



FOSSEE Winter Internship Report

On

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

Submitted by

Sachin Saud

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

Institute of Engineering, Tribhuvan University

Kathmandu, Nepal

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Ajmal Babu M S

Parth Karia

Ajinkya Dahale

July 14, 2025

Acknowledgments

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

Contents

1	Introduction	5
1.1	National Mission in Education through ICT	5
1.1.1	ICT Initiatives of MoE	6
1.2	FOSSEE Project	7
1.2.1	Projects and Activities	7
1.2.2	Fellowships	7
1.3	Osdag Software	8
1.3.1	Osdag GUI	9
1.3.2	Features	9
2	Screening Task	10
2.1	Problem Statement	10
2.2	Screening Tasks Done	10
2.3	Screening task: Python Code	11
2.3.1	Description of the Script	11
2.3.2	Python Code	12
3	Bolted Butt Joint	23
3.1	Problem Statement	23
3.2	Task Done	23
3.3	Python Code	23
3.3.1	Description of the Script	23
3.3.2	Python Script	24
3.3.3	Explanation of the Code	29
3.4	Documentation	30
3.4.1	Directory Structure	30
4	Welded Butt Joint	31
4.1	Problem Statement	31
4.2	Task Done	31
4.3	Python Code	32

4.3.1	Description of the Script	32
4.3.2	Python Script	33
4.3.3	Explanation of the Code	36
5	Double Angle Gusset Joint	38
5.1	Problem Statement	38
5.2	Task Done	38
5.3	Python Code(Bolted gusset joint)	39
5.3.1	Description of the Script	39
5.3.2	Python Script	39
5.3.3	Explanation of the Code	45
5.4	Python Code(Welded gusset joint)	46
5.4.1	Description of the Script	46
5.4.2	Python Script	48
5.4.3	Explanation of the Code	53
6	CAD Integration	55
6.1	Problem Statement	55
6.2	Task Done	55
6.3	Python Code inside common_logic.py	55
6.3.1	Python Script	55
6.3.2	Explanation of the Code	57
6.3.3	Full code	60
7	Gantry Girder	78
7.1	Problem Statement	78
7.2	Task Done	78
7.3	Python Code	78
7.3.1	Description of the Script	78
7.3.2	Python Script	79
7.3.3	Explanation of the Code	82
8	Castellated Beam	84
8.1	Problem Statement	84

8.2	Task Done	84
8.3	Python Code	84
8.3.1	Description of the Script	84
8.3.2	Python Script	86
8.3.3	Explanation of the Code	90
9	Conclusions	92
9.1	Tasks Accomplished	92
9.2	Skills Developed	93
A	Appendix	94
A.1	Work Reports	94
	Bibliography	101

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	SWAYAMPRAHBA	Access 24x7 TV programs	Enable SWAYAMPRAHBA viewing facilities
Digital Content Access			
3	National Digital Library	Access e-content in multiple disciplines	List e-content; form NDL Clubs
4	e-PG Pathshala	Access free books and e-content	Host e-books
5	Shodhganga	Access Indian research theses	List institutional theses
6	e-ShodhSindhu	Access full-text e-resources	Access e-resources for institutions
Hands-on Learning			
7	e-Yantra	Hands-on embedded systems training	Create e-Yantra labs with IIT Bombay
8	FOSSEE	Volunteer for open-source software	Run labs with open-source software
9	Spoken Tutorial	Learn IT skills via tutorials	Provide self-learning IT content
10	Virtual Labs	Perform online experiments	Develop curriculum-based experiments
E-Governance			
11	SAMARTH ERP	Manage student lifecycle digitally	Enable institutional e-governance
Tracking and Research Tools			
12	VIDWAN	Register and access experts	Monitor faculty research outcomes
13	Shodh Shuddhi	Ensure plagiarism-free work	Improve research quality and reputation
14	Academic Bank of Credits	Store and transfer credits	Facilitate credit redemption

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

1.2 FOSSEE Project

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

1.2.1 Projects and Activities

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

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

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

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

For more details, visit the official FOSSEE website.

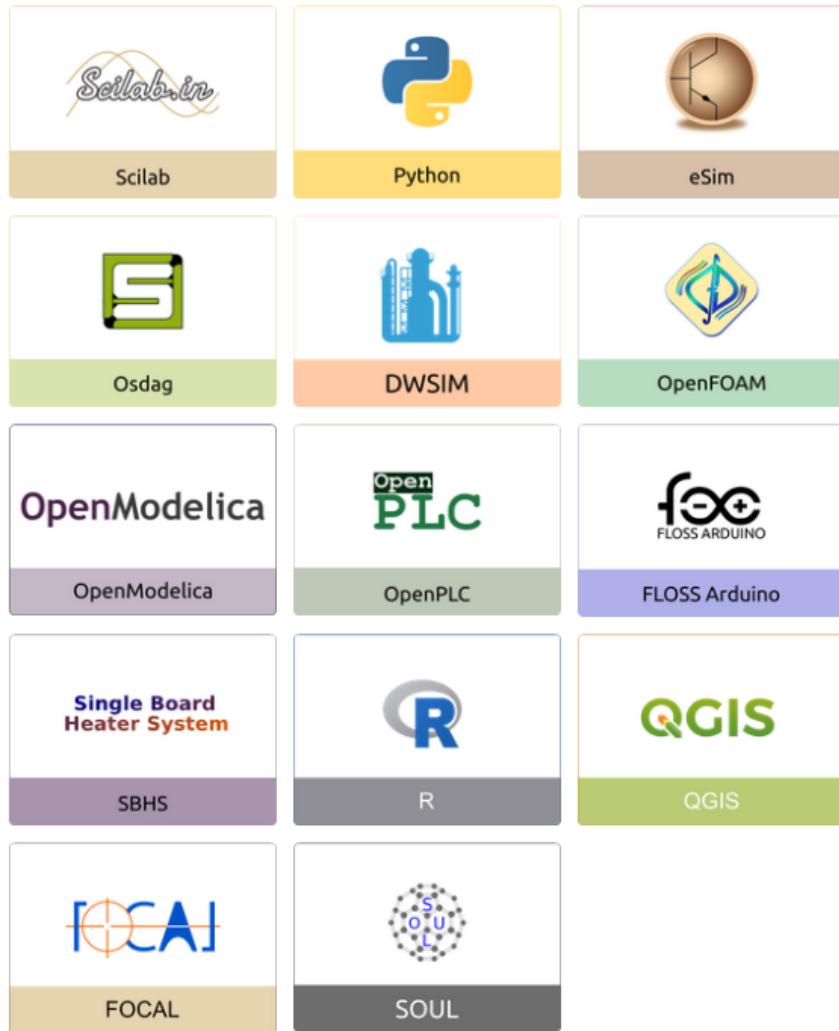


Figure 1.1: FOSSEE Projects and Activities

1.3 Osdag Software

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

1.3.1 Osdag GUI

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

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

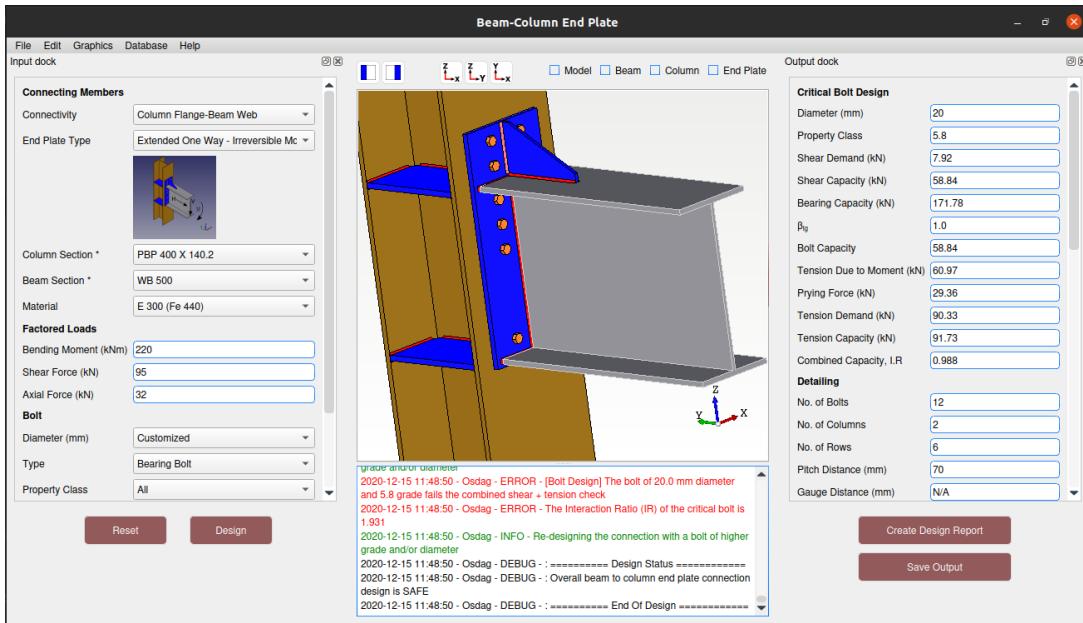


Figure 1.2: Osdag GUI

1.3.2 Features

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

For more details, visit the official Osdag website.

Chapter 2

Screening Task

2.1 Problem Statement

3D Modeling of a Portal Frame Structure.

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

2.2 Screening Tasks Done

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

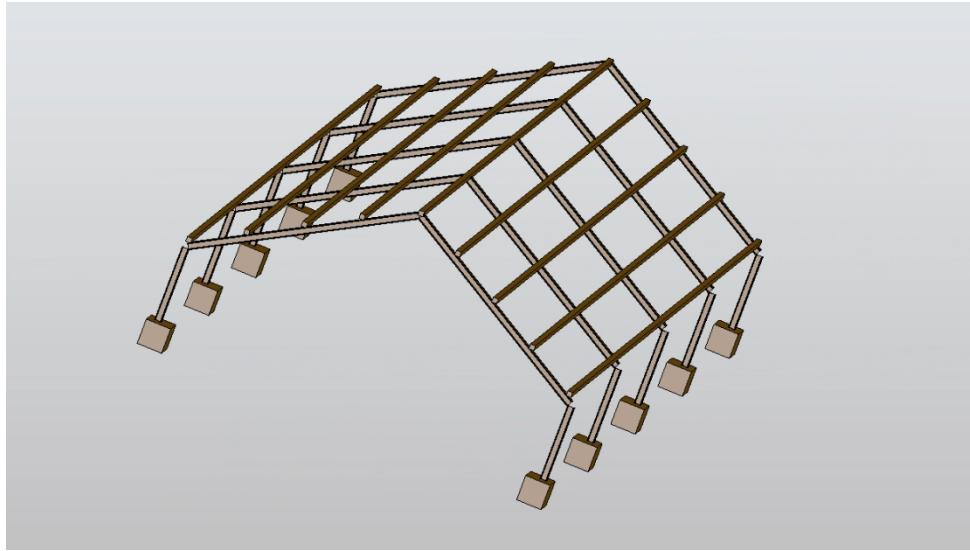


Figure 2.1: Final CAD of portal frame structure

2.3 Screening task: Python Code

2.3.1 Description of the Script

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

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

- Design Calculation:

- `draw_i_section.py`: Creates I-section profiles for columns and rafters using `BRepPrimAPI_MakeBox` and `BRepAlgoAPI_Fuse`.
 - `draw_rectangular_prism.py`: Generates roof prisms using `BRepPrimAPI_MakeBox`.
 - `portal_frame.py`: Assembles the structure by:
 - * Creating five pairs of vertical I-section columns, spaced by `rafter_length`.
 - * Adding inclined I-section rafters from column tops to a calculated apex.
 - * Placing roof prisms along rafters, spaced by $rafter_length \times \cos(\text{angle}) / (\text{num_prisms} - 1)$.
 - * Adding base boxes ($1000 \times 1000 \times 1000$ mm) at column bases.

- **Output:** The scripts produce a complete CAD model of the portal frame structure as a `TopoDS_Shape`, visualized using the OCC display module (`init_display`).

2.3.2 Python Code

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

Listing 2.1: Draw_i_section.py

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

```

17     - web_thickness: Thickness of the web
18
19     Returns:
20     - i_section_solid: The I-section CAD model as a TopoDS_Solid
21     """
22
23     # Dimensions for the I-section
24
25     # Create the bottom flange
26     bottom_flange = BRepPrimAPI_MakeBox(length, width, flange_thickness)
27             ).Shape()
28
29     # Create the top flange
30     top_flange = BRepPrimAPI_MakeBox(length, width, flange_thickness).
31             Shape()
32
33     trsf = gp_Trsf()
34     trsf.SetTranslation(gp_Vec(0, 0, depth - flange_thickness)) # Move
35             to the top
36     top_flange_transform = BRepBuilderAPI_Transform(top_flange, trsf,
37             True).Shape()
38
39
40     # Create the web
41     web = BRepPrimAPI_MakeBox(length, web_thickness, web_height).Shape
42             ()
43
44     trsf = gp_Trsf()
45     trsf.SetTranslation(gp_Vec(0, (width - web_thickness) / 2,
46             flange_thickness)) # Centered between flanges
47     web_transform = BRepBuilderAPI_Transform(web, trsf, True).Shape()
48
49
50     # Combine the flanges and web to form the I-section
51     i_section_solid = BRepAlgoAPI_Fuse(bottom_flange,
52             top_flange_transform).Shape()
53     i_section_solid = BRepAlgoAPI_Fuse(i_section_solid, web_transform).
54             Shape()
55
56
57     return i_section_solid
58
59
60
61 if __name__ == "__main__":

```

```

48     length = 1000.0
49     width = 100.0 # Width of the I-section
50     height = 200.0 # Height of the I-section
51     flange_thickness = 10.0 # Thickness of the flanges
52     web_thickness = 5.0 # Thickness of the web
53
54     i_section = create_i_section(length, width, height,
55                                   flange_thickness, web_thickness)
56
57     # Visualization
58     display, start_display, add_menu, add_function_to_menu =
59     init_display()
60
61     # Show the I-section model
62     display.DisplayShape(i_section, update=True)
63     display.FitAll()
64     start_display()

```

Listing 2.2: portal_frame.py

```

1 import sys
2 from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox
3 from OCC.Display.SimpleGui import init_display
4 from OCC.Core.gp import gp_Pnt
5
6
7 def create_rectangular_prism(length, breadth, height):
8     # Create a rectangular prism using BRepPrimAPI_MakeBox
9     box = BRepPrimAPI_MakeBox(length, breadth, height).Shape()
10    return box
11
12
13 def display_prism(box):
14     # Initialize the display window
15     display, start_display, add_menu, add_function_to_menu =
16     init_display()
17
18     # Display the box
19     display.DisplayShape(box, update=True)

