



FOSSEE Winter Internship Report

On

Interactive 3D Plotting of Shear Force and Bending
Moment and UI Creation of Structural Steel and
Deck Design (pyside6)

Submitted by

John Doe

3rd Year B.Tech Student, Department of XYZ

ABC College of Engineering

Mumbai

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

March 20, 2026

Acknowledgments

I would like to express my sincere gratitude to everyone who supported and guided me throughout the completion of this project. I am especially thankful to the Osdag team, including Ajmal Babu M. S., Ajinkya Dahale, and Parth Karia, for their continuous support, valuable guidance, and encouragement during the course of this work.

I would like to extend my heartfelt thanks to Prof. Siddhartha Ghosh, Principal Investigator (PI), Department of Civil Engineering, IIT Bombay, for his insightful direction and motivation. I am also deeply grateful to Prof. Kannan M. Moudgalya, FOSSEE Project Investigator, Department of Chemical Engineering, IIT Bombay, for providing this opportunity and for his constant guidance. I sincerely acknowledge the contributions of FOSSEE managers Usha Viswanathan and Vinceta Parmar, along with their entire team, for their consistent support.

I would also like to acknowledge the support of the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India, for facilitating this project. I extend my appreciation to my colleagues who worked alongside me during this project for their cooperation and teamwork.

Finally, I would like to express my heartfelt gratitude to my college, department, and faculty members for their continuous encouragement and support throughout my academic journey.

Contents

1	Introduction	4
1.1	National Mission in Education through ICT	4
1.1.1	ICT Initiatives of MoE	5
1.2	FOSSEE Project	6
1.2.1	Projects and Activities	6
1.2.2	Fellowships	6
1.3	Osdag Software	7
1.3.1	Osdag GUI	8
1.3.2	Features	8
2	Screening Task	9
2.1	Problem Statement	9
2.2	Tasks Done	9
3	Internship Task 1 Title	10
3.1	Task 1: Problem Statement	10
3.2	Task 1: Tasks Done	10
3.3	Task 1: Python Code	13
3.3.1	Description of the Script	13
3.3.2	Python Code	14
3.3.3	Explanation of the Code	41
3.4	Task 1: Documentation	42
3.4.1	Directory Structure (src/osdagbridge/core/bridge _{types} /plate _{girder})	42
3.4.2	Program Execution	43
3.4.3	Workflow Description	43
4	Steel Design and Deck Design Output Dialog Implementation	44
4.1	Task 2: Problem Statement	44
4.2	Task 2: Tasks Done	44
4.3	Task 2: Documentation	46
4.3.1	File Structure	46

4.3.2	Data Flow	47
4.3.3	Key Design Decisions	47
5	Conclusions	48
5.1	Tasks Accomplished	48
5.2	Skills Developed	48
A	Appendix	50
A.1	Work Reports	50
	Bibliography	54

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	SWAYAMPBABHA	Access 24x7 TV programs	Enable SWAYAMPBABHA 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.

