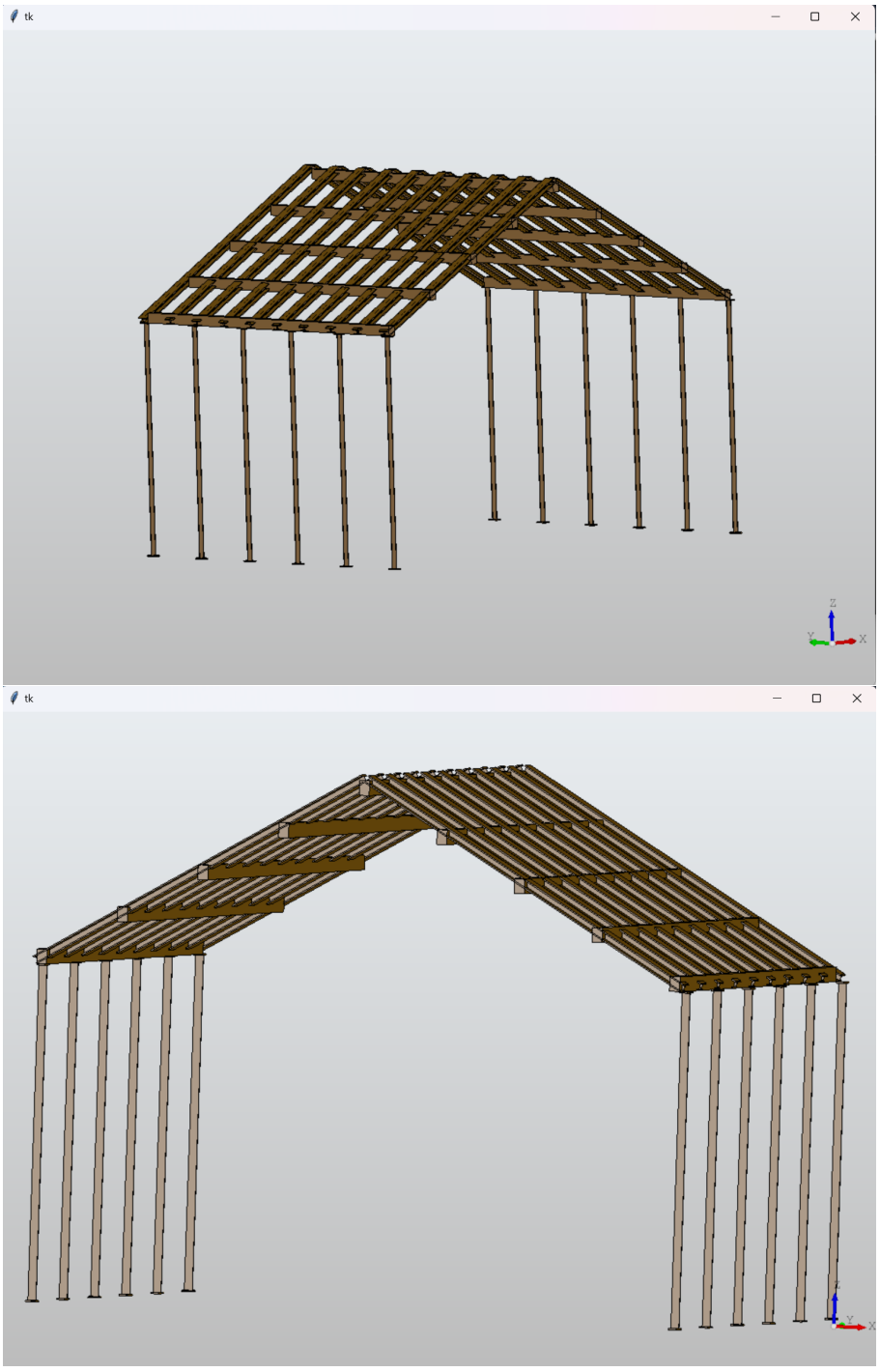

Figure 2.2: Final Portal Frame

Chapter 3

Internship Task 1: CAD for Columns with known support conditions

3.1 Task 1: Problem Statement

Display CADs for Columns with known support conditions in the CAD window.

3.2 Task 1: Tasks Done

The CAD displayed should have optimum dimensions that are calculated based on the user input. The column needs to use the existing ISection.py file to accurately create and place the column vertically.

Methodology

1. **Geometry Construction:** The process starts with defining points, which are used to create edges. These edges form a continuous boundary (wire), which is then converted into a planar surface (face) representing the cross-section of the member.

2. **Extrusion:** The cross-sectional face is extruded along a specific direction and length to create a 3D prism, representing the volume of the structural member.

3. **Feature Integration:** Optional features like notches are incorporated by calculating their position relative to the main geometry. The notch is modeled separately

and subtracted from the prism using Boolean operations, creating voids or recesses in the final geometry.

4. **Coordinate System Definition:** A local coordinate system ensures consistent orientation. For example, the z-axis is aligned vertically for columns, while other axes define horizontal directions.

5. **Section Placement:** The cross-section is placed at a specified origin and aligned according to the defined directions before extrusion, ensuring accurate positioning in 3D space.

3.3 Task 1: Python Codes

3.3.1 create_model() in osdag/cad/items/ISection.py

This function takes a set of defined points and transforms them into a 3D solid model. It starts by generating edges, assembling them into a wire, and then constructing a face from that wire. The face is extruded along a specified direction to create a prism. If a notch feature is provided, it positions and integrates the notch by subtracting its geometry from the solid using a boolean cut.

Python Code

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

Listing 3.1: create_model()

```
1  %--------------------begin code-------------
2      def create_model(self):
3
4          edges = makeEdgesFromPoints(self.points)
5          wire = makeWireFromEdges(edges)
6          aFace = makeFaceFromWire(wire)
7          extrudeDir = self.length * self.wDir  # extrudeDir is a numpy
               array
8          prism = makePrismFromFace(aFace, extrudeDir)
9
10         if self.notchObj is not None:
```

```
11              uDir = numpy.array([-1.0, 0.0, 0])
12              wDir = numpy.array([0.0, 1.0, 0.0])
13              shiftOri = self.D / 2.0 * self.vDir + self.notchObj.width /
                    2.0 * self.wDir + self.B / 2.0 * -self.uDir  # + self.
                   notchObj.width* self.wDir + self.T/2.0 * -self.uDir
14              origin2 = self.sec_origin + shiftOri
15
16              self.notchObj.place(origin2, uDir, wDir)
17
18              notchModel = self.notchObj.create_model()
19              prism = BRepAlgoAPI_Cut(prism, notchModel).Shape()
20
21          return prism
22  %-------------------- end code ---------------
```

Explanation of the Code

- Lines 2-6: Transforms a set of points into basic geometric elements by creating edges, linking them into a continuous wire, and forming a face that represents the cross-sectional profile.
- Lines 7-8: Extrudes the face along a specified direction (scaled by the structural member's length) to form a solid prism.
- Lines 10-18: If a notch feature is defined, computes its precise position using vector operations, positions the notch accordingly, creates the notch model, and removes the corresponding material from the prism through a boolean cut.
- Return: Provides the final 3D model with integrated features (if any notch is present).