```

```

20     # Start the display loop
21     start_display()
22
23
24 if __name__ == "__main__":
25     # Dimensions of the rectangular prism
26     length = 40.0
27     breadth = 20.0
28     height = 100.0
29
30     # Create the rectangular prism
31     box = create_rectangular_prism(length, breadth, height)
32
33     # Display the rectangular prism
34     display_prism(box)

```

Listing 2.3: draw_rectangular_prism

```

1 from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
2 from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_Transform
3 from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox
4 from OCC.Core.gp import gp_Vec, gp_Trsf, gp_Ax1, gp_Dir, gp_Pnt
5 from OCC.Display.SimpleGui import init_display
6 from draw_i_section import create_i_section
7 from draw_rectangular_prism import create_rectangular_prism
8 import math
9
10 def create_custom_structure(
11     column_height, column_thickness, rafter_length, rafter_angle,
12     roof_length, roof_width, roof_height, flange_thickness,
13     web_thickness,
14     prism_spacing
15 ):
16     """
17         Create a CAD model of the described structure with prisms placed
18             uniformly along the rafters to form the roof.
19
20     Parameters:
21     - column_height: Height of the columns
22     - column_thickness: Thickness of the I-section columns

```

```

21     - rafter_length: Length of the rafters
22     - rafter_angle: Angle of inclination of the rafters (in degrees)
23     - roof_length: Length of the rectangular roof prism
24     - roof_width: Width of the rectangular roof prism
25     - roof_height: Height of the rectangular roof prism
26     - flange_thickness: Thickness of the I-section flanges
27     - web_thickness: Thickness of the I-section web
28     - prism_spacing: Horizontal distance between each roof prism along
29         the rafter
30
31     Returns:
32     - structure: The complete structure as a TopoDS_Shape
33     """
34
35     # Convert angle from degrees to radians
36     angle_rad = math.radians(rafter_angle)
37
38     # Distance between opposite vertical columns
39     column_spacing_x = 2 * rafter_length * math.cos(angle_rad)
40
41     # Create vertical I-section columns
42     # Create vertical I-section columns (passing column_height as
43     # first parameter)
44     column = create_i_section(column_height, column_thickness,
45                               column_thickness, flange_thickness, web_thickness)
46
47     # Rotate to make it vertical
48     trsf = gp_Trsf()
49     trsf.SetRotation(gp_Ax1(gp_Pnt(0, 0, 0), gp_Dir(0, 1, 0)), -math.pi
50                 / 2) # Rotate 90 around Y-axis
51     column = BRepBuilderAPI_Transform(column, trsf, True).Shape()
52     columns = []
53
54     # Position columns
55     num_columns = 5
56     column_spacing_y = rafter_length
57     for i in range(num_columns):
58         # Left row
59         trsf = gp_Trsf()
60         trsf.SetTranslation(gp_Vec(0, i * column_spacing_y, 0))

```

```

56     left_column = BRepBuilderAPI_Transform(column, trsf, True).
57         Shape()
58     columns.append(left_column)
59
60     # Right row
61     trsf = gp_Trsf()
62     trsf.SetTranslation(gp_Vec(column_spacing_x+column_thickness, i
63         * column_spacing_y, 0))
64     right_column = BRepBuilderAPI_Transform(column, trsf, True).
65         Shape()
66     columns.append(right_column)
67
68     # Fuse columns into a single shape
69     structure = columns[0]
70     for col in columns[1:]:
71         structure = BRepAlgoAPI_Fuse(structure, col).Shape()
72
73     box = BRepPrimAPI_MakeBox(1000, 1000, 1000).Shape()    # Create a box
74         of size 1000x1000x100
75     for i in range(num_columns):
76         # Left column base position
77         column_base_left = gp_Vec(-column_thickness*2, i *
78             column_spacing_y-column_thickness*1.5, 0)
79         trsf_left = gp_Trsf()
80         trsf_left.SetTranslation(column_base_left)
81         box_left = BRepBuilderAPI_Transform(box, trsf_left, True).Shape
82             ()
83
84         # Right column base position
85         column_base_right = gp_Vec(column_spacing_x-column_thickness*2,
86             i * column_spacing_y-column_thickness*1.5, 0)
87         trsf_right = gp_Trsf()
88         trsf_right.SetTranslation(column_base_right)
89         box_right = BRepBuilderAPI_Transform(box, trsf_right, True).
90             Shape()
91
92         # Fuse the boxes into the structure
93         structure = BRepAlgoAPI_Fuse(structure, box_left).Shape()
94         structure = BRepAlgoAPI_Fuse(structure, box_right).Shape()

```

```

87
88     # Create inclined rafters for each column
89     rafter = create_i_section(rafter_length, column_thickness,
90                               column_thickness, flange_thickness, web_thickness)
91
92     for i in range(num_columns):
93         # Get column tops for left and right sides
94         left_column_top = gp_Pnt(0, i * column_spacing_y, column_height)
95
96         right_column_top = gp_Pnt(0, i * column_spacing_y,
97                                   column_height)
98
99         # Apex of the rafters (the center point above the two columns)
100        apex = gp_Pnt(column_spacing_x / 2, i * column_spacing_y,
101                      column_height + math.tan(angle_rad) * (column_spacing_x / 2))
102
103        # Left rafter for the current frame (connects left column to
104        # apex)
105        trsf = gp_Trsf()
106        trsf.SetTranslation(gp_Vec(0, i * column_spacing_y,
107                           column_height)) # Position at left column
108        rafter_left = BRepBuilderAPI_Transform(rafter, trsf, True).
109                      Shape()
110
111        # Right rafter for the current frame (connects right column to
112        # apex)
113        trsf = gp_Trsf()
114        trsf.SetTranslation(gp_Vec(column_spacing_x, i *
115                               column_spacing_y, column_height)) # Position at right
116                               column
117        rafter_right = BRepBuilderAPI_Transform(rafter, trsf, True).
118                      Shape()
119
120        # Adjust the rafter to the apex, and change its orientation
121        # accordingly
122        trsf_left = gp_Trsf()
123        trsf_left.SetTranslation(gp_Vec(0, i * column_spacing_y,
124                                     column_height)) # Left rafter base at the left column

```

```

112     trsf_left.SetRotation(gp_Ax1(gp_Pnt(0, i * column_spacing_y,
113         column_height), gp_Dir(0, 1, 0)), -angle_rad) # Rotate to
114         desired angle
115     rafter_left = BRepBuilderAPI_Transform(rafter_left, trsf_left,
116         True).Shape()
117
118     trsf_right = gp_Trsf()
119     adjust=20
120     trsf_right.SetTranslation(gp_Vec(column_spacing_x, i *
121         column_spacing_y, column_height)) # Right rafter base at
122         the right column
123     trsf_right.SetRotation(gp_Ax1(gp_Pnt(column_spacing_x+adjust, i
124         * column_spacing_y, column_height+column_thickness/2.1),
125         gp_Dir(0, 1, 0)), angle_rad-math.pi) # Rotate to desired
126         angle
127     rafter_right = BRepBuilderAPI_Transform(rafter_right,
128         trsf_right, True).Shape()
129
130     # Fuse the rafters into the structure
131     structure = BRepAlgoAPI_Fuse(structure, rafter_left).Shape()
132     structure = BRepAlgoAPI_Fuse(structure, rafter_right).Shape()
133
134     # Left Rafter
135     num_prisms = desired_number_of_prisms # Set the desired number of
136         prisms
137     prism_spacing = rafter_length * math.cos(angle_rad) / (num_prisms -
138         1) # Divide the rafter length by the gaps (num_prisms - 1)
139
140     for i in range(num_prisms):
141         if (i==num_prisms-1):
142             # Calculate the position along the rafter (X and Z coordinates)
143             x_position_left = i * prism_spacing+roof_width/2 # Along
144                 the rafter length (X-axis) for the left rafter
145             z_position_left = column_height + roof_height + math.tan(
146                 angle_rad) * x_position_left -column_thickness/2#
147                 Adjusted Z-position along the slope for the left rafter
148         else:
149             x_position_left = i * prism_spacing # Along the rafter
150                 length (X-axis) for the left rafter

```

```

136     z_position_left = column_height + roof_height + math.tan(
137         angle_rad) * x_position_left + flange_thickness*3.5 #
138         Adjusted Z-position along the slope for the left rafter
139
140     # Create a roof prism
141     roof_left = create_rectangular_prism(roof_length, roof_width,
142                                         roof_height)
143
144     # Translation to the position along the left rafter
145     translation_left = gp_Trsf()
146     translation_left.SetTranslation(gp_Vec(x_position_left, 0,
147                                         z_position_left)) # Move to the left rafter position
147     roof_left = BRepBuilderAPI_Transform(roof_left,
148                                         translation_left, True).Shape()
149
150     # Rotation to make the prism perpendicular to the left rafter
151     rotation_axis_left = gp_Ax1(gp_Pnt(x_position_left, 0,
152                                     z_position_left), gp_Dir(math.sin(0), 0, math.cos(0)))
153     rotation_left = gp_Trsf()
154     rotation_left.SetRotation(rotation_axis_left, math.pi / 2) #
155         Rotate 90 degrees to make it perpendicular
156     roof_left = BRepBuilderAPI_Transform(roof_left, rotation_left,
157                                         True).Shape()
158
159     # Fuse the prism into the left rafter structure
160     structure = BRepAlgoAPI_Fuse(structure, roof_left).Shape()
161
162     # Right Rafter
163     for i in range(num_prisms-1):
164         # Calculate the position along the rafter (X and Z coordinates)
165         x_position_right = column_spacing_x - i * prism_spacing
166         x_position = -i * prism_spacing # Along the rafter length (X-
167                                         axis) for the right rafter (negative direction)
168         z_position_right = column_height + math.tan(angle_rad) * abs(
169             x_position) + column_thickness*1.45 # Adjusted Z-position
170                                         along the slope for the right rafter
171
172         # Create a roof prism
173         roof_right = create_rectangular_prism(roof_length, roof_width,

```

```

    roof_height)

164
165     # Translation to the position along the right rafter
166     translation_right = gp_Trsf()
167     translation_right.SetTranslation(gp_Vec(x_position_right, 0,
168                                         z_position_right)) # Move to the right rafter position
168     roof_right = BRepBuilderAPI_Transform(roof_right,
169                                         translation_right, True).Shape()

170
171     # Rotation to make the prism perpendicular to the right rafter
172     rotation_axis_right = gp_Ax1(gp_Pnt(x_position_right, 0,
173                                     z_position_right), gp_Dir(math.sin(0), 0, math.cos(0)))
172     rotation_right = gp_Trsf()
173     rotation_right.SetRotation(rotation_axis_right, math.pi / 2) #
174         # Rotate 90 degrees to make it perpendicular
174     roof_right = BRepBuilderAPI_Transform(roof_right,
175                                         rotation_right, True).Shape()

176
177     # Fuse the prism into the right rafter structure
178     structure = BRepAlgoAPI_Fuse(structure, roof_right).Shape()

179
180
181
182 if __name__ == "__main__":
183     # Define dimensions
184     column_height = 4000.0
185     column_thickness = 250.0 #please don't change this value as many
186         # calculation done on this basis
186     rafter_length = 9000.0
187     rafter_angle = float(input("Enter the angle of inclination:\n"))
188     roof_length = rafter_length * 4 + 200
189     roof_width = 200.0
190     roof_height = 200.0
191     flange_thickness = 20
192     web_thickness = 9.10
193     desired_number_of_prisms = int(input("Enter the number of prisms:\n"
194                                         ""))
194     prism_spacing = rafter_length / (desired_number_of_prisms - 1)

```

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

Chapter 3

Bolted Butt Joint

3.1 Problem Statement

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

3.2 Task Done

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

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

3.3 Python Code

3.3.1 Description of the Script

The script is structured as follows:

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

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

- **Design Calculations:** The program calculates derived values such as plate length, bolt head and nut dimensions, and bolt positions using nested loops. It creates instances of `Plate`, `Bolt`, and `Nut` objects and places them at calculated positions to form the butt joint. The bolt arrangement is based on the specified pitch and gauge, and placement stops once the desired number of bolts is reached.
- **Output:** The script generates and visualizes a 3D CAD model of the bolted butt joint using the Open Cascade (OCC) display module. The output includes the top and bottom plates, a cover plate, and correctly positioned bolts and nuts. Components are displayed in distinct colors, and a small red sphere at the origin helps in identifying the global coordinate system.