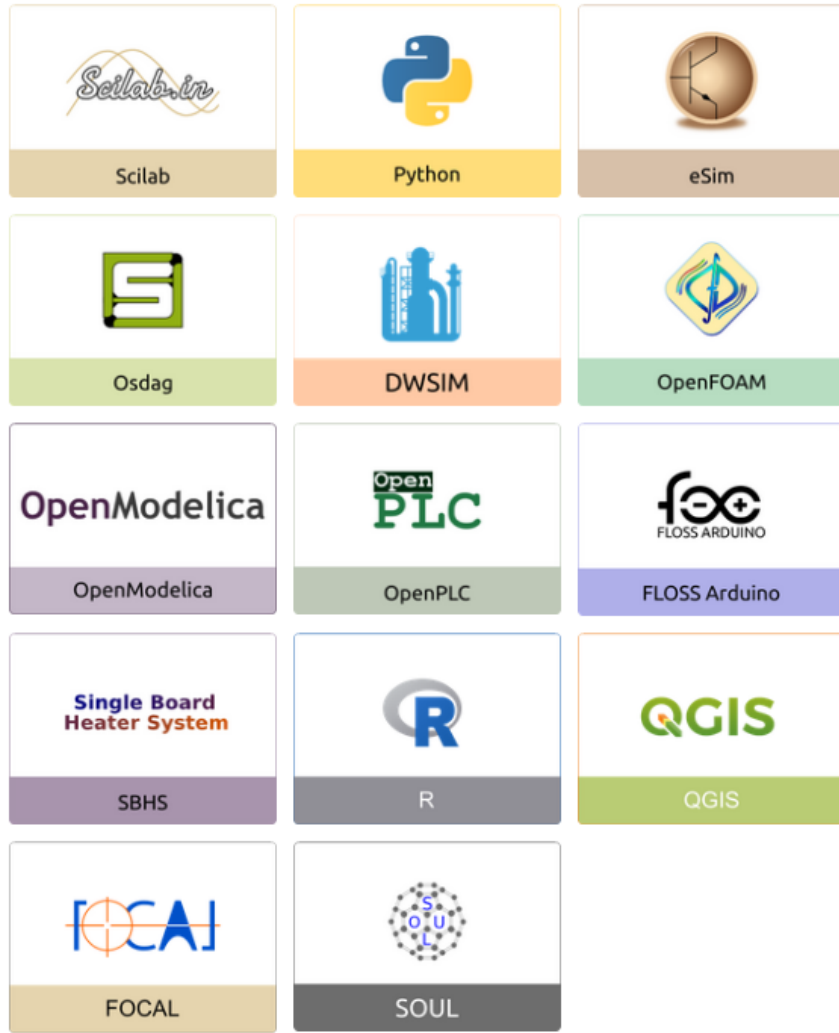


Figure 1.1: FOSSEE Projects and Activities

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.

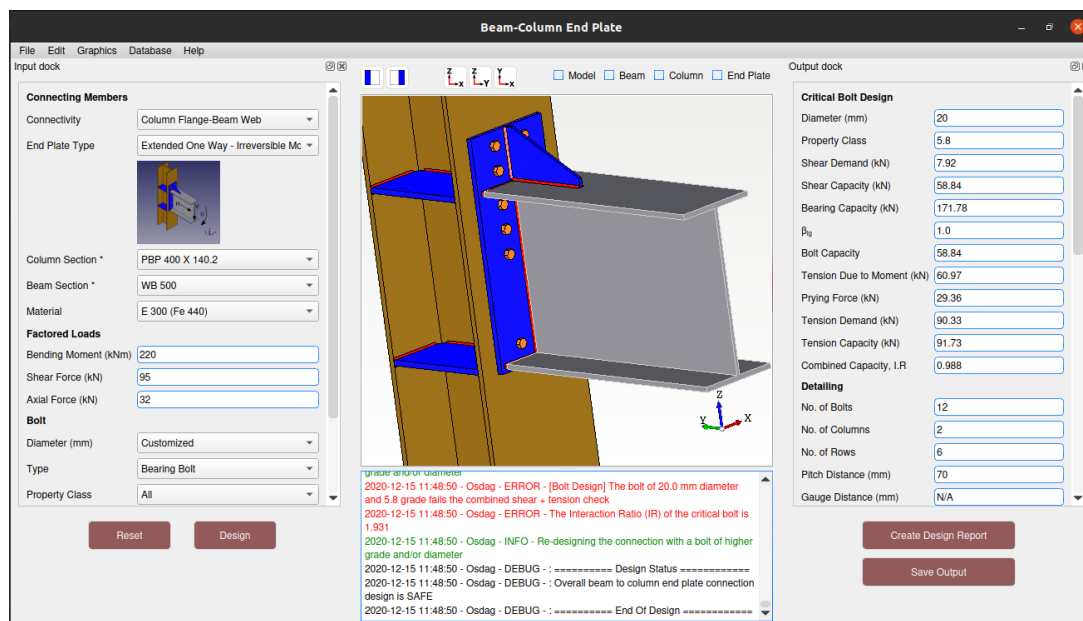


Figure 1.2: Osdag GUI

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

Understanding, cleaning, and analyzing raw data, followed by the visualization of 3D interactive shear force and bending moment diagrams. In these visualizations, each axis represents meaningful parameters, and each node represents essential structural information.