3.3.2 createColumnGeometry() in osdag/cad/CompressionMembers/column.py

This function defines the spatial orientation for a column, ensuring that it remains vertical along the z-axis. It sets the local coordinate system with specific unit vectors and places

the cross-sectional geometry at a defined origin. Once placed, the column model is created based on this orientation, ensuring consistent alignment for the final structural member.

Python Code

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

Listing 3.2: createColumnGeometry()

```
1  %--------------------begin code-------------
2      def createcolumnGeometry(self):
3          """
4          Ensures the column is always vertical along the z-axis.
5
6          :return: Geometric orientation of this component
7          """
8          # The origin of the column
9          columnOriginL = numpy.array([0.0, 0.0, 0.0])
10
11         # Set the column's local u-direction and w-direction
12
13         uDir = numpy.array([1.0, 0.0, 0.0])  # Along x-axis
14         wDir = numpy.array([0.0, 0.0, 1.0])  # Along z-axis (vertical)
15
16         # Place the section at the specified origin with the
               orientation defined
17         self.sec.place(columnOriginL, uDir, wDir)
18
19         # Create the column model based on the section and orientation
20         self.columnModel = self.sec.create_model()
21 %-------------------- end code ---------------
```

Explanation of the Code

- Lines 9-14: Establishes the column's local coordinate system by defining an origin and assigning unit vectors for the horizontal (u-direction) and vertical (w-direction) axes.
- Lines 16-19: Places the cross-sectional geometry at the established origin with the defined orientation to ensure the column stands vertically (aligned along the z-axis).

- Final Step: Generates the complete column model based on the oriented cross-section, ensuring a consistent vertical alignment throughout the component.

3.3.3 createColumnInFrameCAD() in osdag/cad/common_logic.py

The function starts by retrieving the module's column designation and printing it. It then checks if the designation belongs to hollow sections (RHS/SHS), circular hollow sections (CHS), or rolled sections (beams/columns).

For RHS or SHS, it creates the corresponding section property object, extracts key dimensions (flange width, depth, etc.), and generates a rectangular hollow section model.

For CHS, it calculates the radius, prints the required parameters, and creates a circular hollow section model.

For rolled sections, it attempts to create either a beam or a column model based on the designation and extracts detailed section properties such as web thickness, flange thickness, depth, flange width, and radii.

It then creates an ISection object using these parameters and builds a compression member CAD model.

Finally, the function generates the 3D model and returns the section object.

Python Code

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

Listing 3.3: Beam-to-Column Connection Design in Osdag

```
%--------------------begin code-------------
    def createColumnInFrameCAD(self):
        """
            :return: The calculated values/parameters to create 3D CAD
               model of individual components.
        """

        Col = self.module_class
        print("COL_DESIGINATION :",Col.result_designation)

        if 'RHS' in Col.result_designation or 'SHS' in Col.
           result_designation:  # hollow sections 'RHS and SHS'
            if 'RHS' in Col.result_designation:
```

```
                result = RHS(designation=Col.result_designation,
                    material_grade=Col.material)
            else:
                result = SHS(designation=Col.result_designation,
                    material_grade=Col.material)
            Col.section_property = result
            print(f"Parameter L (flange width): {float(Col.
                section_property.flange_width)}")
            print(f"Parameter W (depth): {float(Col.section_property.
                depth)}")
            print(f"Parameter H (length/height): {float(Col.length_zz)}
                ")
            print(f"Parameter T (flange thickness): {float(Col.
                section_property.flange_thickness)}")
            sec = RectHollow(L=float(Col.section_property.flange_width)
                , W=float(Col.section_property.depth),
                             H=float(Col.length_zz), T=float(Col.
                                 section_property.flange_thickness))
            col = CompressionMemberCAD(sec)
            sec=sec.create_model()
            col.create_3DModel()
        elif 'CHS' in Col.result_designation:  # CHS
            result = CHS(designation=Col.result_designation,
                material_grade=Col.material)
            Col.section_property = result
            print(f"Parameter r (radius): {float(Col.section_property.
                depth) / 2}")
            print(f"Parameter T (thickness): {float(Col.
                section_property.flange_thickness)}")
            print(f"Parameter H (height/length): {float(Col.length_zz)}
                ")
            sec = CircularHollow(r=float(Col.section_property.depth) /
                2, T=float(Col.section_property.flange_thickness),
                                 H=float(Col.length_zz))
            col = CompressionMemberCAD(sec)
            sec=sec.create_model()
            col.create_3DModel()
        else: # Beams and Columns (rolled sections)
            try:
```

```
                result = Beam(designation=Col.result_designation,
                    material_grade=Col.material)
            except:
                result = Column(designation=Col.result_designation,
                    material_grade=Col.material)
            Col.section_property = result

            column_tw = float(Col.section_property.web_thickness)
            print(f"column_tw (Web Thickness): {column_tw}")

            column_T = float(Col.section_property.flange_thickness)
            print(f"column_T (Flange Thickness): {column_T}")

            column_d = float(Col.section_property.depth)
            print(f"column_d (Depth): {column_d}")

            column_B = float(Col.section_property.flange_width)
            print(f"column_B (Flange Width): {column_B}")

            column_R1 = float(Col.section_property.root_radius)
            print(f"column_R1 (Root Radius): {column_R1}")

            column_R2 = float(Col.section_property.toe_radius)
            print(f"column_R2 (Toe Radius): {column_R2}")

            column_alpha = 94  # Todo: connect this. Waiting for danish
                to give variable
            column_length = float(Col.length_zz)

            sec = ISection(B=column_B, T=column_T, D=column_d, t=
                column_tw, R1=column_R1, R2=column_R2,
                                alpha=column_alpha, length=column_length,
                                    notchObj=None)
            col = CompressionMemberCAD(sec)
            sec=sec.create_model()

            col.create_3DModel()

        return sec
```

```
72 %-------------------- end code ---------------
```

Explanation of the Code

Branch for Hollow Sections (RHS/SHS)

- Lines 7–8: Retrieves the module's column designation and prints it to indicate the type of section.
- Lines 10–19: Checks for hollow sections ('RHS' or 'SHS'), creates the appropriate section property object, and prints key parameters such as flange width, depth, and thickness.
- Lines 20–24: Constructs a rectangular hollow section using the extracted dimensions, creates a compression member CAD object, then builds and displays the 3D model.