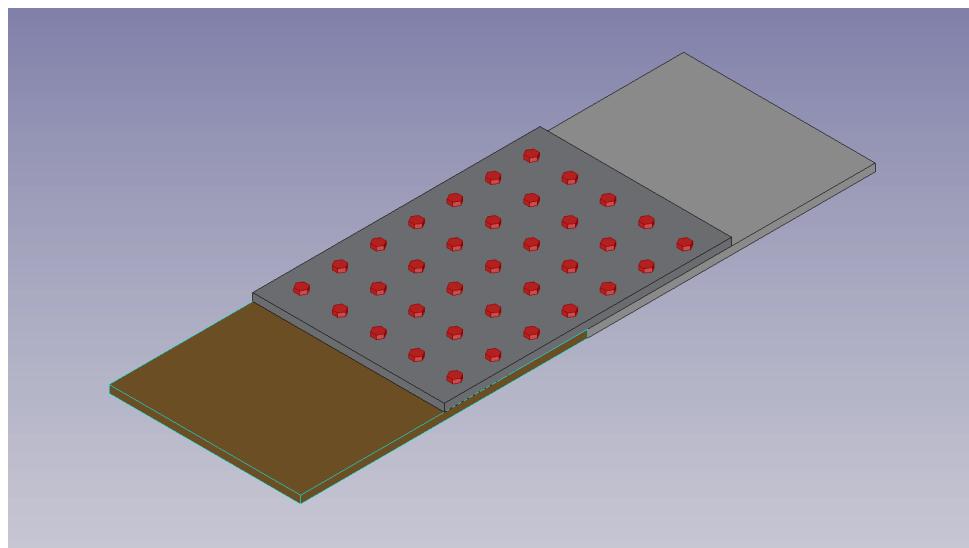


Figure 3.1: Single Plated Bolted Butt Joint

3.3.2 Python Script

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

Listing 3.1: Single Plated Bolted Butt Joint

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

```

4  from OCC.Display.SimpleGui import init_display
5  from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
6  from OCC.Core.BOPAlgo import BOPAlgo_Builder
7  from OCC.Core.Quantity import Quantity_NOC_SADDLEBROWN,
     Quantity_NOC_GRAY,Quantity_NOC_BLUE1,Quantity_NOC_RED,Quantity_Color
     , Quantity_TOC_RGB
8  from OCC.Core.Graphic3d import Graphic3d_NOM_ALUMINIUM,
     Graphic3d_NOM_STEEL
9  from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeSphere
10 # Import the component classes
11 from bolt import Bolt
12 from nut import Nut
13 from plate import Plate
14
15 def create_bolted_butJoint(plate1_thickness = 4, plate2_thickness =
16                           4, cover_thickness=3, plate_width = 100, bolt_dia = 16,
17                           bolt_rows=5,bolt_cols=7,pitch=20,gauge=20,
18                           edge=12,end=13.6,number_bolts=7):
19
20     plate_length = 1.5*plate_width
21     nut_thickness = 3.0
22     # Bolt parameters
23     bolt_head_radius = bolt_dia/2
24     bolt_head_thickness = 3.0
25     bolt_length = (plate1_thickness + plate2_thickness) + nut_thickness
26             # Enough to go through both plates
27     bolt_shaft_radius = 1.5
28
29     # Nut parameters
30     nut_radius = bolt_head_radius
31
32     nut_height = bolt_head_radius
33     nut_inner_radius = bolt_shaft_radius
34
35     # Create the first plate
36     # Position it at the origin
37     origin1 = numpy.array([0.0, 0.0, 0.0]) # Global origin lies at
38             midpoint of plate 1
39     uDir1 = numpy.array([0.0, 0.0, 1.0]) # Points along Z axis (height
40             )

```

```

35     wDir1 = numpy.array([1.0, 0.0, 0.0]) # Points along X axis (length
36         )
37
38     plate1 = Plate(plate_length, plate_width, plate1_thickness)
39     plate1.place(origin1, uDir1, wDir1)
40     plate1_model = plate1.create_model()
41
42     # Create the second plate
43     # Position it so that it properly overlaps with the first plate
44     # The second plate is elevated by plate1_thickness and offset in Y
45     # direction
46
47     origin2 = numpy.array([0.0, plate_length, 0])
48     uDir2 = numpy.array([0.0, 0.0, 1.0])
49     wDir2 = numpy.array([1.0, 0.0, 0.0])
50
51     plate2 = Plate(plate_length, plate_width, plate2_thickness)
52     plate2.place(origin2, uDir2, wDir2)
53     plate2_model = plate2.create_model()
54
55     origin3 = numpy.array([0.0, (plate_width*1.5)/2, max(
56         plate1_thickness, plate2_thickness)]) # Global origin lies at
57         midpoint of plate 1
58     uDir3 = numpy.array([0.0, 0.0, 1.0]) # Points along Z axis (height
59         )
60     wDir3 = numpy.array([1.0, 0.0, 0.0]) # Points along X axis (length
61         )
62
63     platec = Plate(plate_length, plate_width, plate1_thickness)
64     platec.place(origin3, uDir3, wDir3)
65     platec_model = platec.create_model()
66
67     # --- Bolt Dimensions ---
68
69     T = 2
70
71     R, H, r = 3, (max(plate1_thickness, plate2_thickness) +
72         cover_thickness + T) * 1.25, 1
73
74     innerR1 = 1
75
76
77     # --- Calculate Bolt Positions ---
78     bolt_positions = []

```

```

67     count = 0
68     exit_loops = False
69
70     for col in range(bolt_cols):
71         for row in range(bolt_rows):
72             bolt_positions.append((
73                 edge + (row * gauge),
74                 end + (col * pitch),
75                 (max(plate1_thickness, plate2_thickness))/2 +
76                 cover_thickness+T/2
77             ))
78             count += 1
79
80             # Exit after completing current column
81             if count == number_bolts and row == bolt_rows - 1:
82                 exit_loops = True
83                 break
84
85             if exit_loops:
86                 break
87
88             # --- Create and Place Bolts & Nuts ---
89             bolts_models = []
90             nuts_models = []
91             bolt_uDir = numpy.array([1.0, 0.0, 0.0])
92             bolt_shaftDir = numpy.array([0.0, 0.0, -1.0])
93
94             for pos in bolt_positions:
95                 # Bolt
96                 bolt = Bolt(R, T, H, r)
97                 bolt.place(pos, bolt_uDir, bolt_shaftDir)
98                 bolt_model = bolt.create_model()
99                 bolts_models.append(bolt_model)
100
101                 # Nut
102                 nut_origin = numpy.array([pos[0], pos[1], -T])
103                 nut_uDir = numpy.array([1.0, 0.0, 0.0])
104                 nut_wDir = numpy.array([0.0, 0.0, -1.0])
105
106                 nut = Nut(R, T, H, innerR1)

```

```

105     nut.place(nut_origin, nut_uDir, nut_wDir)
106     nut_model = nut.create_model()
107     nuts_models.append(nut_model)
108
109     return plate1_model, plate2_model, platec_model, bolts_models,
110         nuts_models
111
112 # Main execution
113 if __name__ == "__main__":
114     # Create the bolted lap joint
115     plate1, plate2, platec, bolts, nuts = create_bolted_butttJoint()
116
117     redd=Quantity_Color(0.28, 0, 0, Quantity_TOC_RGB)
118
119     # Display the assembly
120     display, start_display, add_menu, add_function_to_menu =
121         init_display()
122
123     # Display individual components with different colors for better
124     # visualization
125     display.DisplayShape(plate1, update=True)
126     display.DisplayShape(plate2, material=Graphic3d_NOM_ALUMINIUM,
127                         update=True)
128     display.DisplayShape(platec, material=Graphic3d_NOM_STEEL, update=
129                         True)
130
131     # --- Display Bolts and Nuts ---
132     for bolt_model in bolts:
133         display.DisplayShape(bolt_model, color=redd, update=True)
134
135     for nut_model in nuts:
136         display.DisplayShape(nut_model, color=redd, update=True)
137
138     # display.DisplayShape(nut, color=Quantity_NOC_SADDLEBROWN, update=
139     #                     True)
140
141     # Highlight the global origin (0,0,0)
142     origin_point = BRepPrimAPI_MakeSphere(1).Shape()    # Small sphere to
143     # mark origin
144
145     display.DisplayShape(origin_point, color=Quantity_NOC_RED, update=

```

```

    True)

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

```

3.3.3 Explanation of the Code

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

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

3.4 Documentation

My contribution your contribution towards Osdag on the directory below:

3.4.1 Directory Structure

```

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

```

Chapter 4

Welded Butt Joint

4.1 Problem Statement

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

4.2 Task Done

Welded Butt Joint

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

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

4.3 Python Code

4.3.1 Description of the Script

The script is structured as follows:

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

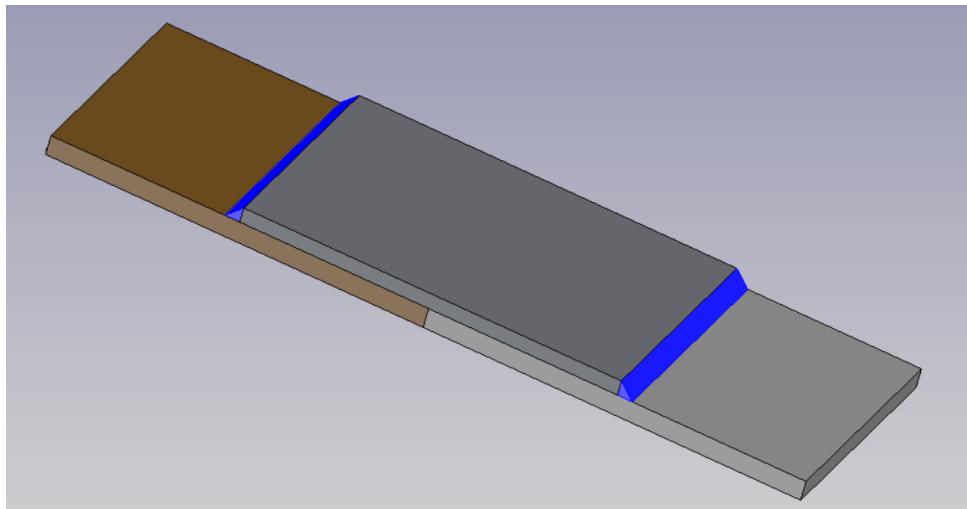


Figure 4.1: Single Plated Welded Butt Joint

4.3.2 Python Script

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

Listing 4.1: Single Plated Welded Butt Joint

```
1 %-----begin code-----
2
3 import numpy
4 from OCC.Display.SimpleGui import init_display
5 from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeSphere
6 from OCC.Core.Quantity import Quantity_NOC_RED, Quantity_Color,
7     Quantity_TOC_RGB
8 from OCC.Core.Graphic3d import Graphic3d_NOM_ALUMINIUM,
9     Graphic3d_NOM_STEEL
10 from OCC.Core.gp import gp_Pnt
11 from plate import Plate
12 from filletweld import FilletWeld # Import FilletWeld class
13
14 def create_welded_butt_joint(plate_width=100, plate1_thickness=10,
15     plate2_thickness=10, cover_thickness=8, weld_size=6, weld_length=80):
16     :
17     plate_length = 1.5 * plate_width
18     weld_length=plate_width
19     # --- Create the first plate ---
20     origin1 = numpy.array([0.0, 0.0, 0.0])
21     uDir1 = numpy.array([0.0, 0.0, 1.0]) # X direction
22     wDir1 = numpy.array([1.0, 0.0, 0.0]) # Z direction
23
24     plate1 = Plate(plate_length, plate_width, plate1_thickness)
25     plate1.place(origin1, uDir1, wDir1)
26     plate1_model = plate1.create_model()
27
28     # --- Create the second plate ---
29     origin2 = numpy.array([0.0, plate_length, 0.0])
30     uDir2 = numpy.array([0.0, 0.0, 1.0])
31     wDir2 = numpy.array([1.0, 0.0, 0.0])
32
33     plate2 = Plate(plate_length, plate_width, plate2_thickness)
34     plate2.place(origin2, uDir2, wDir2)
```

```

31 plate2_model = plate2.create_model()
32
33 # --- Create the cover plate ---
34 origin3 = numpy.array([0.0, plate_length/ 2, max(plate1_thickness,
35                      plate2_thickness)/2+cover_thickness/2])
36 uDir3 = numpy.array([0.0, 0.0, 1.0])
37 wDir3 = numpy.array([1.0, 0.0, 0.0])
38
39 cover_plate = Plate(plate_length, plate_width, cover_thickness)
40 #cover_plate_center_origin = origin3 + numpy.array([0, plate_length
41           / 2, cover_thickness / 2])
42 cover_plate.place(origin3, uDir3, wDir3)
43 cover_plate_model = cover_plate.create_model()
44
45
46 # --- Create the Fillet Welds ---
47 welds_models = []
48
49 # Corrected weld placement for each butt interface
50 weld_origins = [
51     numpy.array([(plate_width - weld_length) / 2, 0, max(
52                 plate1_thickness, plate2_thickness)/2]), # Left weld
53     numpy.array([(plate_width - weld_length) / 2, plate_length, max(
54                 plate1_thickness, plate2_thickness)/2]) # Right weld
55 ]
56
57 weld_dirs = [
58     {'uDir': numpy.array([0.0, 0.0, 1.0]), 'shaftDir': numpy.array
59         ([1.0, 0.0, 0.0])}, # Upward facing weld
60     {'uDir': numpy.array([0.0, 1.0, 0.0]), 'shaftDir': numpy.array
61         ([1.0, 0.0, 0.0])} # Downward facing weld
62 ]
63
64 for i, origin in enumerate(weld_origins):
65     weld = FilletWeld(weld_b, weld_h, weld_length)
66     weld.place(origin, weld_dirs[i]['uDir'], weld_dirs[i]['shaftDir']
67                )

```

```

63     weld_model = weld.create_model()
64     welds_models.append(weld_model)
65
66     return plate1_model, plate2_model, cover_plate_model, welds_models
67
68 #
=====

69 if __name__ == "__main__":
70     plate1, plate2, cover_plate, welds = create_welded_butJoint(
71         plate_width=120,
72         plate1_thickness=12,
73         plate2_thickness=12,
74         cover_thickness=10,
75         weld_size=8,
76         weld_length=100
77     )
78
79 # Initialize 3D viewer
80 display, start_display, add_menu, add_function_to_menu =
81     init_display()
82
83 weld_color = Quantity_Color(0.8, 0.1, 0.1, Quantity_TOC_RGB)
84
85 # Display all parts
86 display.DisplayShape(plate1, update=True, color='silver')
87 display.DisplayShape(plate2, update=True, material=
88     Graphic3d_NOM_ALUMINIUM)
89 display.DisplayShape(cover_plate, update=True, material=
90     Graphic3d_NOM_STEEL)
91
92 for weld_model in welds:
93     display.DisplayShape(weld_model, color=weld_color, update=True)
94
95 # Mark origin
96 origin_marker = BRepPrimAPI_MakeSphere(gp_Pnt(0, 0, 0), 1.5).Shape
97     ()
98 display.DisplayShape(origin_marker, color=Quantity_NOC_RED, update=
99     True)

```

```

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

```

4.3.3 Explanation of the Code

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

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

Chapter 5

Double Angle Gusset Joint

5.1 Problem Statement

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

5.2 Task Done

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

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

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

5.3 Python Code(Bolted gusset joint)

5.3.1 Description of the Script

The script is structured as follows:

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



Figure 5.1: Double Angle Bolted Gusset Plate Joint

5.3.2 Python Script

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

Listing 5.1: Double Angle Bolted Gusset Plate Joint

```

1 %-----begin code-----
2
3 from OCC.Core.gp import gp_Vec, gp_Trsf, gp_Pnt, gp_Ax1, gp_Dir
4 from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox
5 from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
6 from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_Transform
7 from OCC.Display.SimpleGui import init_display
8 from angle_create import create_angle_section
9 from bolt import Bolt # Import the Bolt class
10 from nut import Nut
11 import numpy
12 from trapizoid_plate import create_trapizoid
13 from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_MakePolygon,
14     BRepBuilderAPI_MakeFace
14 from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakePrism
15 from OCC.Core.gp import gp_Vec
16
17 from OCC.Core.Quantity import Quantity_NOC_SADDLEBROWN #for bolt colors
18 from OCC.Core.Graphic3d import * #for gray colour
19 from OCC.Core.Graphic3d import Graphic3d_NOM_ALUMINIUM,
20     Graphic3d_NOM_JADE,Graphic3d_NOM_OBSIDIAN,Graphic3d_NOM_NEON_PHC,
21     Graphic3d_NOM_STEEL
22
23 # Visualization
24 display, start_display, add_menu, add_function_to_menu = init_display()
25
26 length = 1000
27 width = 20 # Width of the angle
28 depth = 20 # Height of angle
29 flange_thickness = 4 # Thickness of the flanges
30 web_thickness = 4 # Thickness of the web
31 plate_width=100
32 plate_thickness=6
33 large_len=200
34 small_len=40