2.2 Tasks Done

As part of the screening task, the initial step involved understanding and interpreting the provided raw structural data. This included identifying key parameters such as nodes, elements, loads, and boundary conditions. The data was carefully cleaned and organized to remove inconsistencies, and ensure it was in a structured format suitable for analysis.

Following data preprocessing, analytical procedures were carried out to compute structural responses such as shear forces and bending moments for the given system. Appropriate methods and logical steps were applied to ensure accurate calculation and mapping of these values across different nodes and elements of the structure.

Finally, the processed data was utilized to develop 3D interactive visualizations of shear force and bending moment diagrams. In these visualizations, each axis represents meaningful parameters such as spatial position and force values, while nodes represent critical structural information. This helped in better understanding the variation and distribution of forces within the structure in an intuitive and visually interactive manner.

Chapter 3

Internship Task 1 Title

3.1 Task 1: Problem Statement

The objective of this task is to design and develop an interactive system for visualizing structural analysis data. The system should provide a widget-based interface that allows users to select relevant parameters such as structural elements, nodes, and loading conditions.

Based on the selected inputs, the system must process the underlying data and generate graphical representations, including shear force diagrams (SFD) and bending moment diagrams (BMD). Additionally, the system should generate contour visualizations specifically for bending moment (BMD contour), enabling a clearer understanding of the variation and distribution of bending moments across the structure.

The overall goal is to transform raw structural data into intuitive and interactive visual formats, facilitating efficient analysis and improved interpretation of structural behavior.

3.2 Task 1: Tasks Done

Methodology, Process, and Implementation

The overall workflow of the task was divided into multiple stages, including data pre-processing, user interaction through widgets, computational analysis, and visualization. Each stage was designed to ensure accurate processing and intuitive representation of

structural data.

1. Data Preprocessing

The raw structural data was first analyzed and cleaned to remove inconsistencies and missing values. The data was then structured into a suitable format containing information about nodes, elements, and loading conditions. This step ensured that the dataset was ready for further computation and visualization.

2. Widget-Based Selection

An interactive widget-based interface was developed to allow users to select parameters such as structural elements, nodes, and load cases. This provided flexibility and enabled dynamic control over the visualization process.

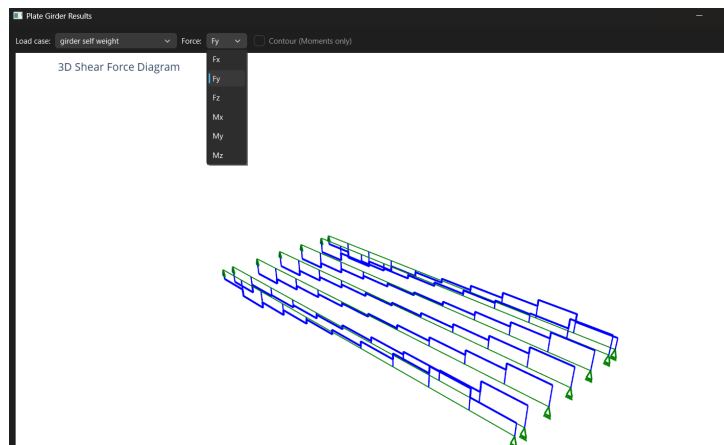


Figure 3.1: Widget-based interface for parameter selection

3. Computation of Structural Responses

Based on the selected inputs, shear force and bending moment values were computed for each element and node using appropriate structural analysis principles. These computed values formed the basis for further graphical representation.

4. Plotting of SFD and BMD

The computed values were used to generate shear force diagrams (SFD) and bending moment diagrams (BMD). These plots illustrated the variation of internal forces along

the length of structural elements.