Branch for Circular Hollow Sections (CHS)

- Lines 25–30: Identifies a CHS designation, creates its section property object, and prints parameters like radius, thickness, and height.
- Lines 31–35: Uses these parameters to instantiate a circular hollow section, creates a compression member CAD, and builds the 3D model.

Branch for Rolled Sections (Beams and Columns)

- Lines 36–41: In the absence of hollow section designations, the code attempts to create a beam or column section, extracting detailed properties such as web thickness, flange thickness, depth, flange width, and radii.
- Lines 42–65: Prints each of these measured parameters, assigns a constant value for an additional geometric parameter (alpha), and creates an ISection object.
- Lines 66–67: Instantiates a compression member CAD object with the ISection, generates the 3D model, and completes the construction of the geometric representation.
- Return: The final section model is returned for further use.

Chapter 4

Internship Task 2: Creating CADs for Struts in Trusses

4.1 Task 2: Problem Statement

Create and display CADs for the 'Struts in Trusses' submodule. Specifically, create separate CADs for each of the three section profiles: Angles, Back to Back Angles-Same side of gusset and Back to Back Angles-Opposite side of gusset.

4.2 Task 2: Tasks Done

Code to create a single angles is available in osdag/cad/items/angle.py. Similarly, the code to create an individual gusset plate is given in osdag/cad/items/Gasset_plate.py. They need to be used to create CADs for the Back to Back Angles section profiles as well.

4.2.1 Methodology

1. **Component Instantiation:** Individual geometric objects representing two angles and two gusset plates are created, each with specific design parameters. This modular approach ensures that each element is defined separately before integration.

2. **Spatial Organization:** A strategic placement strategy is employed, where orientation vectors and offsets are calculated to accurately position each component.

This involves determining a central reference point, setting rotational directions, and applying offsets to ensure the angles are arranged back-to-back with a defined spacing, while gussets are precisely positioned at each end.

3. **Transformation Consistency:** The logic emphasizes maintaining consistent coordinate systems and orientation for all parts. This guarantees that the relative positioning of the angles and gusset plates aligns correctly in the overall assembly without misalignments or gaps.

4. **Model Integration:** Finally, Boolean fusion operations are used to combine the individual models into a unified structure. This step consolidates the separate elements, ensuring that they interact as a single component in the final 3D model.

4.3 Task 2: Python Codes

4.3.1 class BackToBackAnglesWithGussetsSameSide in osdag/-cad/items/double_angles.py

The code defines a class that constructs two back-to-back angle components and two gusset plates based on provided dimensions and orientation parameters.
It sets up the individual components by instantiating separate objects for the angles and gussets, storing key design dimensions and calculated offsets.
A placement method is used to align the angles with specific rotations and to position the gusset plates at calculated end locations relative to a central reference.
Finally, the individual 3D models of the components are fused together using boolean operations to create the final integrated structure.

Python Code

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

Listing 4.1: class BackToBackAnglesWithGussetsSameSide

```
1 %--------------------begin code-------------
2 class BackToBackAnglesWithGussetsSameSide:
3     def __init__(self, L, A, B, T, R1, R2, gusset_L, gusset_H, gusset_T
      , gusset_degree, spacing=2):
```

```
        """
        Creates back-to-back angles with gusset plates automatically
            positioned at the ends
        Args:
            L: Length of the angle
            A: Vertical leg length
            B: Horizontal leg length
            T: Thickness
            R1, R2: Corner radii
            gusset_L: Length of gusset plate
            gusset_H: Height of gusset plate
            gusset_T: Thickness of gusset plate
            gusset_degree: Angle of gusset plate
            spacing: Gap between angles
        """
        self.angle1 = Angle(L, A, B, T, R1, R2)
        self.angle2 = Angle(L, B, A, T, R1, R2)
        self.gusset1 = GassetPlate(gusset_L, gusset_H, gusset_T,
            gusset_degree)
        self.gusset2 = GassetPlate(gusset_L, gusset_H, gusset_T,
            gusset_degree)
        self.spacing = spacing

        # Store dimensions for later use
        self.L = L
        self.A = A  # Vertical leg length
        self.B = B  # Horizontal leg length
        self.T = T  # Angle thickness
        self.gusset_L = gusset_L
        self.gusset_H = gusset_H
        self.gusset_T = gusset_T

        # Calculate Z-offsets based on angle length
        # Position plates at the ends with a small margin
        margin = gusset_L/2  # Adjust this value to fine-tune the end
            position
        self.gusset1_offsets = numpy.array([0, 0, margin])  # At the
            start
        self.gusset2_offsets = numpy.array([0, 0, L - margin])  # At
```

```
            the end
38
39          # Base origin and directions
40          self.sec_origin = numpy.array([0, 0, 0])
41          self.uDir = numpy.array([1.0, 0, 0])
42          self.wDir = numpy.array([0.0, 0, 1.0])
43          self.vDir = numpy.cross(self.wDir, self.uDir)
44
45      def place(self, sec_origin, uDir, wDir):
46          """Places the angles and gusset plates with automatic end
                positioning"""
47          self.sec_origin = sec_origin
48          self.uDir = uDir
49          self.wDir = wDir
50          self.vDir = numpy.cross(self.wDir, self.uDir)
51
52          self.sec_origin = sec_origin
53          self.uDir = uDir
54          self.wDir = wDir
55          self.vDir = numpy.cross(self.wDir, self.uDir)
56
57          # Place first angle with offset to center on gusset
58          angle1_origin = self.sec_origin
59          rotated_uDir_angle1 = -self.vDir  # Point the horizontal leg
                towards negative y
60          rotated_vDir_angle1 = -self.uDir  # Adjust vertical direction
                accordingly
61          self.angle1.place(angle1_origin, rotated_uDir_angle1, self.wDir
                )
62
63          # Place second angle with spacing
64          angle2_origin = angle1_origin + self.spacing * self.vDir
65          rotated_uDir = self.uDir
66          rotated_vDir = -self.vDir
67          self.angle2.place(angle2_origin, rotated_uDir, self.wDir)
68
69          # Calculate center point between angles
70          center_point = self.sec_origin + (self.spacing / 2) * self.vDir
71
```

```
72          # Place first gusset plate at the start
73          gusset1_origin = (
74              center_point
75              + self.gusset1_offsets[0] * self.uDir  # X offset (0)
76              + self.gusset1_offsets[1] * self.vDir  # Y offset (0)
77              + self.gusset1_offsets[2] * self.wDir  # Z offset (start
                    position)
78          )
79          gusset1_uDir = numpy.array([0, 0, 1.0])
80          gusset1_wDir = numpy.array([0, -1.0, 0])
81          self.gusset1.place(gusset1_origin, gusset1_uDir, gusset1_wDir)
82
83          # Place second gusset plate at the end
84          gusset2_origin = (
85              center_point
86              + self.gusset2_offsets[0] * self.uDir  # X offset (0)
87              + self.gusset2_offsets[1] * self.vDir  # Y offset (0)
88              + self.gusset2_offsets[2] * self.wDir  # Z offset (end
                    position)
89          )
90          gusset2_uDir = numpy.array([0, 0, -1.0])
91          gusset2_wDir = numpy.array([0, 1.0, 0])
92          self.gusset2.place(gusset2_origin, gusset2_uDir, gusset2_wDir)
93
94
95
96      def create_model(self):
97          """Creates the 3D model of back-to-back angles with gusset
                plates"""
98          # Create models
99          angle1_prism = self.angle1.create_model()
100         angle2_prism = self.angle2.create_model()
101         gusset1_prism = self.gusset1.create_model()
102         gusset2_prism = self.gusset2.create_model()
103
104         # Combine all shapes
105         combined_angles = BRepAlgoAPI_Fuse(angle1_prism, angle2_prism).
                Shape()
106         combined_with_gusset1 = BRepAlgoAPI_Fuse(combined_angles,
```

```
            gusset1_prism).Shape()
107         final_shape = BRepAlgoAPI_Fuse(combined_with_gusset1,
            gusset2_prism).Shape()
108
109         return final_shape
110 %-------------------- end code ---------------
```