```

```

35 trap_height=125
36
37
38 # Define trapezoid corner points (parallel sides along Y-axis)
39 p1 = gp_Pnt(0, 0, 0)      # Bottom-left (short side)
40 p2 = gp_Pnt(0, large_len, 0) # Top-left (short side)
41 p3 = gp_Pnt(trap_height, 160, 0) # Top-right (wide side)
42 p4 = gp_Pnt(trap_height, 40, 0) # Bottom-right (wide side)
43
44 # Create a polygon (wire)
45 polygon = BRepBuilderAPI_MakePolygon()
46 polygon.Add(p1)
47 polygon.Add(p2)
48 polygon.Add(p3)
49 polygon.Add(p4)
50 polygon.Close()
51
52 # Create a face from the polygon
53 face = BRepBuilderAPI_MakeFace(polygon.Wire())
54
55 # Extrude to make a 3D plate (thickness of 5 units)
56
57 trap_plate1 = BRepPrimAPI_MakePrism(face.Face(), gp_Vec(0, 0,
58                                         plate_thickness)).Shape()
59
60 # Define transformation for rotation
61 trsf = gp_Trsf()
62 rotation_axis = gp_Ax1(gp_Pnt(0, 0, 0), gp_Dir(0, 0, 1)) # Z-axis
63 angle = 180 * (3.14159265 / 180) # Convert degrees to radians
64 trsf.SetRotation(rotation_axis, angle)
65 trap_plate2r = BRepBuilderAPI_Transform(trap_plate1, trsf, True).Shape()
66
67 # transformation after rotation plate2
68 trsf = gp_Trsf()
69 trsf.SetTranslation(gp_Vec(length+trap_height, large_len, 0)) # Move to
70 the top
71 trap_plate2 = BRepBuilderAPI_Transform(trap_plate2r, trsf, True).Shape

```

```

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

```

```

104     numpy.array([(5*trap_height)/6., (large_len-2*width)/2+(width*3)
105                  /2., plate_thickness+flange_thickness]), # Bolt 4
106     numpy.array([length+trap_height-(2*trap_height)/3., (large_len-2*
107                  width)/2+width/2., plate_thickness+flange_thickness]), # Bolt
108                  1
109
110     numpy.array([length+trap_height-(5*trap_height)/6., (large_len-2*
111                  width)/2+width/2., plate_thickness+flange_thickness]), # Bolt
112                  2
113     numpy.array([length+trap_height-(2*trap_height)/3., (large_len-2*
114                  width)/2+(width*3)/2., plate_thickness+flange_thickness]), # Bolt
115                  3
116     numpy.array([length+trap_height-(5*trap_height)/6., (large_len-2*
117                  width)/2+(width*3)/2., plate_thickness+flange_thickness]) # Bolt
118                  4
119 ]
120
121
122
123 # Bolt orientation
124 uDir = numpy.array([1., 0., 0.]) # X-direction
125 shaftDir = numpy.array([0., 0., -1.]) # Z-direction
126
127 # Generate and display bolts
128 for pos in bolt_positions:
129     bolt = Bolt(R, T, H, r)
130     bolt.place(pos, uDir, shaftDir)
131     display.DisplayShape(bolt.create_model(), color="blue", update=True)
132
133
134 #create nuts
135 innerR1 =1
136
137 # Nut positions
138 nut_positions = [
139
140     numpy.array([(2*trap_height)/3., (large_len-2*width)/2+width/2., -
141                  T]), # Bolt 1
142     numpy.array([(5*trap_height)/6., (large_len-2*width)/2+width/2., -
143                  T]), # Bolt 2
144     numpy.array([(2*trap_height)/3., (large_len-2*width)/2+(width*3)

```

```

    /2., -T]), # Bolt 3
132 numpy.array([(5*trap_height)/6., (large_len-2*width)/2+(width*3)
    /2., -T]), # Bolt 4
133 numpy.array([length+trap_height-(2*trap_height)/3., (large_len-2*
    width)/2+width/2., -T]), # Bolt 1
134 numpy.array([length+trap_height-(5*trap_height)/6., (large_len-2*
    width)/2+width/2., -T]), # Bolt 2
135 numpy.array([length+trap_height-(2*trap_height)/3., (large_len-2*
    width)/2+(width*3)/2., -T]), # Bolt 3
136 numpy.array([length+trap_height-(5*trap_height)/6., (large_len-2*
    width)/2+(width*3)/2., -T]) # Bolt 4
137
138 ]
139
140 # Orientation
141 uDir = numpy.array([1., 0., 0.])
142 wDir = numpy.array([0., 0., 1.])
143
144 # Generate and display nuts
145 for pos in nut_positions:
146     nut = Nut(R, T, H, innerR1)
147     nut.place(pos, uDir, wDir)
148     display.DisplayShape(nut.create_model(), color="blue", update=True)
149
150
151
152
153 # Show the I-section model
154 #display.DisplayShape(trap_plate1, update=True)
155 display.DisplayShape(trap_plate2, material=Graphic3d_NOM_ALUMINIUM,
    update=True)
156 display.DisplayShape(angle_transform, update=True)
157 display.DisplayShape(trap_plate1, material=Graphic3d_NOM_ALUMINIUM,
    update=True)
158 display.FitAll()
159 start_display()
160
161 %----- end code -----
```

5.3.3 Explanation of the Code

- **Lines 1–5:** Imports core modules from the OpenCascade (OCC) library needed for geometric modeling, transformations, primitives, and visualization. Modules like `gp_Vec`, `BRepPrimAPI_MakeBox`, and `BRepAlgoAPI_Fuse` are fundamental for 3D modeling.
- **Lines 6–11:** Imports custom utility functions and classes like `create_angle_section`, `Bolt`, `Nut`, and `create_trapezoid`, which are assumed to be user-defined scripts to modularize geometry creation.
- **Lines 13–16:** Initializes the OCC 3D viewer using `init_display()`, enabling interactive visualization of the created CAD models.
- **Lines 18–23:** Defines geometry parameters for the angle section and trapezoidal gusset plates. This includes rod dimensions (`length`, `width`, `depth`), plate thickness, and gusset dimensions such as the longer and shorter bases and height.
- **Lines 26–33:** Sets up 2D corner points of a trapezoidal plate and constructs a wire polygon using these points. These define the base face for the gusset plate.
- **Lines 35–37:** Converts the polygon into a planar face, which is required for 3D extrusion.
- **Lines 39–41:** Extrudes the trapezoidal face along the Z-axis to form a solid 3D gusset plate.
- **Lines 44–47:** Rotates the first gusset plate 180 degrees about the Z-axis to mirror it for the opposite end.
- **Lines 49–51:** Translates the rotated plate to the other end of the rod to complete the symmetric gusset placement.
- **Line 54:** Creates the L-shaped angle section using a helper function `create_angle_section` with parametric inputs.
- **Lines 56–58:** Translates the angle section so that it fits precisely between the two gusset plates, accounting for the plate and flange thickness.

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

5.4 Python Code(Welded gusset joint)

5.4.1 Description of the Script

The script is structured as follows:

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

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

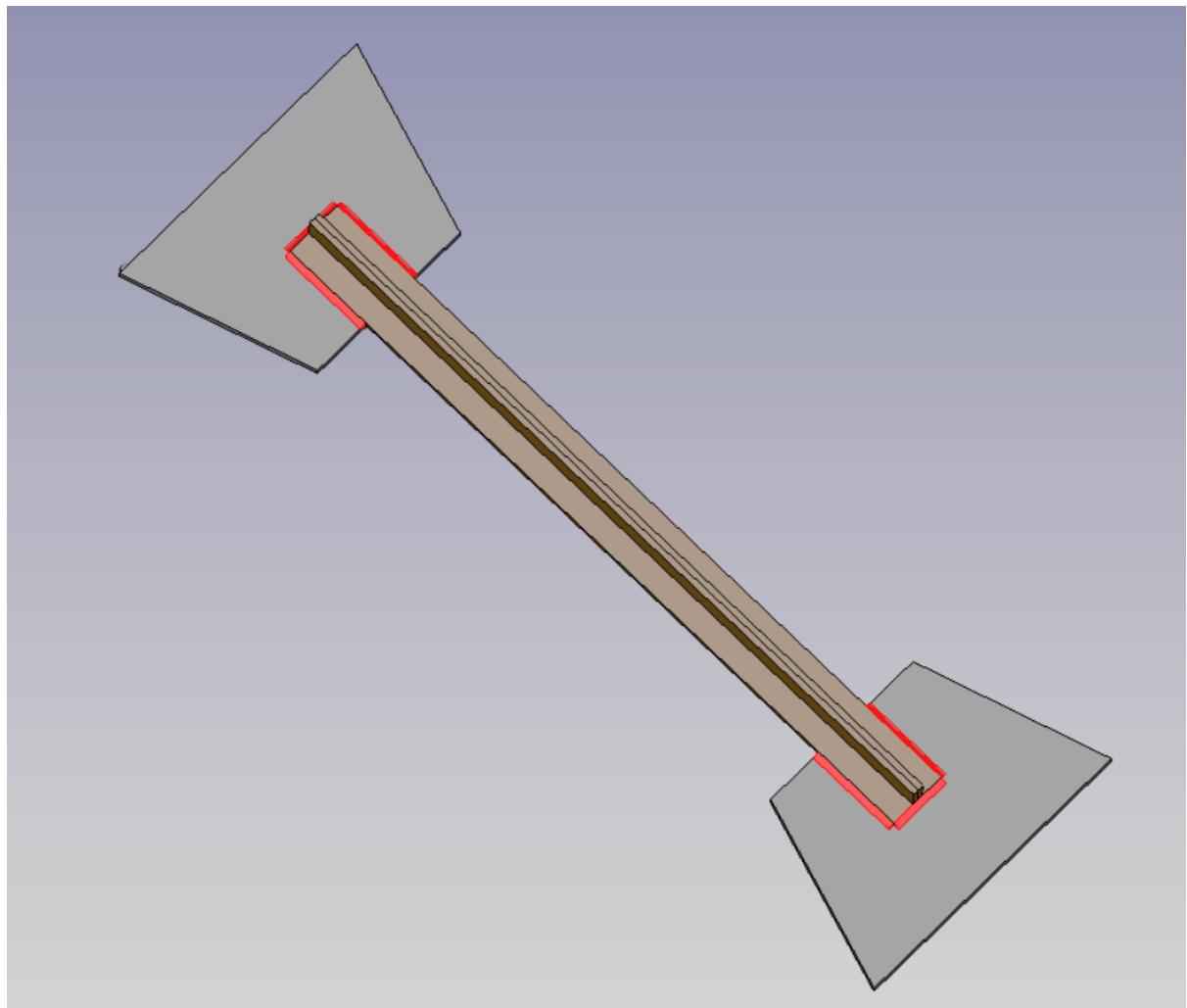


Figure 5.2: Double angle gusset weld joint

5.4.2 Python Script

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

Listing 5.2: Double angle gusset weld joint

```
1 %-----begin code-----
2
3 from OCC.Core.gp import gp_Vec, gp_Trsf, gp_Pnt, gp_Ax1, gp_Dir
4 from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox
5 from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
6 from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_Transform
7 from OCC.Display.SimpleGui import init_display
8 from angle_create import create_angle_section
9 from filletweld import FilletWeld # Import FilletWeld class
10 import numpy
11 from trapizoid_plate import create_trapizoid
12 from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_MakePolygon,
13     BRepBuilderAPI_MakeFace
14 from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakePrism
15
16 from OCC.Core.Quantity import Quantity_Color, Quantity_TOC_RGB
17 from OCC.Core.Aspect import Aspect_GFM_VER # Import the correct
18     gradient mode
19
20 from OCC.Core.Quantity import Quantity_NOC_SADDLEBROWN #for bolt colors
21 from OCC.Core.Graphic3d import * #for gray colour
22 from OCC.Core.Graphic3d import Graphic3d_NOM_ALUMINIUM,
23     Graphic3d_NOM_JADE, Graphic3d_NOM_OBSIDIAN, Graphic3d_NOM_NEON_PHC,
24     Graphic3d_NOM_STEEL
25
26
27 # Visualization
28 display, start_display, add_menu, add_function_to_menu = init_display()
29
30 top_color = Quantity_Color(0.27, 0.27, 0.44, Quantity_TOC_RGB) # Dark
31     blue (#444470)
32 bottom_color = Quantity_Color(0.67, 0.67, 0.67, Quantity_TOC_RGB) #
33     Light gray (#AAAAAA)
```

```

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

```

```

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

68

69

70 # Define transformation for rotation
71 trsf = gp_Trsf()
72 rotation_axis = gp_Ax1(gp_Pnt(0, 0, 0), gp_Dir(0, 0, 1)) # Z-axis
73 angle = 180 * (3.14159265 / 180) # Convert degrees to radians
74 trsf.SetRotation(rotation_axis, angle)
75 trap_plate2r = BRepBuilderAPI_Transform(trap_plate1, trsf, True).Shape()
76

77 # transformation after rotation plate2
78 trsf = gp_Trsf()
79 trsf.SetTranslation(gp_Vec(length+trap_height, large_len, 0)) # Move to
    the top
80 trap_plate2 = BRepBuilderAPI_Transform(trap_plate2r, trsf, True).Shape()
81
82
83
84
85 angle=create_angle_section(length, width, depth, flange_thickness,
    web_thickness)
86
87 trsf = gp_Trsf()
88 trsf.SetTranslation(gp_Vec(trap_height/2,(large_len-2*width)/2,
    plate_thickness)) # Move to the top
89 angle_transform = BRepBuilderAPI_Transform(angle, trsf, True).Shape()
90
91 """
92 p1=(0,0,0)
93 p2=(0, large_len, 0)
94 p3=(trap_height , (large_len-small_len)/2+small_len, 0)
95 p4=(trap_height , (large_len-small_len)/2, 0)
96
97
98 trap_plate1=create_trapizoid(p1,p2,p3,p4)
99