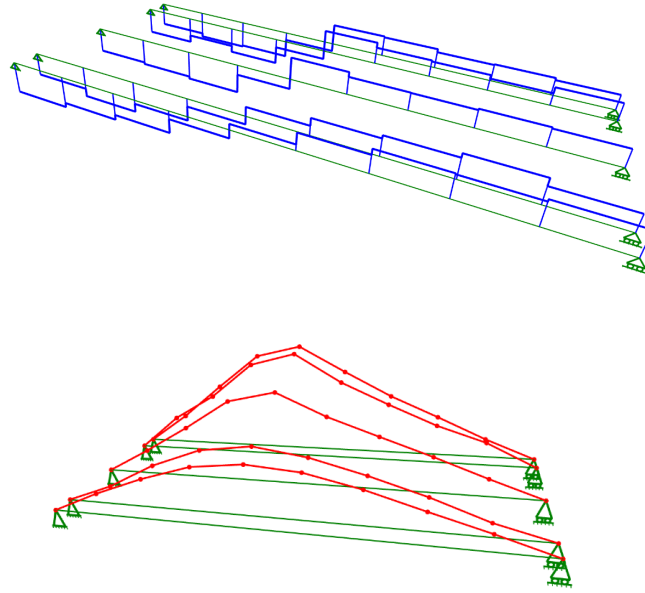


Figure 3.2: Shear Force and Bending Moment Diagrams

5. Bending Moment Contour Visualization

To enhance data interpretation, contour plots were generated specifically for bending moments. These contours provided a spatial view of moment distribution across the structure, making it easier to identify critical regions.

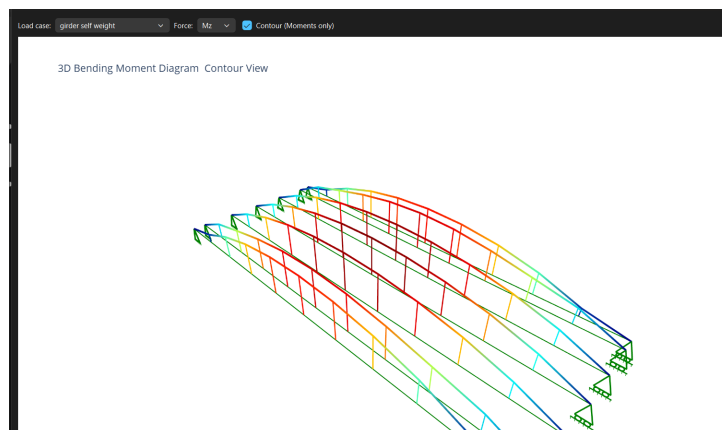


Figure 3.3: Bending Moment Contour Visualization

6. Integration and Implementation

All components were integrated into a unified workflow, enabling smooth interaction between data selection, computation, and visualization. The implementation ensured that user inputs dynamically updated the graphical outputs.

7. Summary of Process

Step	Process	Outcome
1	Data Preprocessing	Clean and structured data
2	Widget Selection	User-controlled inputs
3	Computation	SFD and BMD values
4	Plotting	Graphical diagrams (SFD, BMD)
5	Contour Visualization	BMD contour representation
6	Integration	Interactive system

Table 3.1: Summary of Methodology

3.3 Task 1: Python Code

This section presents the Python implementation developed during the internship for structural analysis and visualization of bridge grillage systems. The script integrates OpenSeesPy for structural modeling and analysis, Plotly for 3D interactive visualization, and PySide6 for building a graphical user interface. The application enables users to visualize shear force diagrams (SFD), bending moment diagrams (BMD), and bending moment contour plots based on different load cases.

3.3.1 Description of the Script

The script is structured as follows:

- **Structural Model Creation:** A bridge grillage model is created using OpenSeesPy, and multiple load cases such as self-weight, deck load, and barrier loads are applied.
- **Data Extraction:** The results from the structural analysis are extracted and organized based on elements, nodes, and force/moment components.

- **Visualization Functions:** Separate functions are implemented for generating 3D Shear Force Diagrams (SFD), Bending Moment Diagrams (BMD), and Bending Moment Contour plots.
- **Interactive Interface:** A PySide6-based GUI is developed to allow users to select load cases, force/moment components, and visualization type dynamically.
- **Rendering:** The plots are generated using Plotly and displayed in the interface using a web engine.

3.3.2 Python Code

The complete Python script used for structural visualization is provided below.