Explanation of the Code

Explanation of the init Function

- Lines 18–22: Instantiates and stores geometric subcomponents (two angles and two gusset plates) using input parameters for lengths, thicknesses, radii, and gusset specifications.
- Lines 25–31: Saves key dimensions as object attributes to be used later in positioning, ensuring design parameters remain accessible throughout the assembly process.
- Lines 35–38: Calculates offsets for gusset plates based on the overall angle length and a margin, establishing where the gusset plates should be placed at the ends.
- Lines 39–43: Defines a base coordinate system by setting an origin and unit vectors, and computes a cross-product vector to support consistent spatial orientation.

Explanation of the place Function

- Lines 47–55: Updates the local coordinate system of the component by setting the origin and unit vectors for the directions, ensuring all subsequent placements follow the same orientation framework.
- Lines 57–61: Places the first angle by translating it to the base origin and applying a specific rotation so that its horizontal leg faces the designated direction.
- Lines 64–67: Positions the second angle at a calculated offset from the first angle using a defined spacing, ensuring both angles are arranged back-to-back.
- Lines 70–92: Computes a central reference point between the angles, then places the first and second gusset plates at the start and end positions respectively, using their predefined offset vectors and assigned rotation directions.

Explanation of the create_model Function

- Lines 99–102: Generates individual 3D models of the two angles and two gusset plates by invoking their respective model creation methods.
- Line 105: Combines the two angle models using a fusion (boolean union) operation, ensuring they form a single connected shape.
- Lines 106-107: Sequentially fuses the gusset plate models with the combined angle structure to integrate all the components.
- Return: Outputs the final unified 3D model, representing the complete assembly of back-to-back angles coupled with gusset plates.

4.3.2 class BackToBackAnglesWithGussetsOppSide in osdag/-cad/items/double_angles.py

The code defines a class that builds back-to-back angles with gusset plates positioned on opposite sides, ensuring perfect centering along both axes.
The constructor (init) instantiates the angle and gusset objects, stores key design dimensions, and computes offsets based on the gusset and angle parameters.
The place method positions the angles and gusset plates relative to a given coordinate system, applying offsets to center the components correctly.
The create_model method generates the individual 3D models and fuses them using boolean operations to produce the final integrated assembly.

Python Code

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

Listing 4.2: class BackToBackAnglesWithGussetsSameSide

```
1 %--------------------begin code-------------
2 class BackToBackAnglesWithGussetsOppSide:
3     def __init__(self, L, A, B, T, R1, R2, gusset_L, gusset_H, gusset_T
      , gusset_degree, spacing=2):
4         """
5         Creates back-to-back angles with gusset plates automatically
          positioned at the ends and perfectly centered
```

```
6           Args:
7               L: Length of the angle
8               A: Vertical leg length
9               B: Horizontal leg length
10              T: Thickness
11              R1, R2: Corner radii
12              gusset_L: Length of gusset plate
13              gusset_H: Height of gusset plate (width of the shorter end)
14              gusset_T: Thickness of gusset plate
15              gusset_degree: Angle of gusset plate
16              spacing: Gap between angles (including both angle
                    thicknesses)
17          """
18          self.angle1 = Angle(L, A, B, T, R1, R2)
19          self.angle2 = Angle(L, B, A, T, R1, R2)
20          self.gusset1 = GassetPlate(gusset_L, gusset_H, gusset_T,
                gusset_degree)
21          self.gusset2 = GassetPlate(gusset_L, gusset_H, gusset_T,
                gusset_degree)
22          self.spacing = spacing
23
24          # Store dimensions for later use
25          self.L = L
26          self.A = A  # Vertical leg length
27          self.B = B  # Horizontal leg length
28          self.T = T  # Angle thickness
29          self.gusset_L = gusset_L
30          self.gusset_H = gusset_H
31          self.gusset_T = gusset_T
32
33          # Calculate Z-offsets based on angle length
34          margin = gusset_L/2
35
36          # Calculate X offset to center the double angle on gusset plate
                width
37          x_offset = gusset_H/2  # Center of gusset plate width
38
39          # Calculate Y offset for gusset plates
40          # For gusset1, we need to account for its forward growth by
```

```
            adding half its thickness
41          gusset1_y_offset = spacing/2 + gusset_T/2.0
42          # For gusset2, we can use the center point since it grows
                backwards
43          gusset2_y_offset = spacing/2 - gusset_T/2.0
44
45          self.gusset1_offsets = numpy.array([x_offset, gusset1_y_offset,
                 margin])
46          self.gusset2_offsets = numpy.array([x_offset, gusset2_y_offset,
                 L - margin])
47
48          # Base origin and directions
49          self.sec_origin = numpy.array([0, 0, 0])
50          self.uDir = numpy.array([1.0, 0, 0])
51          self.wDir = numpy.array([0.0, 0, 1.0])
52          self.vDir = numpy.cross(self.wDir, self.uDir)
53
54      def place(self, sec_origin, uDir, wDir):
55          """Places the angles and gusset plates with perfect centering
                on both axes"""
56          self.sec_origin = sec_origin
57          self.uDir = uDir
58          self.wDir = wDir
59          self.vDir = numpy.cross(self.wDir, self.uDir)
60
61          # Calculate the offset needed to center angles on gusset plate
62          x_center_offset = (self.gusset_H - self.A)/2
63
64          # Place first angle with offset to center on gusset
65          angle1_origin = self.sec_origin + x_center_offset * self.uDir
66          rotated_uDir_angle1 = -self.vDir  # Point the horizontal leg
                towards negative y
67          rotated_vDir_angle1 = -self.uDir  # Adjust vertical direction
                accordingly
68          self.angle1.place(angle1_origin, rotated_uDir_angle1, self.wDir
                )
69
70          # Place second angle with spacing
71          angle2_origin = angle1_origin + self.spacing * self.vDir
```

```
72          rotated_uDir = self.uDir
73          rotated_vDir = -self.vDir
74          self.angle2.place(angle2_origin, rotated_uDir, self.wDir)
75
76          # Place first gusset plate
77          gusset1_origin = (
78              self.sec_origin
79              + self.gusset1_offsets[0] * self.uDir  # X offset (centered
                    )
80              + self.gusset1_offsets[1] * self.vDir  # Y offset (adjusted
                     for forward growth)
81              + self.gusset1_offsets[2] * self.wDir  # Z offset (start
                    position)
82          )
83          gusset1_uDir = numpy.array([0, 0, 1.0])
84          gusset1_wDir = numpy.array([1.0, 0, 0])
85          self.gusset1.place(gusset1_origin, gusset1_uDir, gusset1_wDir)
86
87          # Place second gusset plate
88          gusset2_origin = (
89              self.sec_origin
90              + self.gusset2_offsets[0] * self.uDir  # X offset (centered
                    )
91              + self.gusset2_offsets[1] * self.vDir  # Y offset (at
                    center point)
92              + self.gusset2_offsets[2] * self.wDir  # Z offset (end
                    position)
93          )
94          gusset2_uDir = numpy.array([0, 0, -1.0])
95          gusset2_wDir = numpy.array([1.0, 0, 0])
96          self.gusset2.place(gusset2_origin, gusset2_uDir, gusset2_wDir)
97
98      def create_model(self):
99          """Creates the 3D model of back-to-back angles with gusset
                plates"""
100         # Create models
101         angle1_prism = self.angle1.create_model()
102         angle2_prism = self.angle2.create_model()
103         gusset1_prism = self.gusset1.create_model()
```

```
104         gusset2_prism = self.gusset2.create_model()
105
106         # Combine all shapes
107         angle1_with_gusset = BRepAlgoAPI_Fuse(angle1_prism,
                gusset1_prism).Shape()
108         angle2_with_gusset = BRepAlgoAPI_Fuse(angle2_prism,
                gusset2_prism).Shape()
109         final_shape = BRepAlgoAPI_Fuse(angle1_with_gusset,
                angle2_with_gusset).Shape()
110
111         return final_shape
112 %-------------------- end code ---------------
```