```

```

100 trsf = gp_Trsf()
101 trsf.SetTranslation(gp_Vec(500, 0, 0)) # Move to the top
102 trap_plate2= BRepBuilderAPI_Transform(trap_plate1, trsf, True).Shape()
103 """
104
105
106 # Define parameters for the fillet weld
107 b = flange_thickness
108 h = flange_thickness
109
110 # Define multiple origins and directions for placing the fillet welds
111 origins = [
112     numpy.array([trap_height/2,(large_len-2*width)/2, plate_thickness])
113     ,
114     numpy.array([length,(large_len-2*width)/2, plate_thickness]),
115     numpy.array([trap_height/2,(large_len-2*width)/2+2*width,
116                 plate_thickness]),
117     numpy.array([length,(large_len-2*width)/2+2*width, plate_thickness
118                 ]),
119     numpy.array([trap_height/2,(large_len-2*width)/2, plate_thickness])
120     ,
121     numpy.array([length+trap_height/2,(large_len-2*width)/2,
122                 plate_thickness]),# First placement, # Second placement
123 ]
124
125 #uDir = numpy.array([0.0, 0.0, 1.0]) # Common uDir for all placements
126 #shaftDir = numpy.array([0.0, 1.0, 0.0]) # Common wDir for all
127 # placements
128
129 # Create and display multiple fillet welds
130 for i, origin in enumerate(origins):
131     if i<=3:
132         shaftDir = numpy.array([1.0, 0.0, 0.0])
133         L=trap_height/2
134         if i<=1:
135             uDir = numpy.array([0.0, 0.0, 1.0])
136         else:
137             uDir = numpy.array([0.0, 1.0, 0.0])

```

```

133     else:
134         shaftDir = numpy.array([0.0, 1.0, 0.0])
135         L=width*2
136         if i==4:
137             uDir = numpy.array([-1.0, 0.0, 0.0])
138         else:
139             uDir = numpy.array([0.0, 0.0, 1.0])
140
141
142     # Create a fillet weld object
143     FWeld = FilletWeld(b, h, L)
144
145     # Place the fillet weld at the specified origin and direction
146     FWeld.place(origin, uDir, shaftDir)
147
148     # Create the 3D model (prism) for the fillet weld
149     prism = FWeld.create_model()    # Rotate the second weld
150     display.DisplayShape(prism,color=redd, update=True)
151
152
153
154
155     # Add a message at the origin
156     Point = gp_Pnt(0.0, 0.0, 0.0)
157     display.DisplayMessage(Point, "Origin")
158
159     # Show the I-section model
160     #display.DisplayShape(trap_plate1, update=True)
161     display.DisplayShape(trap_plate2,material=Graphic3d_NOM_ALUMINIUM,
162                           update=True)
163     display.DisplayShape(angle_transform, update=True)
164     display.DisplayShape(trap_plate1,material=Graphic3d_NOM_ALUMINIUM,
165                           update=True)
166     display.FitAll()
167     start_display()
168
169
170
171
172
173
174
175
176
177
178 %----- end code -----
```

5.4.3 Explanation of the Code

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

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

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

Chapter 6

CAD Integration

6.1 Problem Statement

To integrate the CAD generation script into Osdag.

6.2 Task Done

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

6.3 Python Code inside common_logic.py

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

6.3.1 Python Script

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

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

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

```

35     butt_joint, plate1, plate2, platec, bolts, nuts =
36         create_bolted_butt_joint(self.plate1_thickness, self.
37             plate2_thickness, self.cover_thickness, self.plate_width
38             , self.bolt_dia,
39                 self.bolt_rows, self.bolt_cols, self.pitch,
40                     self.gauge, self.edge, self.end, self.
41                         number_bolts)
42
43     return butt_joint, plate1, plate2, platec, bolts, nuts
44
45
46
47
48     # Call function to create the welded joint
49     plate1_model, plate2_model, cover_plate_model, welds_models =
50         create_welded_butt_joint(
51             plate_width=self.plate_width,
52             plate1_thickness=self.plate1_thickness,
53             plate2_thickness=self.plate2_thickness,
54             cover_thickness=self.cover_thickness,
55             weld_size=self.weld_size,
56             weld_length=self.weld_length
57         )
58
59     self.nuts_models = []      #      prevents crash for welded
60     self.bolt_models = []      #      optional
61
62     return plate1_model, plate2_model, cover_plate_model,
63         welds_models

```

6.3.2 Explanation of the Code

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

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

Listing 6.2: Snippet of code for method for Bolted and Welded Butt Joint CAD generation inside common.logic.py

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

```

7      if self.component == "Model":
8          osdag_display_shape(self.display, self.plate1_model,
9              update=True, material=Graphic3d_NOM_ALUMINIUM)
10         osdag_display_shape(self.display, self.plate2_model,
11             update=True)
12         osdag_display_shape(self.display, self.platec_model,
13             update=True)
14
15         if hasattr(self, 'bolt_models'):
16             for bolt in self.bolt_models:
17                 osdag_display_shape(self.display, bolt, update=
18                     True, color=Quantity_NOC_SADDLEBROWN)
19
20         if hasattr(self, 'nuts_models'):
21             for nut in self.nuts_models:
22                 osdag_display_shape(self.display, nut,
23                     update=True, color=
24                         Quantity_NOC_SADDLEBROWN)
25
26     elif self.mainmodule == 'Butt Joint Welded Connection':
27         self.col = self.module_class()
28         self.plate1_model, self.plate2_model, self.
29             cover_plate_model, self.welds_models = self.
30             createWeldedButtJoint()
31
32
33     if self.component == "Model":
34         osdag_display_shape(self.display, self.plate1_model,
35             update=True, material=Graphic3d_NOM_ALUMINIUM)
36         osdag_display_shape(self.display, self.plate2_model,
37             update=True)
38         osdag_display_shape(self.display, self.
39             cover_plate_model, update=True)
40         if hasattr(self, 'welds_models'):
41             for weld in self.welds_models:
42                 osdag_display_shape(self.display, weld, update=
43                     True, color=Quantity_NOC_SADDLEBROWN)
44         for nut in self.nuts_models:
45             osdag_display_shape(self.display, nut, update=True,
46                 color=Quantity_NOC_SADDLEBROWN)

```

6.3.3 Full code

Listing 6.3: Snippet of code for method for KEY specification for Bolted and Welded Butt Joint inside ui_tempalte.py

```
1
2     def display_3DModel(self , component , bgcolor):
3
4         self.component = component
5
6         self.display.EraseAll()
7
8         self.display.View_Iso()
9
10        self.display.FitAll()
11
12        self.display.DisableAntiAliasing()
13
14        if bgcolor == "gradient_bg":
15
16            self.display.set_bg_gradient_color([51, 51, 102] , [150 ,
17                                         150, 170])
18
19        else:
20
21            self.display.set_bg_gradient_color([255, 255, 255] , [255 ,
22                                         255, 255])
23
24
25
26
27        if self.mainmodule == "Shear Connection":
28
29            A = self.module_class()
30
31            self.loc = A.connectivity
32
33
34
35
36
37        if self.loc == "Column Flange-Beam Web" and self.connection
38
39            == KEY_DISP_FINPLATE:
40
41            # pass
42
43            # print("hghghghg")
44
45            self.display.View.SetProj(OCC.Core.V3d.V3d_XnegYnegZpos
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
127
128
129
129
130
131
132
133
134
135
136
137
138
139
139
140
141
142
143
144
145
146
147
148
149
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
187
188
189
189
190
191
192
193
194
195
196
197
197
198
199
199
200
201
202
203
204
205
206
207
207
208
209
209
210
211
212
213
214
215
215
216
217
217
218
219
219
220
221
222
223
223
224
225
225
226
227
227
228
229
229
230
231
232
233
233
234
235
235
236
237
237
238
239
239
240
241
241
242
243
243
244
245
245
246
247
247
248
249
249
250
251
251
252
253
253
254
255
255
256
257
257
258
259
259
260
261
261
262
263
263
264
265
265
266
267
267
268
269
269
270
271
271
272
273
273
274
275
275
276
277
277
278
279
279
280
281
281
282
283
283
284
285
285
286
287
287
288
289
289
290
291
291
292
293
293
294
295
295
296
297
297
298
299
299
300
301
301
302
303
303
304
305
305
306
307
307
308
309
309
310
311
311
312
313
313
314
315
315
316
317
317
318
319
319
320
321
321
322
323
323
324
325
325
326
327
327
328
329
329
330
331
331
332
333
333
334
335
335
336
337
337
338
339
339
340
341
341
342
343
343
344
345
345
346
347
347
348
349
349
350
351
351
352
353
353
354
355
355
356
357
357
358
359
359
360
361
361
362
363
363
364
365
365
366
367
367
368
369
369
370
371
371
372
373
373
374
375
375
376
377
377
378
379
379
380
381
381
382
383
383
384
385
385
386
387
387
388
389
389
390
391
391
392
393
393
394
395
395
396
397
397
398
399
399
400
401
401
402
403
403
404
405
405
406
407
407
408
409
409
410
411
411
412
413
413
414
415
415
416
417
417
418
419
419
420
421
421
422
423
423
424
425
425
426
427
427
428
429
429
430
431
431
432
433
433
434
435
435
436
437
437
438
439
439
440
441
441
442
443
443
444
445
445
446
447
447
448
449
449
450
451
451
452
453
453
454
455
455
456
457
457
458
459
459
460
461
461
462
463
463
464
465
465
466
467
467
468
469
469
470
471
471
472
473
473
474
475
475
476
477
477
478
479
479
480
481
481
482
483
483
484
485
485
486
487
487
488
489
489
490
491
491
492
493
493
494
495
495
496
497
497
498
499
499
500
501
501
502
503
503
504
505
505
506
507
507
508
509
509
510
511
511
512
513
513
514
515
515
516
517
517
518
519
519
520
521
521
522
523
523
524
525
525
526
527
527
528
529
529
530
531
531
532
533
533
534
535
535
536
537
537
538
539
539
540
541
541
542
543
543
544
545
545
546
547
547
548
549
549
550
551
551
552
553
553
554
555
555
556
557
557
558
559
559
560
561
561
562
563
563
564
565
565
566
567
567
568
569
569
570
571
571
572
573
573
574
575
575
576
577
577
578
579
579
580
581
581
582
583
583
584
585
585
586
587
587
588
589
589
590
591
591
592
593
593
594
595
595
596
597
597
598
599
599
600
601
601
602
603
603
604
605
605
606
607
607
608
609
609
610
611
611
612
613
613
614
615
615
616
617
617
618
619
619
620
621
621
622
623
623
624
625
625
626
627
627
628
629
629
630
631
631
632
633
633
634
635
635
636
637
637
638
639
639
640
641
641
642
643
643
644
645
645
646
647
647
648
649
649
650
651
651
652
653
653
654
655
655
656
657
657
658
659
659
660
661
661
662
663
663
664
665
665
666
667
667
668
669
669
670
671
671
672
673
673
674
675
675
676
677
677
678
679
679
680
681
681
682
683
683
684
685
685
686
687
687
688
689
689
690
691
691
692
693
693
694
695
695
696
697
697
698
699
699
700
701
701
702
703
703
704
705
705
706
707
707
708
709
709
710
711
711
712
713
713
714
715
715
716
717
717
718
719
719
720
721
721
722
723
723
724
725
725
726
727
727
728
729
729
730
731
731
732
733
733
734
735
735
736
737
737
738
739
739
740
741
741
742
743
743
744
745
745
746
747
747
748
749
749
750
751
751
752
753
753
754
755
755
756
757
757
758
759
759
760
761
761
762
763
763
764
765
765
766
767
767
768
769
769
770
771
771
772
773
773
774
775
775
776
777
777
778
779
779
780
781
781
782
783
783
784
785
785
786
787
787
788
789
789
790
791
791
792
793
793
794
795
795
796
797
797
798
799
799
800
801
801
802
803
803
804
805
805
806
807
807
808
809
809
810
811
811
812
813
813
814
815
815
816
817
817
818
819
819
820
821
821
822
823
823
824
825
825
826
827
827
828
829
829
830
831
831
832
833
833
834
835
835
836
837
837
838
839
839
840
841
841
842
843
843
844
845
845
846
847
847
848
849
849
850
851
851
852
853
853
854
855
855
856
857
857
858
859
859
860
861
861
862
863
863
864
865
865
866
867
867
868
869
869
870
871
871
872
873
873
874
875
875
876
877
877
878
879
879
880
881
881
882
883
883
884
885
885
886
887
887
888
889
889
890
891
891
892
893
893
894
895
895
896
897
897
898
899
899
900
901
901
902
903
903
904
905
905
906
907
907
908
909
909
910
911
911
912
913
913
914
915
915
916
917
917
918
919
919
920
921
921
922
923
923
924
925
925
926
927
927
928
929
929
930
931
931
932
933
933
934
935
935
936
937
937
938
939
939
940
941
941
942
943
943
944
945
945
946
947
947
948
949
949
950
951
951
952
953
953
954
955
955
956
957
957
958
959
959
960
961
961
962
963
963
964
965
965
966
967
967
968
969
969
970
971
971
972
973
973
974
975
975
976
977
977
978
979
979
980
981
981
982
983
983
984
985
985
986
987
987
988
989
989
990
991
991
992
993
993
994
995
995
996
997
997
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1
```

```

31         elif self.loc == "Column Flange-Beam Web" and self.
32             connection == KEY_DISP_SEATED_ANGLE:
33                 self.display.View.SetProj(OCCT.Core.V3d.V3d_XnegYnegZpos
34                     )
35
36         elif self.loc == "Column Flange-Beam Web" and self.
37             connection == KEY_DISP_SEATED_ANGLE:
38                 self.display.View.SetProj(OCCT.Core.V3d.V3d_XposYnegZpos
39                     )
40
41
42         if self.component == "Column":
43             osdag_display_shape(self.display, self.connectivityObj.
44                 get_columnModel(), update=True)
45
46         elif self.component == "Beam":
47             osdag_display_shape(self.display, self.connectivityObj.
48                 get_beamModel(), material=Graphic3d_NOM_ALUMINIUM,
49                     update=True)
50
51         elif component == "cleatAngle":
52
53             osdag_display_shape(self.display, self.connectivityObj.
54                 angleModel, color=Quantity_NOC_BLUE1, update=True)
55             osdag_display_shape(self.display, self.connectivityObj.
56                 angleLeftModel, color=Quantity_NOC_BLUE1,
57                     update=True)
58
59             nutboltlist = self.connectivityObj.nut_bolt_array.
60                 get_models()
61
62             for nutbolt in nutboltlist:
63                 osdag_display_shape(self.display, nutbolt, color=
64                     Quantity_NOC_SADDLEBROWN, update=True)
65
66
67             elif component == "SeatAngle":
68                 osdag_display_shape(self.display, self.connectivityObj.
69                     topclipangleModel, color=Quantity_NOC_BLUE1,
70                         update=True)
71
72                 osdag_display_shape(self.display, self.connectivityObj.
73                     angleModel, color=Quantity_NOC_BLUE1, update=True)
74
75             nutboltlist = self.connectivityObj.nut_bolt_array.
76                 get_models()
77
78             for nutbolt in nutboltlist:

```

```

56         osdag_display_shape(self.display, nutbolt, color=
57                         Quantity_NOC_SADDLEBROWN, update=True)
58
59     elif self.component == "Plate":
60         osdag_display_shape(self.display, self.connectivityObj.
61                             weldModelLeft, color=Quantity_NOC_RED, update=True)
62         osdag_display_shape(self.display, self.connectivityObj.
63                             weldModelRight, color=Quantity_NOC_RED, update=True)
64         osdag_display_shape(self.display, self.connectivityObj.
65                             plateModel, color=Quantity_NOC_BLUE4, update=True)
66         nutboltlist = self.connectivityObj.nut_bolt_array.
67                     get_models()
68         for nutbolt in nutboltlist:
69             osdag_display_shape(self.display, nutbolt, color=
70                         Quantity_NOC_SADDLEBROWN, update=True)
71
72     elif self.component == "Model":
73
74         osdag_display_shape(self.display, self.connectivityObj.
75                             columnModel, update=True)
76         osdag_display_shape(self.display, self.connectivityObj.
77                             beamModel, material=Graphic3d_NOM_ALUMINIUM,
78                                         update=True)
79         if self.connection == KEY_DISP_FINPLATE or self.
80             connection == KEY_DISP_ENDPLATE:
81             osdag_display_shape(self.display, self.
82                             connectivityObj.weldModelLeft, color=
83                             Quantity_NOC_RED, update=True)
84             osdag_display_shape(self.display, self.
85                             connectivityObj.weldModelRight, color=
86                             Quantity_NOC_RED, update=True)
87             osdag_display_shape(self.display, self.
88                             connectivityObj.plateModel, color=
89                             Quantity_NOC_BLUE1,
90                                         update=True)
91
92         elif self.connection == KEY_DISP_CLEATANGLE:
93             osdag_display_shape(self.display, self.
94                             connectivityObj.angleModel, color=

```

```

                                Quantity_NOC_BLUE1 ,
79                               update=True)
80
81                         osdag_display_shape(self.display, self.
82
83                               connectivityObj.angleLeftModel, color=
84
85                               Quantity_NOC_BLUE1 ,
86                               update=True)
87
88                         osdag_display_shape(self.display, self.
89
90                               connectivityObj.topclipangleModel, color=
91
92                               Quantity_NOC_BLUE1 ,
93                               update=True)
94
95
96
97
98
99
100
101
102
103
104
105
                                Quantity_NOC_BLUE1 ,
                               update=True)
                               osdag_display_shape(self.display, self.
                               connectivityObj.angleModel, color=
                               Quantity_NOC_BLUE1 ,
                               update=True)
nutboltlist = self.connectivityObj.nut_bolt_array.
                           get_models()
for nutbolt in nutboltlist:
    osdag_display_shape(self.display, nutbolt, color=
                           Quantity_NOC_SADDLEBROWN, update=True)

if self.mainmodule == "Moment Connection":
    if self.connection == KEY_DISP_BEAMCOVERPLATE:

        self.B = self.module_class()
        # else:
        #     pass
        #
        # self.loc = A.connectivity
        self.CPObj = self.createBBCoverPlateCAD()  #
                           CPBoltedObj is an object which gets all the
                           calculated values of CAD models
if self.component == "Beam":
    # Displays both beams
    osdag_display_shape(self.display, self.CPObj.
                           get_only_beams_Models(), update=True)

```

```

106
107     elif self.component == "Connector":
108         osdag_display_shape(self.display, self.CPObj.
109                             get_flangewebplatesModel(), update=True,
110                             color=Quantity_NOC_BLUE1)
111         if self.B.preference != 'Outside':
112             osdag_display_shape(self.display, self.CPObj.
113                             get_innetplatesModels(), update=True,
114                             color=Quantity_NOC_BLUE1)
115
116
117         osdag_display_shape(self.display, self.CPObj.
118                             get_nut_bolt_arrayModels(), update=True,
119                             color=Quantity_NOC_YELLOW)
120
121
122         # Todo: remove velove commented lines
123
124
125         if self.B.preference != 'Outside':
126             osdag_display_shape(self.display, self.CPObj.
127                             get_innetplatesModels(), update=True,
128                             color=Quantity_NOC_BLUE1)
129
130
131         osdag_display_shape(self.display, self.CPObj.
132                             get_nut_bolt_arrayModels(), update=True,
133                             color=Quantity_NOC_YELLOW)
134
135         elif self.connection == KEY_DISP_BB_EP_SPLICE:
136             self.B = self.module_class()
137
138             self.ExtObj = self.createBBEndPlateCAD()
139
140
141             if component == "Beam":
142                 osdag_display_shape(self.display, self.ExtObj.
143                                 get_beam_models(), update=True)

```

```

137
138     elif component == "Connector":
139         osdag_display_shape(self.display, self.ExtObj.
140                             get_plate_connector_models(), update=True,
141                             color='Blue')
142         osdag_display_shape(self.display, self.ExtObj.
143                             get_welded_models(), update=True, color='Red')
144         osdag_display_shape(self.display, self.ExtObj.
145                             get_nut_bolt_array_models(), update=True,
146                             color=Quantity_NOC_SADDLEBROWN)
147
148     elif component == "Model":
149
150         # osdag_display_shape(self.display, self.ExtObj.
151         #                     get_models(), update=True)
152         osdag_display_shape(self.display, self.ExtObj.
153                             get_beam_models(), update=True)
154         osdag_display_shape(self.display, self.ExtObj.
155                             get_plate_connector_models(), update=True,
156                             color='Blue')
157         osdag_display_shape(self.display, self.ExtObj.
158                             get_welded_models(), update=True, color='Red')
159         osdag_display_shape(self.display, self.ExtObj.
160                             get_nut_bolt_array_models(), update=True,
161                             color=Quantity_NOC_SADDLEBROWN)
162
163
164     if self.connection == KEY_DISP_BEAMCOVERPLATEWELD:
165
166         self.B = self.module_class()
167         self.CPObj = self.createBBCoverPlateCAD()
168         beams = self.CPObj.get_beam_models()
169         plates = self.CPObj.get_plate_models()
170         welds = self.CPObj.get_welded_modules()
171
172
173         if self.component == "Beam":
174
175             # Displays both beams
176             osdag_display_shape(self.display, beams, update=
177                                 True)

```

```

167     elif self.component == "Connector":
168         osdag_display_shape(self.display, plates, update=
169                         True, color=Quantity_NOC_BLUE1)
170         osdag_display_shape(self.display, welds, update=
171                         True, color=Quantity_NOC_RED)
172     elif self.component == "Model":
173         osdag_display_shape(self.display, beams, update=
174                         True)
175         osdag_display_shape(self.display, plates, update=
176                         True, color=Quantity_NOC_BLUE1)
177         osdag_display_shape(self.display, welds, update=
178                         True, color=Quantity_NOC_RED)
179
180
181     elif self.connection == KEY_DISP_COLUMNCOVERPLATE:
182         self.C = self.module_class()
183         self.CPObj = self.createCCCoverPlateCAD()
184         columns = self.CPObj.get_column_models()
185         plates = self.CPObj.get_plate_models()
186         nutbolt = self.CPObj.get_nut_bolt_models()
187         onlycolumn = self.CPObj.get_only_column_models()
188
189
190     if self.component == "Column":
191         # Displays both beams
192         osdag_display_shape(self.display, onlycolumn,
193                         update=True)
194     elif self.component == "Cover Plate":
195         osdag_display_shape(self.display, plates, update=
196                         True, color=Quantity_NOC_BLUE1)
197         osdag_display_shape(self.display, nutbolt, update=
198                         True, color=Quantity_NOC_YELLOW)
199     elif self.component == "Model":
200         osdag_display_shape(self.display, columns, update=
201                         True)
202         osdag_display_shape(self.display, plates, update=
203                         True, color=Quantity_NOC_BLUE1)
204         osdag_display_shape(self.display, nutbolt, update=
205                         True, color=Quantity_NOC_YELLOW)
206
207
208
209
210
211
212
213
214

```

```

195     elif self.connection == KEY_DISP_BCENDPLATE:
196         self.Bc = self.module_class()
197         self.ExtObj = self.createBCEndPlateCAD()
198
199         self.display.View.SetProj(OCC.Core.V3d.V3d_XnegYnegZpos
200             )
201         c_length = self.column_length
202         # Point1 = gp_Pnt(0.0, 0.0, c_length)
203         # DisplayMsg(self.display, Point1, self.Bc.
204             supporting_section.designation)
205         b_length = self.beam_length + self.Bc.
206             supporting_section.depth/2+100
207         # Point2 = gp_Pnt(0.0,-b_length, c_length/2)
208         # DisplayMsg(self.display, Point2, self.Bc.
209             supported_section.designation)
210         # Displays the beams #TODO ANAND
211         if component == "Column":
212             self.display.View_Iso()
213             osdag_display_shape(self.display, self.ExtObj.
214                 columnModel, update=True)
215             # Point1 = gp_Pnt(-self.Bc.supporting_section.
216                 flange_width/2, 0, c_length)
217             # DisplayMsg(self.display, Point1, self.Bc.
218                 supporting_section.designation)
219             # Point = gp_Pnt(0.0, 0.0, 10)
220             # DisplayMsg(self.display, Point, "Column")
221
222             elif component == "Beam":
223                 self.display.View_Iso()
224                 osdag_display_shape(self.display, self.ExtObj.
225                     beamModel, update=True,
226                         material=
227                             Graphic3d_NOM_ALUMINIUM)
228                 # Point2 = gp_Pnt(0.0, -b_length, c_length / 2)
229                 # DisplayMsg(self.display, Point2, self.Bc.
230                     supported_section.designation)
231                 # , color = 'Dark Gray'
232
233             elif component == "Connector":

```

```

224     osdag_display_shape(self.display, self.ExtObj.
225         get_plate_connector_models(), update=True,
226             color='Blue')
227     osdag_display_shape(self.display, self.ExtObj.
228         get_welded_models(), update=True, color='Red')
229     osdag_display_shape(self.display, self.ExtObj.
230         get_nut_bolt_array_models(), update=True,
231             color=Quantity_NOC_SADDLEBROWN)
232
233     elif component == "Model":
234
235         osdag_display_shape(self.display, self.ExtObj.
236             get_column_models(), update=True)
237         osdag_display_shape(self.display, self.ExtObj.
238             get_beam_models(), update=True,
239                 material=
240                     Graphic3d_NOM_ALUMINIUM)
241
242         osdag_display_shape(self.display, self.ExtObj.
243             get_plate_connector_models(), update=True,
244                 color='Blue')
245
246         osdag_display_shape(self.display, self.ExtObj.
247             get_welded_models(), update=True, color='Red')
248
249         osdag_display_shape(self.display, self.ExtObj.
250             get_nut_bolt_array_models(), update=True,
251                 color=Quantity_NOC_SADDLEBROWN)
252
253         # Point1 = gp_Pnt(self.Bc.supporting_section.
254             flange_width/2, -self.Bc.supporting_section.
255                 depth/2, c_length*0.75)
256
257         # DisplayMsg(self.display, Point1, self.Bc.
258             supporting_section.designation)
259
260         # Point2 = gp_Pnt(self.Bc.supporting_section.
261             flange_width/2, -b_length, c_length / 2)
262
263         # DisplayMsg(self.display, Point2, self.Bc.
264             supported_section.designation)
265
266         # Erase(DisplayMsg(self.display, Point2, self.Bc.
267             supported_section.designation))
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
759

```

```

248     elif self.connection == KEY_DISP_COLUMNCOVERPLATEWELD:
249         self.C = self.module_class()
250         self.CPObj = self.createCCCoverPlateCAD()
251         columns = self.CPObj.get_column_models()
252         plates = self.CPObj.get_plate_models()
253         welds = self.CPObj.get_welded_modules()
254
255         if self.component == "Column":
256             # Displays both beams
257             osdag_display_shape(self.display, columns, update=
258                                 True)
259         elif self.component == "Cover Plate":
260             osdag_display_shape(self.display, plates, update=
261                                 True, color=Quantity_NOC_BLUE1)
262             osdag_display_shape(self.display, welds, update=
263                                 True, color=Quantity_NOC_RED)
264         elif self.component == "Model":
265             osdag_display_shape(self.display, columns, update=
266                                 True)
267             osdag_display_shape(self.display, plates, update=
268                                 True, color=Quantity_NOC_BLUE1)
269             osdag_display_shape(self.display, welds, update=
270                                 True, color=Quantity_NOC_RED)
271
272         elif self.connection == KEY_DISP_COLUMNENDPLATE:
273             self.CEP = self.module_class()
274             self.CEPObj = self.createCCEndPlateCAD()
275             columns = self.CEPObj.get_column_models()
276             plates = self.CEPObj.get_plate_models()
277             welds = self.CEPObj.get_weld_models()
278             nutBolts = self.CEPObj.get_nut_bolt_models()
279
280             if self.component == "Column":
281                 osdag_display_shape(self.display, columns, update=
282                                     True)
283
284             elif self.component == "Connector":
285                 osdag_display_shape(self.display, plates, update=
286                                     True, color=Quantity_NOC_BLUE1)

```

```

279         osdag_display_shape(self.display, welds, update=
280                         True, color=Quantity_NOC_RED)
281         osdag_display_shape(self.display, nutBolts, update=
282                         True, color=Quantity_NOC_YELLOW)
283
284     elif self.component == "Model":
285         osdag_display_shape(self.display, columns, update=
286                         True)
287         osdag_display_shape(self.display, plates, update=
288                         True, color=Quantity_NOC_BLUE1)
289         osdag_display_shape(self.display, welds, update=
290                         True, color=Quantity_NOC_RED)
291         osdag_display_shape(self.display, nutBolts, update=
292                         True, color=Quantity_NOC_YELLOW)
293
294     elif self.connection == KEY_DISP_BASE_PLATE:
295         self.Bp = self.module_class
296
297         self.BPObj = self.createBasePlateCAD()
298
299
300         column = self.BPObj.get_column_model()
301         plate = self.BPObj.get_plate_connector_models()
302         weld = self.BPObj.get_welded_models()
303         nut_bolt = self.BPObj.get_nut_bolt_array_models()
304         conc = self.BPObj.get_concrete_models()
305         grout = self.BPObj.get_grout_models()
306
307
308     if self.component == "Model": # Todo: change this into
309         key
310         osdag_display_shape(self.display, column, update=
311                         True)
312         osdag_display_shape(self.display, plate, color=
313                         Quantity_NOC_BLUE1, update=True)
314         osdag_display_shape(self.display, weld, color=
315                         Quantity_NOC_RED, update=True)
316         osdag_display_shape(self.display, nut_bolt, color=
317                         Quantity_NOC_YELLOW, update=True)
318         osdag_display_shape(self.display, conc, color=GRAY,
319                         transparency=0.5, update=True)