Listing 3.1: 3D Structural Visualization Tool using OpenSeesPy, Plotly, and PySide6

```

1  %-----begin code-----
2
3  import sys
4  import openseespy.opensees as ops
5  from pathlib import Path
6  from PySide6.QtWidgets import (
7      QApplication, QWidget, QVBoxLayout, QHBoxLayout,
8      QRadioButton, QButtonGroup, QLabel, QComboBox, QCheckBox
9  )
10 from PySide6.QtWebEngineWidgets import QWebEngineView
11 from PySide6.QtCore import QUrl
12
13 import numpy as np
14 import plotly.graph_objects as go
15
16
17
18 from .analyser import BridgeGrillageModel
19 FORCE_MAP = {
20     "Fx": ("Vx_i", "Vx_j"),
21     "Fy": ("Vy_i", "Vy_j"),
22     "Fz": ("Vz_i", "Vz_j"),
23     "Mx": ("Mx_i", "Mx_j"),
24     "My": ("My_i", "My_j"),
25     "Mz": ("Mz_i", "Mz_j"),
26 }
27
28
29 bridge = BridgeGrillageModel()
30 bridge.create_model()

```

```

31 bridge.create_self_weight_load()
32 bridge.create_deck_load()
33 bridge.create_wearing_course_load()
34 bridge.create_footpath_load()
35 bridge.create_crash_barrier_load()
36 bridge.create_railing_load()
37 bridge.create_median_load()
38 bridge.analyze()
39
40
41 results = bridge.model.get_results()
42
43 #results = convert_object_to_float(girder_results)
44
45
46
47 LOADCASES = [
48     "girder self weight",
49     "Deck slab load",
50     "Wearing course self weight",
51     "Footpath load",
52     "Crash barrier load",
53     "Railing load",
54     "Median load",
55 ]
56
57 ds_all = results
58 def get_ds(loadcase):
59     return ds_all.sel(Loadcase=loadcase)
60
61
62
63
64 # =====
65 # TEMP HTML (single file, overwritten)
66 '''
67 TEMP_HTML = (
68     Path(__file__).resolve()
69     .parent.parent.parent      # (file      dir      parent      parent)
70     / "temp_files"
71     / "temp_plot.html"
72 )
73 TEMP_HTML = str(TEMP_HTML)
74 '''
75
76 # Base directory (same as your logic)
77 BASE_DIR = (
78     Path(__file__).resolve()
79     .parent.parent.parent

```

```

80 )
81
82 # Ensure temp_files directory exists
83 TEMP_DIR = BASE_DIR / "temp_files"
84 TEMP_DIR.mkdir(parents=True, exist_ok=True)
85
86 # Final HTML file path
87 TEMP_HTML = TEMP_DIR / "temp_plot.html"
88
89 # If you really need string
90 TEMP_HTML = str(TEMP_HTML)
91
92 # COMMON IMPORTS (unchanged)
93
94
95 # LOAD NODES & ELEMENT CONNECTIVITY (unchanged)
96
97
98 # -----
99 # Extract all nodes from in-memory OpenSees model
100 # .....
101 # =====
102 # BUILD GEOMETRY ONCE (NO FILES, NO EXEC)
103 # =====
104
105 # Node coordinates
106 nodes = {
107     int(n): list(map(float, ops.nodeCoord(n)))
108     for n in ops.getNodeTags()
109 }
110
111 # Element connectivity
112 members = {
113     int(e): list(map(int, ops.eleNodes(e)))
114     for e in ops.getEleTags()
115 }
116
117
118
119
120
121
122
123
124 # =====
125 # SFD (UNCHANGED)
126 def build_figure_sfd(ds, force_key):
127
128     def find_component(name):