Explanation of the Code

Explanation of the init Function

- Lines 18–22: Instantiates two angle objects and two gusset plate objects with provided dimensions and design parameters.
- Lines 25–31: Stores key geometric dimensions (such as lengths, leg dimensions, and thicknesses) for later reference in positioning.
- Lines 33–46: Calculates offsets, including a margin based on gusset plate length and offsets in the X and Y directions, to ensure gussets are centered relative to the double angle configuration.
- Lines 49–52: Establishes the base coordinate system by defining an origin and unit vectors for the directions, and computes the cross-product vector for consistent spatial orientation.

Explanation of the place Function

- Lines 56–59: Updates the component's local coordinate system by setting a new origin and orientation vectors to guide the placement.
- Lines 61–68: Determines an additional X offset to center the angles relative to the gusset plate width, and directly applies this offset to the first angle's placement.

- Lines 71–74: Positions the second angle at a defined spacing from the first angle by translating along the computed cross-product vector.
- Lines 77–96: Places the gusset plates at the start and end positions using the precomputed gusset offsets, while assigning proper orientation vectors to ensure correct alignment.

Explanation of the create_model Function

- Lines 101–104: Generates the individual 3D models from the two angles and the two gusset plates by invoking their respective model creation methods.
- Lines 107–108: Fuses the model of the first angle with its corresponding gusset plate and similarly for the second angle, ensuring that each pairing forms a unified subassembly.
- Line 109: Combines the two subassemblies into a final unified shape using a further fusion operation.
- Return: Outputs the final 3D integrated model, representing the complete assembly of back-to-back angles with oppositely centered gusset plates.

4.3.3 createStrutsInTrusses() in osdag/cad/common_logic.py

The function "createStrutsInTrusses" determines the type of strut profile to construct based on the provided section profile stored in Col.sec_profile.
For a simple "Angles" profile, it computes key dimensions for a single angle and creates a 3D model using the Angle class.
It prints numerical values such as length, leg dimensions, thickness, and radii. For profiles labeled "Back to Back Angles – Same side of gusset," it computes design parameters and uses the BackToBackAnglesWithGussetsSameSide class.
The function sets a fixed spacing value and example gusset plate dimensions, prints gusset thickness, and selects leg assignments based on location.
It then positions the assembly by setting origin and direction vectors. For profiles labeled "Back to Back Angles – Opposite side of gusset," it similarly computes dimensions and uses the BackToBackAnglesWithGussetsOppSide class.

The spacing for this case is set to the plate thickness, and dimensions are selected based on the "Long Leg" or "Short Leg" designation.

After creating the appropriate assembly using one of the previous two classes, the function places the assembly in 3D space.

Finally, it calls create_model on the assembly to generate and return the final fused 3D shape.