```

```

306         osdag_display_shape(self.display, grout, color=GRAY
307                         , transparency=0.5, update=True)
308
309     elif self.component == "Column":
310         osdag_display_shape(self.display, column, update=
311                             True)
312
313     elif self.component == "Connector":
314         osdag_display_shape(self.display, plate, color=
315                             Quantity_NOC_BLUE1, update=True)
316         osdag_display_shape(self.display, weld, color=
317                             Quantity_NOC_RED, update=True)
318         osdag_display_shape(self.display, nut_bolt, color=
319                             Quantity_NOC_YELLOW, update=True)
320
321     elif self.mainmodule == 'Columns with known support conditions':
322         :
323         self.col = self.module_class()
324         self.ColObj = self.createColumnInFrameCAD()
325
326         if self.component == "Model":
327             osdag_display_shape(self.display, self.ColObj, update=
328                                 True)
329
330         elif self.mainmodule == 'Lap Joint Bolted Connection':
331             self.col = self.module_class()
332             self.assembly, self.plate1_model, self.plate2_model, self.
333                 bolt_models, self.nuts_models = self.createBoltedLapJoint
334                 ()
335
336         if self.component == "Model":
337             osdag_display_shape(self.display, self.plate1_model,
338                                 update=True, material=Graphic3d_NOM_ALUMINIUM)
339             osdag_display_shape(self.display, self.plate2_model,
340                                 update=True)
341             if hasattr(self, 'bolt_models'):
342                 for bolt in self.bolt_models:
343                     osdag_display_shape(self.display, bolt, update=
344                                         True, color=Quantity_NOC_SADDLEBROWN)

```

```

333
334             if hasattr(self, 'nuts_models'):
335                 for nut in self.nuts_models:
336                     osdag_display_shape(self.display, nut,
337                                         update=True, color=
338                                         Quantity_NOC_SADDLEBROWN)
339
340         elif self.mainmodule == 'Butt Joint Bolted Connection':
341             self.col = self.module_class()
342             self.assembly, self.plate1_model, self.plate2_model, self.
343             platec_model, self.bolt_models, self.nuts_models = self.
344             createButtJointBoltedCAD()
345
346             if self.component == "Model":
347                 osdag_display_shape(self.display, self.plate1_model,
348                                     update=True, material=Graphic3d_NOM_ALUMINIUM)
349                 osdag_display_shape(self.display, self.plate2_model,
350                                     update=True)
351                 osdag_display_shape(self.display, self.platec_model,
352                                     update=True)
353
354             if hasattr(self, 'bolt_models'):
355                 for bolt in self.bolt_models:
356                     osdag_display_shape(self.display, bolt, update=
357                                         True, color=Quantity_NOC_SADDLEBROWN)
358
359             if hasattr(self, 'nuts_models'):
360                 for nut in self.nuts_models:
361                     osdag_display_shape(self.display, nut,
362                                         update=True, color=
363                                         Quantity_NOC_SADDLEBROWN)
364
365         elif self.mainmodule == 'Butt Joint Welded Connection':
366             self.col = self.module_class()
367             self.plate1_model, self.plate2_model, self.
368             cover_plate_model, self.welds_models = self.
369             createWeldedButtJoint()
370
371             if self.component == "Model":

```

```

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

```

```

390     if self.connection == KEY_DISP_TENSION_BOLTED:
391         self.T = self.module_class()
392         self.TObj = self.createTensionCAD()
393
394         member = self.TObj.get_members_models()
395         plate = self.TObj.get_plates_models()
396
397         nutbolt = self.TObj.get_nut_bolt_array_models()
398
399         onlymember = self.TObj.get_only_members_models()
400         # distance = self.T.length/2 - (2* self.T.plate.
401         # end_dist_provided + (self.T.plate.bolt_line - 1 ) *
402         # self.T.plate.pitch_provided)
403
404         # Point = gp_Pnt(distance, 0.0, 300)
405         # DisplayMsg(self.display, Point, self.T.section_size_1
406         # .designation)
407
408
409         if self.component == "Member": # Todo: change this
410             into key
411             osdag_display_shape(self.display, onlymember,
412                                 update=True)
413             elif self.component == "Plate":
414                 osdag_display_shape(self.display, plate, color=
415                             Quantity_NOC_BLUE1, update=True)
416                 osdag_display_shape(self.display, nutbolt, color=
417                             Quantity_NOC_YELLOW, update=True)
418             elif self.component == "Endplate":
419                 endplate = self.TObj.get_end_plates_models()
420                 end_nutbolt = self.TObj.
421                     get_end_nut_bolt_array_models()
422                 osdag_display_shape(self.display, endplate, color=
423                             Quantity_NOC_BLUE1, update=True)
424                 osdag_display_shape(self.display, end_nutbolt,
425                             color=Quantity_NOC_YELLOW, update=True)
426             else:
427                 connector = BRepAlgoAPI_Fuse(nutbolt, plate).Shape
428                     ()
429                 shape = BRepAlgoAPI_Fuse(connector, member).Shape()

```

```

418         self.TObj.shape = shape
419         osdag_display_shape(self.display, member, update=
420                             True)
421         osdag_display_shape(self.display, plate, color=
422                             Quantity_NOC_BLUE1, update=True)
423         osdag_display_shape(self.display, nutbolt, color=
424                             Quantity_NOC_YELLOW, update=True)
425
426
427
428     elif self.connection == KEY_DISP_TENSION_WELDED:
429
430         self.T = self.module_class()
431
432         self.TObj = self.createTensionCAD()
433
434
435         member = self.TObj.get_members_models()
436         plate = self.TObj.get_plates_models()
437         welds = self.TObj.get_welded_models()
438
439         if self.component == "Member": # Todo: change this
440             into key
441
442             osdag_display_shape(self.display, member, update=
443                                 True)
444
445         elif self.component == "Plate":
446
447             osdag_display_shape(self.display, plate, color=
448                             Quantity_NOC_BLUE1, update=True)
449
450             osdag_display_shape(self.display, welds, color=
451                             Quantity_NOC_RED, update=True)
452
453         elif self.component == "Endplate":
454
455             endplate = self.TObj.get_end_plates_models()
456
457             osdag_display_shape(self.display, endplate, color=
458                             Quantity_NOC_BLUE1, update=True)
459
460         else:
461
462             connector = BRepAlgoAPI_Fuse(welds, plate).Shape()
463
464             shape = BRepAlgoAPI_Fuse(connector, member).Shape()
465
466             self.TObj.shape = shape
467
468             osdag_display_shape(self.display, member, update=
469                                 True)
470
471             osdag_display_shape(self.display, plate, color=
472                             Quantity_NOC_BLUE1, update=True)

```

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

```

1
2 (Line 1843- 1889)
3     def return_class(self ,name):
4
5         if name == KEY_DISP_FINPLATE:
6             return FinPlateConnection
7
8         elif name == KEY_DISP_ENDPLATE:
9
10        elif name == KEY_DISP_BUTTJOINTBOLTED:
11            return ButtJointBolted
12
13        elif name == KEY_DISP_BUTTJOINTWELDED:
14            return ButtJointWelded
15
16 \subsection{Output of the CAD Integration:}

```

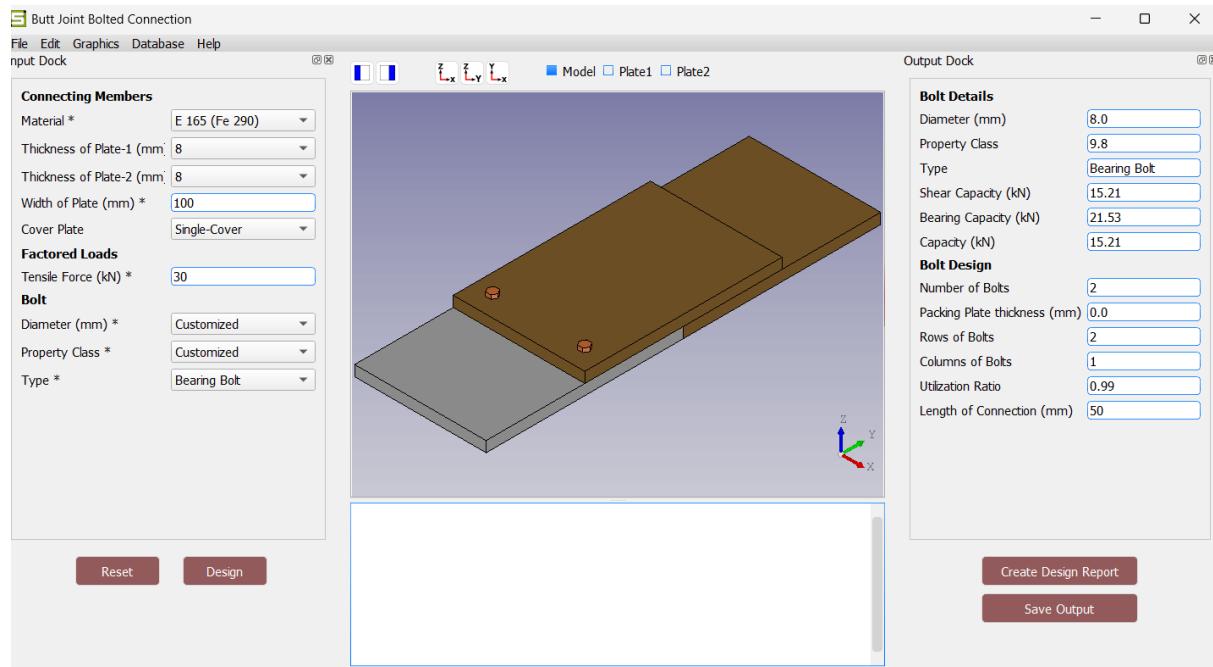


Figure 6.1: CAD Integration of Bolted Butt Joint Connection

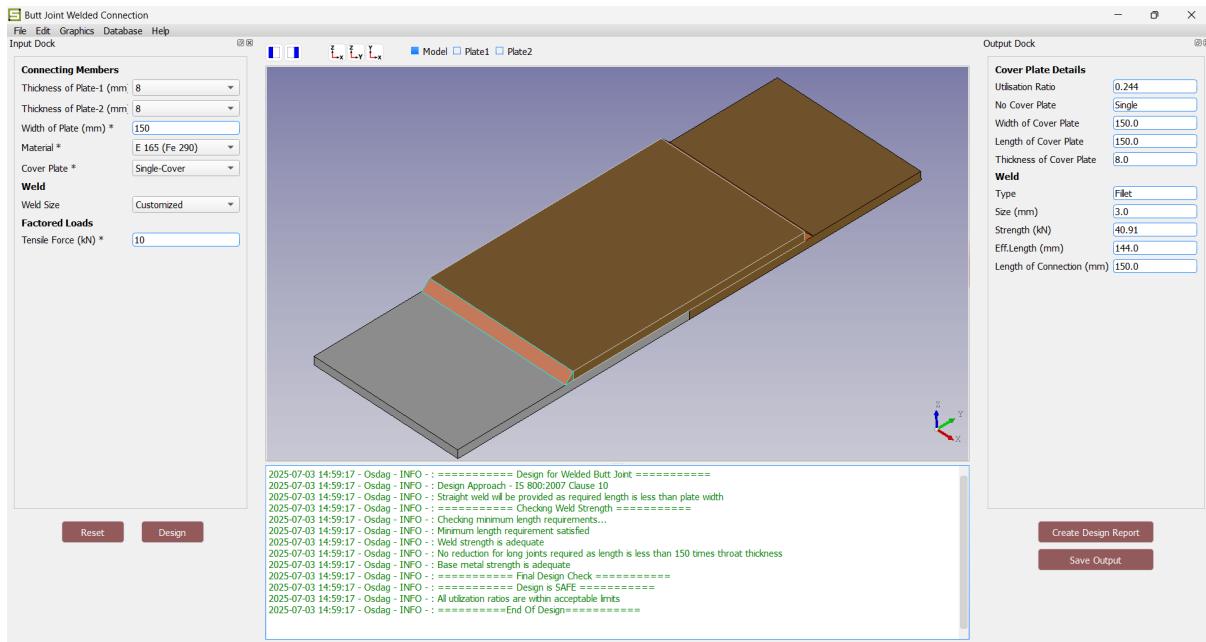


Figure 6.2: CAD Integration of Welded Butt Joint Connection

Chapter 7

Gantry Girder

7.1 Problem Statement

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

7.2 Task Done

Gantry Girder

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

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

7.3 Python Code

7.3.1 Description of the Script

The script is structured as follows:

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

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

- **Geometry Construction:** The script generates the I-section using a custom `create_i_section` function and the C-section using `create_c_section`. The C-section is then rotated 90° about the X-axis to simulate its vertical orientation in real structures. A translation transformation is applied to correctly position the C-section on top of the I-section.
- **Output:** The final output is a 3D CAD model of a gantry girder visualized using the Open Cascade (OCC) viewer. It includes the base I-section and a vertically oriented C-section mounted above it. The components are colored with different materials (aluminum and copper) and a gradient background enhances visual clarity.