```

```

129         for c in ds["Component"].values:
130             if c.lower() == name.lower():
131                 return c
132         return None
133
134     comp_i_name, comp_j_name = FORCE_MAP[force_key]
135
136     comp_i = find_component(comp_i_name)
137     comp_j = find_component(comp_j_name)
138
139
140     def get_force(elem, comp):
141         return float(ds["forces"].sel(Element=elem, Component=comp).values)
142
143     # GIRDER GROUPING
144     Z_TOL = 3 # decimals for grouping (important!)
145
146     node_z = {}
147     for n in ops.getNodeTags():
148         z = float(ops.nodeCoord(n)[2])
149         node_z[int(n)] = round(z, Z_TOL)
150     from collections import defaultdict
151
152     girders = defaultdict(list)
153
154     for ele in ops.getEleTags():
155         n1, n2 = map(int, ops.eleNodes(ele))
156
157         z1 = node_z[n1]
158         z2 = node_z[n2]
159
160         # only longitudinal members (same Z at both ends)
161         if z1 == z2:
162             girders[z1].append(int(ele))
163
164
165     # BUILD GIRDER POLYLINES
166     def build_polyline(elem_list, comp_i, comp_j):
167         xs, ys, zs, vals, node_ids = [], [], [], [], []
168
169         for e in elem_list:
170             n1, n2 = members[e]
171             x1, y1, z1 = nodes[n1]
172
173             xs.append(x1)
174             ys.append(y1)
175             zs.append(z1)
176             vals.append(round(get_force(e, comp_i), 3))
177             node_ids.append(n1)

```

```

178
179     # Last end node
180     last_e = elem_list[-1]
181     n1, n2 = members[last_e]
182     x2, y2, z2 = nodes[n2]
183
184     xs.append(x2)
185     ys.append(y2)
186     zs.append(z2)
187     vals.append(round(get_force(last_e, comp_j),3))
188     node_ids.append(n2)
189
190     return np.array(xs), np.array(ys), np.array(zs), np.array(vals), node_ids
191 # ===== 3D STEPPED SFD =====
192
193 fig_sfd = go.Figure()
194
195
196 #girder_spacing = 3.0
197
198 for i, (gid, elems) in enumerate(girders.items()):
199
200     xs, ys, zs, vy, node_ids = build_polyline(elems, comp_i, comp_j)
201
202     Vy = vy.astype(float)
203
204     # use real Z from coordinates file
205     z_base = np.mean(zs) # or zs[0]
206
207     if max(Vy)-min(Vy) ==0:
208         shear_scale = 0.1 * abs((max(xs) - min(xs)) / (max(Vy) - 0))
209
210     else:
211         shear_scale = 0.1 * abs((max(xs) - min(xs)) / (max(Vy) - min(Vy)))
212
213     # ----- BASELINE (GROUND) -----
214     fig_sfd.add_trace(go.Scatter3d(
215         x=xs,
216         y=[0] * len(xs),
217         z=zs,#[z_base] * len(xs),
218         mode="lines",
219         line=dict(color="green", width=3),
220         hoverinfo="skip",
221         showlegend=False
222     ))
223
224     # ----- STEPPED SHEAR (MOUNTAINS) -----
225     x_step = np.repeat(xs, 2)[1:-1]
226     Vy_step = np.repeat(Vy[:-1], 2)

```

```

227
228     y_step = Vy_step * shear_scale
229     z_step = [z_base] * len(y_step)
230
231     fig_sfd.add_trace(go.Scatter3d(
232         x=x_step,
233         y=y_step,
234         z=z_step,
235         mode="lines",
236         line=dict(color="blue", width=6),
237         hoverinfo="text",
238         text=[
239             #f"Girder {gid}"
240             f"<br>Node {nid}"
241             f"<br>X = {x:.3f}"
242             f"<br>{force_key} = {v:.3f}"
243             for x, v, nid in zip(x_step, Vy_step, np.repeat(node_ids, 2)[1:-1])
244         ],
245         showlegend=False
246     ))
247
248     # ----- VERTICAL WALLS (CLIFFS) -----
249
250     for xi, vyi in zip(xs, Vy):
251         if xi == xs[-1]:
252             fig_sfd.add_trace(go.Scatter3d(
253                 x=[xi, xi],
254                 y=[0, -vyi * shear_scale],
255                 z=[z_base, z_base],
256                 mode="lines",
257                 line=dict(color="blue", width=4),
258                 hoverinfo="skip",
259                 showlegend=False
260             ))
261
262             # skip end nodes
263         else:
264             fig_sfd.add_trace(go.Scatter3d(
265                 x=[xi, xi],
266                 y=[0, vyi * shear_scale],
267                 z=[z_base, z_base],
268                 mode="lines",
269                 line=dict(color="blue", width=4),
270                 hoverinfo="skip",
271                 showlegend=False
272             ))
273
274     #fig_bmd = go.Figure()
275     # ----- SUPPORT GEOMETRY -----