Python Code

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

Listing 4.3: createStrutsInTrusses()

```
%--------------------begin code-------------
    def createStrutsInTrusses(self):
        Col = self.module_class
        Col.section_property = AngleComponent(designation = Col.
           result_designation, material_grade = Col.material)
        if Col.sec_profile=="Angles":

            L = float(Col.length)
            A = float(Col.section_property.max_leg)
            B = float(Col.section_property.min_leg)
            T = float(Col.section_property.thickness)
            R1 = float(Col.section_property.root_radius)
            R2 = float(Col.section_property.toe_radius)
            print("Length (L):", L)
            print("Max Leg (A):", A)
            print("Min Leg (B):", B)
            print("Thickness (T):", T)
            print("Root Radius (R1):", R1)
            print("Toe Radius (R2):", R2)

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

            angle = Angle(L, A, B, T, R1, R2)
            _place = angle.place(origin, uDir, wDir)
```

```
            point = angle.computeParams()
            prism = angle.create_model()

            return prism
        elif Col.sec_profile=="Back to Back Angles - Same side of
            gusset":

            L = float(Col.length)
            T = float(Col.section_property.thickness)
            R1 = float(Col.section_property.root_radius)
            R2 = float(Col.section_property.toe_radius)
            spacing = 6.0  # Gap between angles
            print("Length (L):", L)
            print("Thickness (T):", T)
            print("Root Radius (R1):", R1)
            print("Toe Radius (R2):", R2)

            # Example dimensions for gusset plates
            gusset_L = 100  # Length
            gusset_H = 100  # Height
            gusset_T = float(Col.plate_thickness)    # Thickness
            print("Gusset Thickness : ", Col.plate_thickness)
            gusset_degree = 30  # Angle in degrees

            # Create and display the assembly
            origin = numpy.array([0., 0., 0.])
            uDir = numpy.array([1., 0., 0.])
            wDir = numpy.array([0., 0., 1.])
            if Col.loc == "Long Leg":
                B = float(Col.section_property.max_leg)
                A = float(Col.section_property.min_leg)
            elif Col.loc == "Short Leg":
                A = float(Col.section_property.max_leg)
                B = float(Col.section_property.min_leg)
            print("Vertical Leg :", A)
            print("Horizontal Leg :", B)
            assembly = BackToBackAnglesWithGussetsSameSide(L, A, B, T,
                R1, R2, gusset_L, gusset_H, gusset_T, gusset_degree,
                spacing)
```

```
62              assembly.place(origin, uDir, wDir)
63              shape = assembly.create_model()
64
65              return shape
66
67          elif Col.sec_profile=="Back to Back Angles - Opposite side of
                gusset":
68
69              L = float(Col.length)
70              T = float(Col.section_property.thickness)
71              R1 = float(Col.section_property.root_radius)
72              R2 = float(Col.section_property.toe_radius)
73              spacing = float(Col.plate_thickness)   # Gap between angles
74              print("Length (L):", L)
75              print("Thickness (T):", T)
76              print("Root Radius (R1):", R1)
77              print("Toe Radius (R2):", R2)
78
79
80              # Example dimensions for gusset plates
81              gusset_L = 100  # Length
82              gusset_H = 100 # Height
83              gusset_T =  float(Col.plate_thickness)   # Thickness
84              print("Gusset Thickness : ", Col.plate_thickness)
85              gusset_degree = 30  # Angle in degrees
86
87              # Create and display the assembly
88              origin = numpy.array([0., 0., 0.])
89              uDir = numpy.array([1., 0., 0.])
90              wDir = numpy.array([0., 0., 1.])
91              if Col.loc == "Long Leg":
92                  A = float(Col.section_property.max_leg)
93                  B = float(Col.section_property.min_leg)
94              elif Col.loc == "Short Leg":
95                  B = float(Col.section_property.max_leg)
96                  A = float(Col.section_property.min_leg)
97              print("Vertical Leg :", A)
98              print("Horizontal Leg :", B)
99              assembly = BackToBackAnglesWithGussetsOppSide(L, A, B, T,
```

```
                R1, R2, gusset_L, gusset_H, gusset_T, gusset_degree,
                spacing)
100             assembly.place(origin, uDir, wDir)
101             shape = assembly.create_model()
102
103             return shape
104 %-------------------- end code ---------------
```

Explanation of the Code

- Lines 1–5: The function begins by retrieving the module's column properties and setting the section properties using an AngleComponent object if the strut is a simple angle. It prints out the primary dimensions (such as length, leg sizes, thickness, and radii) for debugging and verification.

- Branch for "Angles": For a basic angle profile, the code instantiates an Angle object with the computed dimensions. It then calls methods to position the angle, compute any additional parameters, and generate its 3D model, returning the resulting prism.

- Branch for "Back to Back Angles – Same side of gusset": If the section profile matches this type, the function establishes specific gusset plate dimensions and spacing. It prints relevant parameters, adjusts the vertical and horizontal leg dimensions based on location, and then utilizes the previously defined BackToBackAnglesWithGussetsSameSide class. The assembly is positioned using a common origin and unit vectors before its 3D model is created and returned.

- Branch for "Back to Back Angles – Opposite side of gusset": In this case, a slightly different spacing is used (equal to the plate thickness), and the leg dimensions are adjusted accordingly. The assembly is constructed with the BackToBackAnglesWithGussetsOppSide class. After setting its placement with the specified origin and directions, the final shape is generated by fusing the constituent components and then returned.

Chapter 5

Internship Task 3: Creating CADs for Flexural Members

5.1 Task 3: Problem Statement

Display columns oriented horizontally in the CAD window. For now, both the Simply Supported Beam and Cantilever Beam submodules display the same CAD.

5.2 Task 3: Tasks Done

The column geometry should be changed to place the columns horizontally. Separate functions should be made for both the submodules. The functions should use the class given in ISection.py.

5.2.1 Methodology Behind the Creation of the CAD Model

1. **Geometric Definition:** We start by defining the cross-sectional geometry of the component using a set of points. These points are used to create edges, which are then combined into a closed wire and converted into a face representing the cross-section.

2. **Extrusion for 3D Solid:** Extrude the cross-sectional face along a specified direction and length to create a 3D solid model of the component. This step adds volume to the 2D geometry, forming the primary structure.

3. **Feature Integration:** If additional features (e.g., notches) are required, calculate their precise placement relative to the main geometry using vector transformations. Create the feature as a separate model and subtract it from the primary solid using Boolean operations, modifying the geometry as needed.

4. **Coordinate System and Orientation Setup:** Establish a local coordinate system with defined origin and directional vectors (e.g., x-axis for horizontal alignment). This ensures that all subsequent operations are performed in a consistent spatial framework.

5. **Placement and Alignment:** Position the component in 3D space by aligning it with the desired orientation (e.g., horizontal along the x-axis). Use calculated vectors to ensure proper alignment relative to other structural elements.

6. **Model Assembly:** Combine all components and features into a unified CAD model. Ensure that all transformations, placements, and orientations are applied consistently to create an accurate representation of the final structure.

7. **Final Output:** Generate and return the completed 3D model.

5.3 Task 3: Python Codes

5.3.1 create_model() in osdag/cad/items/ISection.py

This function takes a set of defined points and transforms them into a 3D solid model. It starts by generating edges, assembling them into a wire, and then constructing a face from that wire. The face is extruded along a specified direction to create a prism. If a notch feature is provided, it positions and integrates the notch by subtracting its geometry from the solid using a boolean cut.