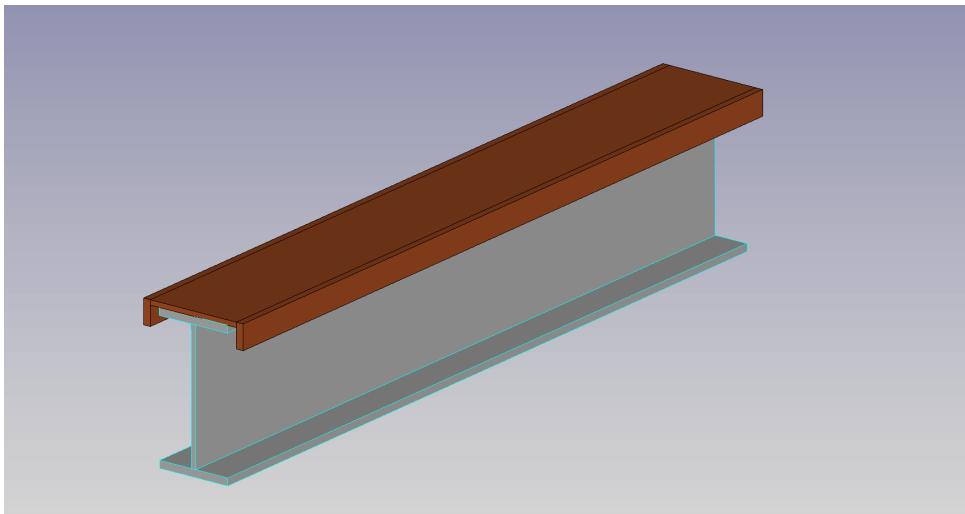


Figure 7.1: Gantry Girder Beam

7.3.2 Python Script

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

Listing 7.1: Gantry Girder Beam

```
1 -----begin code-----
2
3 from OCC.Core.gp import gp_Pnt, gp_Dir, gp_Vec, gp_Trsf, gp_Ax1
4 from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakeBox
5 from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Fuse
6 from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_Transform
```

```

7  from OCC.Display.SimpleGui import init_display
8  from OCC.Core.Quantity import Quantity_Color, Quantity_TOC_RGB
9  from OCC.Core.Quantity import Quantity_Color, Quantity_TOC_RGB
10 from OCC.Core.Aspect import Aspect_GFM_VER # Import the correct
     gradient mode
11 from draw_i_section import create_i_section
12 from draw_c_section import create_c_section
13
14 def create_gantry_girder(length,
15                         i_flange_thickness, i_web_thickness, i_width,
16                         i_depth,
17                         c_flange_thickness, c_web_thickness, c_width,
18                         c_depth):
19
20     # Create I-section
21     i_section = create_i_section(length, i_width, i_depth,
22                                   i_flange_thickness, i_web_thickness)
23
24     # Create C-section
25     c_section = create_c_section(length, c_width, c_depth,
26                                   c_flange_thickness, c_web_thickness)
27
28     # Rotate the C-section 90 degrees about its length (X-axis)
29     rotation_axis = gp_Ax1(gp_Pnt(0, 0, 0), gp_Dir(1, 0, 0)) #
          Corrected: using gp_Pnt and gp_Dir
30     rotation = gp_Trsf()
31     rotation.SetRotation(rotation_axis, 3.14159/2) # 90 degrees in
          radians (pi/2)
32     rotated_c_section = BRepBuilderAPI_Transform(c_section, rotation,
33                                                 True).Shape()
34
35     # Position the rotated C-section on top of the I-section
36
37     trsf = gp_Trsf()
38     trsf.SetTranslation(gp_Vec(0, i_width/2+c_depth/2, (i_depth-
          c_width+c_web_thickness)))
39     transformed_c_section = BRepBuilderAPI_Transform(rotated_c_section,
40                                                 trsf, True).Shape()

```

```

36     # Combine the sections
37     girder = BRepAlgoAPI_Fuse(i_section, transformed_c_section).Shape()
38
39     return girder
40
41 if __name__ == "__main__":
42     length = 1500.0
43
44     # I-section parameters
45     i_width = 150.0
46     i_depth = 300.0
47     i_flange_thickness = 15.0
48     i_web_thickness = 10.0
49
50     # C-section parameters
51     c_width = 50.0
52     c_depth = 220.0
53     c_flange_thickness = 15.0
54     c_web_thickness = 10.0
55
56     gantry_girder = create_gantry_girder(length,
57                                         i_flange_thickness,
58                                         i_web_thickness, i_width,
59                                         i_depth,
60                                         c_flange_thickness,
61                                         c_web_thickness, c_width,
62                                         c_depth)
63
63     # Visualization
64     display, start_display, add_menu, add_function_to_menu =
65         init_display()
66
66     top_color = Quantity_Color(0.27, 0.27, 0.44, Quantity_TOC_RGB)    #
67         Dark blue (#444470)
68     bottom_color = Quantity_Color(0.67, 0.67, 0.67, Quantity_TOC_RGB)
69         # Light gray (#AAAAAA)
70
71     # Set the gradient background (top-to-bottom)
72     display.View.SetBgGradientColors(top_color, bottom_color,
73                                     Aspect_GFM_VER, True)

```

```

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

```

7.3.3 Explanation of the Code

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

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

Chapter 8

Castellated Beam

8.1 Problem Statement

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

8.2 Task Done

Castellated Beam

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

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

8.3 Python Code

8.3.1 Description of the Script

The script is structured as follows:

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

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

- **I-Section and Notch Creation:** The `Notch` object is instantiated with notch parameters, and an `ISection` object is created using all geometric inputs. The I-section beam model is generated through methods such as `place`, `compute_params`, and `create_model` to obtain a 3D prism representing the beam with notches.
- **Transformation:** A translation transformation is applied to the I-section prism to shift it by +20 units in the X-direction and +25 units in the Z-direction, positioning the beam appropriately in the coordinate space.
- **Hexagonal Pattern Parameters:** Variables like `a`, `b`, `c`, `e`, and `j` control the geometry and spacing of the hexagonal openings cut into the beam web to create the castellated effect. The horizontal (`hex_v`) and vertical (`hex_h`) dimensions of the hexagon are computed based on beam and pattern parameters.
- **Hexagonal Prisms Creation:** A loop calculates the number of hexagonal openings (`n`) along the beam length, then generates hexagonal wires and extrudes them into prisms at calculated positions along the beam.
- **Extrude Cutting:** Each hexagonal prism is subtracted from the translated I-section prism using Boolean cut operations, creating the characteristic castellated web openings.
- **Visualization:** The final castellated beam shape is displayed using the OCC visualization module, fitting the view and starting the GUI event loop for interactive inspection.

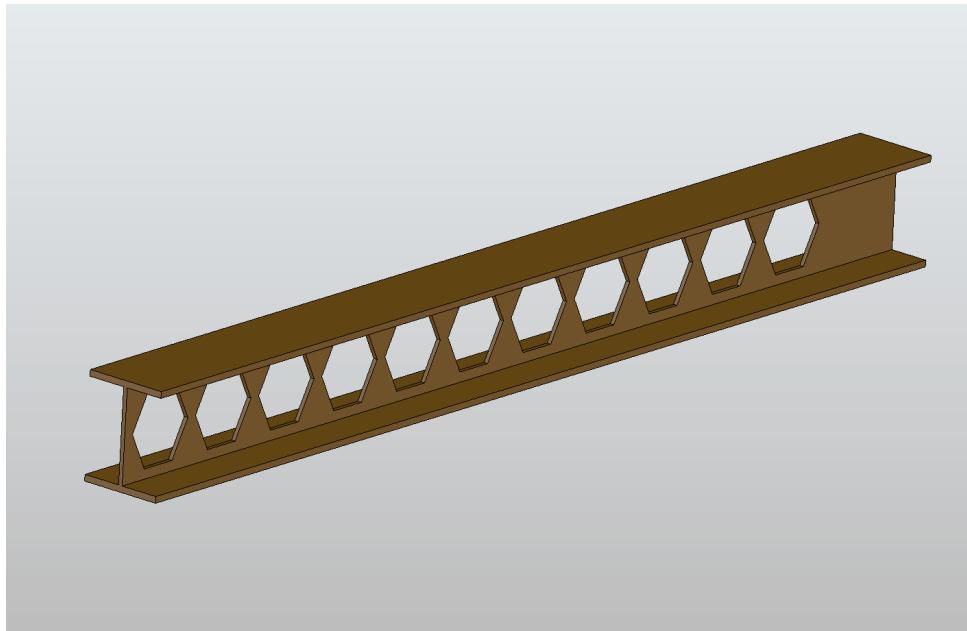


Figure 8.1: Castellated Beam

8.3.2 Python Script

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

Listing 8.1: Hexagonal Prism For extrude cut on final code

```
1 %-----begin code-----
2
3 import math
4 from OCC.Core.gp import gp_Pnt, gp_Vec
5 from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_MakePolygon
6 from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_MakeFace
7 from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakePrism
8 from OCC.Core.gp import gp_Pnt
9
10
11 def hexagon(center, hex_v, hex_h):
12     polygon = BRepBuilderAPI_MakePolygon()
13     for i in range(6):
14         angle = math.radians(i * 60)
15         y = center.Y() + (hex_h / 2) * math.cos(angle)
16         z = center.Z() + (hex_v / 2) * math.sin(angle)
17         polygon.Add(gp_Pnt(center.X(), y, z)) # Keep X constant
18     polygon.Close()
```

```

19     return polygon.Wire()
20
21 if __name__ == '__main__':
22     from OCC.Display.SimpleGui import init_display
23     display, start_display, add_menu, add_function_to_menu =
24         init_display()
25
26     center = gp_Pnt(0, 0, 0)
27     hex_w = 5
28     hex_h = 7
29     hex_wire = hexagon(center, hex_w, hex_h)
30
31     # Create hexagon face
32     hex_face = BRepBuilderAPI_MakeFace(hex_wire).Face()
33
34     # Extrude the face along X-axis
35     length = 10
36     extrude_vec = gp_Vec(length, 0, 0)
37     hex_prism = BRepPrimAPI_MakePrism(hex_face, extrude_vec).Shape()
38
39     display.DisplayShape(hex_prism, update=True)
40     start_display()

```

Listing 8.2: Castillated Beam

```

1 -----begin code-----
2
3 import math
4 import numpy
5 from OCC.Core.gp import gp_Pnt, gp_Vec, gp_Trsf
6 from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_Transform
7 from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_MakePolygon
8 from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_MakeFace
9 from OCC.Core.BRepPrimAPI import BRepPrimAPI_MakePrism
10 from OCC.Display.SimpleGui import init_display
11 from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_MakeFace
12 from OCC.Core.BRepAlgoAPI import BRepAlgoAPI_Cut
13 from Hexagon_beam2 import hexagon
14 from ISection import ISection
15 from notch import Notch

```

```

16 | from OCC.Display.SimpleGui import init_display
17 | display, start_display, add_menu, add_function_to_menu = init_display()
18 |
19 |
20 | # Creating I section
21 | B = 40 # width of flanges
22 | T = 3 # flanges thickness
23 | D = 50 # height of beam ( flange thickness +web height)
24 | t = 2 # web thickness
25 | R1 = 5 # notch radius
26 | R2 = 5
27 | alpha = 1
28 | length = 500 # length of beam
29 | width = 10 # notch
30 | hight = 10 # notch
31 | notchObj = Notch(R1, hight, width, length)
32 |
33 | origin = numpy.array([0.,0.,0.])
34 | uDir = numpy.array([1.,0.,0.])
35 | shaftDir = numpy.array([0.,1.,0.])
36 |
37 | ISec = ISection(B, T, D, t, R1, R2, alpha, length, notchObj)
38 | place = ISec.place(origin, uDir, shaftDir)
39 | point = ISec.compute_params()
40 | prism = ISec.create_model()
41 |
42 |
43 | # Define the translation vector (shift by +20 in X and +25 in Z) to
44 | # keep it in origin
45 | translation_vector = gp_Vec(20, 0, 25)
46 |
47 | # Create a transformation
48 | trsf = gp_Trsf()
49 | trsf.SetTranslation(translation_vector)
50 |
51 | # Apply the transformation to the prism
52 | translated_prism = BRepBuilderAPI_Transform(prism, trsf, True).Shape()
53 |

```

```

54 #display.DisplayShape(translated_prism, update=True)
55
56
57 #creating hexagon pattern:
58
59 a=5 # edge gap
60 b=30 # length of flat horizontal edge
61 c=5 # gap between two hexagon
62 e=3 # distance between flat edge and middle corner
63 j=5 # gap between flast edge and flange of i section
64 hex_v = D-2*j #horizontal length of hexagon
65 hex_h = 2*e+b #vertical length of hexagon
66 tw = 30 # extrtrude cut length
67
68 n=round((length-2*b+c)/(2*e+b+c))
69
70 # List to hold all hexagonal prisms
71 hex_prisms = []
72 # Create 10 hexagons along X-axis
73 for i in range(n):
74     center = gp_Pnt(0,hex_h/2+a + i * (hex_h+c), D/2) # shift X each
75         time
76     hex_wire = hexagon(center, hex_v, hex_h)
77     hex_face = BRepBuilderAPI_MakeFace(hex_wire).Face()
78     extrude_vec = gp_Vec(tw, 0, 0)
79     hex_prism = BRepPrimAPI_MakePrism(hex_face, extrude_vec).Shape()
80     hex_prisms.append(hex_prism)
81
82 # Display all prisms
83 #for prism in hex_prisms:
84 #    display.DisplayShape(prism, update=False)
85
86 #extrude cutting
87 beam=translated_prism
88 for cutter in hex_prisms:
89     beam = BRepAlgoAPI_Cut(beam, cutter).Shape()
90
91 display.DisplayShape(beam, update=True)

```

```

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

```

8.3.3 Explanation of the Code

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

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

Chapter 9

Conclusions

9.1 Tasks Accomplished

CAD Generation Scripts Developed

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

1. Butt Joint Bolted CAD Generation Script

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

2. Welded Butt Joint CAD Generation Script

Created a script that generates a fully parametric welded butt joint model with two plates and weld geometry using the Open Cascade framework.

3. CAD Integration of Both Bolted and Welded Butt Joint Connections in Osdag

Successfully integrated both CAD scripts into the Osdag application to allow real-time 3D visualization of either connection based on user selection.

4. Double Angle Gusset Plate Connection

Implemented the CAD model for a double-angle gusset plate connection, including angle section placement and weld modeling at each interface.

5. Gantry Girder Beam CAD Generation Script

Developed a 3D model combining an I-section and a rotated C-section to represent a gantry girder beam system with accurate geometric alignment and placement.

6. Castellated Beam CAD Generation Script

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

9.2 Skills Developed

Skills Gained During Internship

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

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

Chapter A

Appendix

A.1 Work Reports

Internship Work Report

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

Internship Work Report

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

Internship Work Report

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

Internship Work Report

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

Internship Work Report

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

Internship Work Report

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

Bibliography

- [1] Siddhartha Ghosh, Danish Ansari, Ajmal Babu Mahasrankintakam, Dharma Teja Nuli, Reshma Konjari, M. Swathi, and Subhrajit Dutta. Osdag: A Software for Structural Steel Design Using IS 800:2007. In Sondipon Adhikari, Anjan Dutta, and Satyabrata Choudhury, editors, *Advances in Structural Technologies*, volume 81 of *Lecture Notes in Civil Engineering*, pages 219–231, Singapore, 2021. Springer Singapore.
- [2] FOSSEE Project. FOSSEE News - January 2018, vol 1 issue 3. Accessed: 2024-12-05.
- [3] FOSSEE Project. Osdag website. Accessed: 2024-12-05.