```

```

276 L = max(xs) - min(xs)
277 h = 0.0199 * L      # support height
278 w = 0.006 * L       # support half-width
279 r = 0.0025 * L      # roller radius
280 # ----- PIN SUPPORT (START NODE) -----
281 x0, z0 = xs[0], zs[0]
282
283 fig_sfd.add_trace(go.Scatter3d(
284     x=[x0 - w, x0, x0 + w, x0 - w],
285     y=[-h, 0, -h, -h],
286     z=[z0, z0, z0, z0],
287     mode="lines",
288     line=dict(color="green", width=5),
289     showlegend=False,
290     hoverinfo="skip"
291 ))
292 # ===== PIN SUPPORT BASE (GROUND + HATCH) =====
293 n_hatch = 6
294 hatch_len = 0.15 * h
295
296 # ground line
297 fig_sfd.add_trace(go.Scatter3d(
298     x=[x0 - 1.3*w, x0 + 1.3*w],
299     y=[-h, -h],
300     z=[z_base, z_base],
301     mode="lines",
302     line=dict(color="green", width=4),
303     showlegend=False,
304     hoverinfo="skip"
305 ))
306
307 # hatch lines
308 xs_hatch = np.linspace(x0 - 1.2*w, x0 + 1.2*w, n_hatch)
309
310 for xh in xs_hatch:
311     fig_sfd.add_trace(go.Scatter3d(
312         x=[xh - 0.04*w, xh + 0.04*w],
313         y=[-h, -h - hatch_len],
314         z=[z_base, z_base],
315         mode="lines",
316         line=dict(color="green", width=2),
317         showlegend=False,
318         hoverinfo="skip"
319     ))
320
321
322 # ----- ROLLER SUPPORT (END NODE) -----
323 x1, z1 = xs[-1], zs[-1]
324

```

```

325     # Triangle
326     fig_sfd.add_trace(go.Scatter3d(
327         x=[x1 - w, x1, x1 + w, x1 - w],
328         y=[-h, 0, -h, -h],
329         z=[z1, z1, z1, z1],
330         mode="lines",
331         line=dict(color="green", width=5),
332         showlegend=False,
333         hoverinfo="skip"
334     ))
335     Yb = -h    # base of triangle
336
337
338     # Rollers (two wheels)
339     theta = np.linspace(0, 2 * np.pi, 60)
340
341     for dx in [-w / 2, w / 2]:
342         fig_sfd.add_trace(go.Scatter3d(
343             x = x1 + dx + r * np.cos(theta),
344             y = (Yb - r) + r * np.sin(theta),    #tangent to triangle base
345             z = [z1] * len(theta),
346             mode="lines",
347             line=dict(color="green", width=5),
348             showlegend=False,
349             hoverinfo="skip"
350         ))
351         # ===== ROLLER SUPPORT BASE (GROUND + HATCH) =====
352         # ===== ROLLER SUPPORT BASE (TANGENT TO ROLLERS) =====
353         n_hatch = 6
354         hatch_len = 0.15 * h
355
356         y_ground = Yb - 2*r    # tangent to roller bottom
357
358         # ground line
359         fig_sfd.add_trace(go.Scatter3d(
360             x=[x1 - 1.3*w, x1 + 1.3*w],
361             y=[y_ground, y_ground],
362             z=[z_base, z_base],
363             mode="lines",
364             line=dict(color="green", width=4),
365             showlegend=False,
366             hoverinfo="skip"
367         ))
368
369         # hatch lines
370         xs_hatch = np.linspace(x1 - 1.2*w, x1 + 1.2*w, n_hatch)
371
372         for xh in xs_hatch:
373             fig_sfd.add_trace(go.Scatter3d(