Python Code

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

Listing 5.1: create_model()

```
1 %--------------------begin code-------------
2     def create_model(self):
```

```
3
4           edges = makeEdgesFromPoints(self.points)
5           wire = makeWireFromEdges(edges)
6           aFace = makeFaceFromWire(wire)
7           extrudeDir = self.length * self.wDir  # extrudeDir is a numpy
                array
8           prism = makePrismFromFace(aFace, extrudeDir)
9
10          if self.notchObj is not None:
11              uDir = numpy.array([-1.0, 0.0, 0])
12              wDir = numpy.array([0.0, 1.0, 0.0])
13              shiftOri = self.D / 2.0 * self.vDir + self.notchObj.width /
                    2.0 * self.wDir + self.B / 2.0 * -self.uDir  # + self.
                    notchObj.width* self.wDir + self.T/2.0 * -self.uDir
14              origin2 = self.sec_origin + shiftOri
15
16              self.notchObj.place(origin2, uDir, wDir)
17
18              notchModel = self.notchObj.create_model()
19              prism = BRepAlgoAPI_Cut(prism, notchModel).Shape()
20
21          return prism
22  %-------------------- end code ---------------
```

Explanation of the Code

- Lines 2-6: Transforms a set of points into basic geometric elements by creating edges, linking them into a continuous wire, and forming a face that represents the cross-sectional profile.
- Lines 7-8: Extrudes the face along a specified direction (scaled by the structural member's length) to form a solid prism.
- Lines 10-18: If a notch feature is defined, computes its precise position using vector operations, positions the notch accordingly, creates the notch model, and removes the corresponding material from the prism through a boolean cut.
- Return: Provides the final 3D model with integrated features (if any notch is

present).

5.3.2 createcolumnFlexGeometry() in osdag/cad/CompressionMembers/column.py

The function ensures that the column is reoriented so that its main axis aligns horizontally along the x-axis.

It establishes a new local coordinate system by setting the origin and defining unit vectors along the x-axis (u-direction) and y-axis (w-direction).

The cross-sectional geometry is then placed into this coordinate system, and a 3D model of the column is generated.

The resulting model is stored for further processing or visualization.

Python Code

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

Listing 5.2: createcolumnFlexGeometry()

```
%--------------------begin code-------------
    def createcolumnFlexGeometry(self):
        """
        Ensures the column is always horizontal along the x-axis.

        :return: Geometric orientation of this component
        """
        # The origin of the column
        columnOriginL = numpy.array([0.0, 0.0, 0.0])

        # Set the column's local u-direction and w-direction

        uDir = numpy.array([1.0, 0.0, 0.0])  # Along x-axis
        wDir = numpy.array([0.0, 1.0, 0.0])  # Along y-axis (horizontal
            )

        # Place the section at the specified origin with the new
            orientation
        self.sec.place(columnOriginL, uDir, wDir)

```

```
19          # Create the column model based on the section and orientation
20          self.columnModel = self.sec.create_model()
21  %-------------------- end code ---------------
```

Explanation of the Code

- Lines 8–14: Defines the base origin and establishes a new local coordinate system by assigning unit vectors for the horizontal (x-axis) and vertical (y-axis) directions.
- Lines 16–17: Positions the existing cross-sectional geometry within this coordinate system using the specified origin and directional vectors, ensuring consistent orientation.
- Final Step: Stores the created model.

5.3.3 createSimplySupportedBeam() in osdag/cad/common_logic.py

The code constructs a simply supported beam model by retrieving the optimum section properties.
It converts these section parameters into a profile using an ISection object and sets the beam's effective length.
The beam is then positioned in a defined coordinate system, and a compression member CAD model is created using these inputs.
Finally, it generates a 3D model for the simply supported beam for further analysis or visualization. **The same code is used in createCantileverBeam() function.**

Python Code

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

Listing 5.3: createSimplySupportedBeam()

```
1  %--------------------begin code-------------
2      def createSimplySupportedBeam(self):
3
4          Flex = self.module_class
5
6          print(f"Flex.support {Flex.support}")
```

```
7
8           Flex.section_property = Flex.section_connect_database(Flex,
                Flex.result_designation)
9           column_tw = float(Flex.section_property.web_thickness)
10          print(f"Flex.section_property.web_thickness : {Flex.
                section_property.web_thickness}")
11          column_T = float(Flex.section_property.flange_thickness)
12          print(f"Flex.section_property.flange_thickness : {Flex.
                section_property.flange_thickness}")
13          column_d = float(Flex.section_property.depth)
14          print(f"Flex.section_property.depth : {Flex.section_property.
                depth}")
15          column_B = float(Flex.section_property.flange_width)
16          print(f"Flex.section_property.flange_width : {Flex.
                section_property.flange_width}")
17          column_R1 = float(Flex.section_property.root_radius)
18          print(f"Flex.section_property.root_radius : {Flex.
                section_property.root_radius}")
19          column_R2 = float(Flex.section_property.toe_radius)
20          print(f"Flex.section_property.toe_radius : {Flex.
                section_property.toe_radius}")
21          column_alpha = 94  # Todo: connect this. Waiting for danish to
                give variable
22          column_length = float(Flex.result_eff_len)*1000
23
24          sec = ISection(B=column_B, T=column_T, D=column_d, t=column_tw,
                 R1=column_R1, R2=column_R2,
25                                 alpha=column_alpha, length=column_length,
                                     notchObj=None)
26          _place=sec.place(numpy.array([0.,0.,0.]),numpy.array
                ([1.,0.,0.]),numpy.array([0.,1.,0.]))
27          col = CompressionMemberCAD(sec)
28
29          sec=sec.create_model()
30          col.create_Flex3DModel()
31
32          return sec
33  %-------------------- end code ---------------
```

5.3.4 Explanation of the Code

- Lines 4-8: The function starts by accessing the support type and retrieving section properties — such as web thickness, flange thickness, depth, flange width, and the root and toe radii.

- Lines 9-20: These parameters are printed for debugging and converted to appropriate numerical types, and the effective beam length is computed and scaled.

- Line 24: An ISection object is instantiated using the extracted parameters, creating the cross-sectional profile for the simply supported beam.

- Line 26: The cross-sectional geometry is positioned in a predefined coordinate system with set origin and unit vectors.

- Line 27: A CompressionMemberCAD object is created based on the ISection, ensuring that the model adheres to the specified placement.

- Line 29: The 3D model of the simply supported beam is generated by invoking the creation routines on the ISection and the CompressionMemberCAD objects.

- Line 30: Finally, the complete model is returned, making it ready for further processing, analysis, or visualization.

Chapter 6

CAD Manual

A manual to get started on working on CADs in Osdag has been made. It contains the flow of logic in frontend and backend, accompanied by flowcharts and diagrams. New interns may use the manual to understand the structure of the code and get started on modifying the existing CADs and creating new ones. It is beginner-friendly and only requires rudimentary knowledge of python and basic steel structures.

Chapter 7

Conclusions

7.1 Tasks Accomplished

7.1.1 Task 1: CAD for Columns with known support conditions

The geometry is constructed by defining points, creating edges, and forming a cross-sectional face. This face is extruded along a specific direction to create a 3D solid. Optional features like notches are integrated using boolean operations. A local coordinate system ensures consistent orientation, and the cross-section is placed at a defined origin with proper alignment for accurate 3D positioning.

7.1.2 Task 2: Creating CADs for Struts in Trusses

Individual components are created, including two angles and two gusset plates, each defined with specific design parameters. A modular approach ensures these elements are instantiated separately before integration. Strategic placement is achieved by calculating orientation vectors and offsets, determining a central reference point, and arranging the angles back-to-back with defined spacing while positioning gussets precisely at the ends. Consistent coordinate systems and orientations are maintained to ensure proper alignment of all parts within the assembly. Finally, Boolean fusion operations are employed to combine the individual models into a unified structure, consolidating all components into a single, cohesive 3D model.

7.1.3 Task 3: Creating CADs for Flexural Members

Again, the process begins with defining the cross-sectional geometry using points, which are converted into edges, wires, and a face. This face is extruded along a specified direction to form a 3D solid, creating the primary structure. Additional features like notches are integrated by calculating their positions and using boolean operations to modify the geometry. A local coordinate system is established with defined origin and directional vectors to ensure consistent alignment. The component is positioned in 3D space with precise orientation relative to other elements. All components and features are then combined into a unified CAD model, which is finalized and returned as the complete 3D representation.

7.2 Skills Developed

Technical Skills Gained

1. **Parametric CAD Modeling**
    - Learned to create parametric 3D models using PythonOCC, including defining points, edges, wires, faces, and extruding them to construct structural components.
2. **Boolean Operations**
    - Gained expertise in integrating features like notches into models using precise placements and Boolean operations (e.g., fusion and subtraction).
3. **Coordinate Systems and Spatial Orientation**
    - Developed skills in defining local coordinate systems with origin points and directional vectors for accurate placement and alignment of components in 3D space.
4. **Modular Design and Assembly**
    - Improved the ability to design modular components (e.g., angles, gusset plates) and assemble them into complex structures using calculated offsets and orientations.

5. **Integration of Structural Profiles**

    - Enhanced knowledge of working with structural profiles like I-sections and back-to-back angles, including extracting parameters from databases for accurate modeling.

6. **PythonOCC Workflow Optimization**

    - Learned to streamline CAD workflows in PythonOCC by breaking down modeling tasks into logical steps such as geometry creation, extrusion, feature integration, placement, and assembly.

Professional Skills Gained

1. **Team Collaboration**

    - Improved teamwork by collaborating effectively with individuals from different disciplines, ensuring smooth integration of ideas and workflows.

2. **Effective Communication**

    - Practiced clear and concise communication of technical concepts to team members through structured explanations and discussions.

3. **Report Writing and Documentation**

    - Enhanced skills in documenting methodologies, creating detailed reports, and presenting processes in a logical format for better understanding by diverse audiences.

4. **Cross-Disciplinary Coordination**

    - Learned to work with professionals from various disciplines, aligning technical details with broader project requirements to achieve cohesive results.

These technical and professional skills collectively strengthen capabilities in computational geometry, CAD modeling, teamwork, and effective communication within engineering projects.

Chapter A

Appendix

A.1 Work Reports

Name:	Aryan Gupta		
Project:	Osdag		
Date	**Day**	**Task**	**Hours worked**
12-Nov-24	Tuesday	Joining|Installed Osdag| Went through Osdag spoken tutorials	2
13-Nov-24	Wednesday	Studied basics of compression and flexural members | Menu driven program which asks the user to selectbbetween a cylindrical and cuboidal compression member and enter fillet radius for filleted edges	3
14-Nov-24	Thursday	Created custom installer for using Compression and Flexural member in Osdag	4
15-Nov-24	Friday	Developed code to create and visualize an i-section beam using pythonocc, taking user parameters	4
16-Nov-24	Saturday	Converting codebase to use pythonocc version 0.18 from version 0.8 to run osdag home page	4
17 to 30-Nov-24		END SEMESTER EXAMINATIONS AND TRAVEL	
1-Dec-24	Sunday	Installing and configuring Osdag using conda	2
2-Dec-24	Monday	Checking 28 osi files using Osdag to see accuracy	3
3-Dec-24	Tuesday	Isolated files required for creation of fin plate. Edited so that they can work independently from the Osdag main page without using conda package. Created a tracer wrapper script to track which function is being called from the osda	4
4-Dec-24	Wednesday	Understanding nut.py,bolt.py,ModelUtils.py and ISection.py and their call order to see creation of fin plate. Replicated trace wrapper for Flexural member.	4
5-Dec-24	Thursday	Understanding _init_.py under utilities, notch.py, beamWebBeamWebConnectivity.py, colWebBeamWebConnectivity.py	3
6-Dec-24	Friday	Understand flexure.py and common_logic.py	4
7-Dec-24	Saturday	Created trace_wrapper.py to see the flow of code in creating the CAD.	4
8-Dec-24	Sunday	Understood the flow of logic in creating CADs for Columns with known support conditions. Rectified error in object type of Topo_DS_Shape .	3
9-Dec-24	Monday	Displayed RHS/SHS and CHS column CADs with user input.	4
10-Dec-24	Tuesday	Understood theory behind back to back angles	4
11-Dec-24	Wednesday	Dispalyed individual angle for Angle section profile.	4
12-Dec-24	Thursday	Understood theory behind Gusset plates.	4
13-Dec-24	Friday	Created separate class for Back to back angles on same side of gusset plate.	3
14-Dec-24	Saturday	Modified CAD for double angles to include optimim space in between.	4
15-Dec-24	Sunday	BREAK	
16-Dec-24	Monday	Created separate class for Back to back angles on opposite side of gusset plate.	3
17-Dec-24	Tuesday	Meeting with mentors for problems with CAD display	4
18-Dec-24	Wednesday	Fixed discussed issues with Back to Back angles	4
19-Dec-24	Thursday	Fixed dimensions of gusset plates after feedback from mentors	4
20-Dec-24	Friday	Looked into the sqlite database for correct values of CADs with different result designations	3
21-Dec-24	Saturday	Corrected dimensions being taken from user for back to back angles	4
22-Dec-24	Sunday	Meeting to discuss problems with report generation	4
23-Dec-24	Monday	Fixed dimensions being different in CAD and result designation	3
24-Dec-24	Tuesday	Created common CAD for Cantlever Beam and Simply Supported Beam	3
25-Dec-24	Wednesday	Fixed column orientation in Flexural Members	3
26-Dec-24	Thursday	Created separate functions for cantilever beam and simply supported beam	3
27-Dec-24	Friday	Explaored different options for adding black edges to the final CAD to prevent slowdown	4
28-Dec-24	Saturday	Started with documentation for CAD manual	4
29-Dec-24	Sunday	Added workflows for all CADs using trace_wrapper.py	4
30-Dec-24	Monday	Started documentation for final internship report	3
31-Dec-24	Tuesday	Attended meeting with mentors for report content and format	
1-Jan-25	Wednesday	BREAK	

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.