```

```

374         x=[xh - 0.04*w, xh + 0.04*w],
375         y=[y_ground, y_ground - hatch_len],
376         z=[z_base, z_base],
377         mode="lines",
378         line=dict(color="green", width=2),
379         showlegend=False,
380         hoverinfo="skip"
381     ))
382
383
384
385 fig_sfd.update_layout(
386     title="3D Shear Force Diagram",
387     scene=dict(
388
389
390         xaxis=dict(
391             showbackground=False,    showticklabels=False, title="",
392             showspikes=False
393
394         ),
395         yaxis=dict(
396             showbackground=False,    showticklabels=False, title="",
397             showspikes=False
398
399         ),
400         zaxis=dict(
401             showbackground=False,    showticklabels=False, title="",
402             showspikes=False, autorange="reversed"
403
404         ),
405
406         aspectmode="data",
407         camera=dict(
408             eye=dict(x=1.6, y=1.2, z=1.6),
409             up=dict(x=0, y=1, z=0)
410         ),
411
412         margin=dict(l=0, r=0, t=40, b=0)
413     )
414 fig_sfd.write_html(TEMP_HTML, include_plotlyjs=True, full_html=True)
415
416
417 # =====
418 # BMD
419

```

```

420
421
422
423
424 def build_figure_bmd(ds, force_key):
425     # LOAD INTERNAL FORCES (NETCDF)
426     def find_component(name):
427         for c in ds["Component"].values:
428             if c.lower() == name.lower():
429                 return c
430         return None
431
432     comp_i_name, comp_j_name = FORCE_MAP[force_key]
433     comp_i = find_component(comp_i_name)
434     comp_j = find_component(comp_j_name)
435
436     def get_force(elem, comp):
437         return float(ds["forces"].sel(Element=elem, Component=comp).values)
438
439     Z_TOL = 3 # decimals for grouping (important!)
440
441     node_z = {}
442     for n in ops.getNodeTags():
443         z = float(ops.nodeCoord(n)[2])
444         node_z[int(n)] = round(z, Z_TOL)
445     from collections import defaultdict
446
447     girders = defaultdict(list)
448
449     for ele in ops.getEleTags():
450         n1, n2 = map(int, ops.eleNodes(ele))
451
452         z1 = node_z[n1]
453         z2 = node_z[n2]
454
455         # only longitudinal members (same Z at both ends)
456         if z1 == z2:
457             girders[z1].append(int(ele))
458
459
460     # BUILD GIRDER POLYLINES
461     def build_polyline(elem_list, comp_i, comp_j):
462         xs, ys, zs, vals, node_ids = [], [], [], [], []
463
464         for e in elem_list:
465             n1, n2 = members[e]
466             x1, y1, z1 = nodes[n1]
467             xs.append(x1)
468             ys.append(y1)

```

```

469         zs.append(z1)
470         vals.append(round(get_force(e, comp_i),3))
471         node_ids.append(n1)
472         print(f"Element list: {elem_list}")
473         # Last end node
474         last_e = elem_list[-1]
475         n1, n2 = members[last_e]
476         x2, y2, z2 = nodes[n2]
477
478         xs.append(x2)
479         ys.append(y2)
480         zs.append(z2)
481         vals.append(round(get_force(last_e, comp_j),3))
482         node_ids.append(n2)
483
484         return np.array(xs), np.array(ys), np.array(zs), np.array(vals), node_ids
485
486     # PLOTLY 3D BMD INTERACTIVE
487     fig_bmd = go.Figure()
488     '''
489     xfull=[]
490     mzfull =[]
491     for gid, elems in girders.items():
492         xs, ys, zs, mz, node_ids = build_polyline(elems, Mz_i, Mz_j)
493         xfull.extend(xs)
494         mzfull.extend(mz)
495     diffxfull = max(xfull)-min(xfull)
496     diffmzfull = max(mzfull)-min(mzfull)
497     #factormz= abs(diffmzfull/diffxfull)*0.2
498     '''
499     for gid, elems in girders.items():
500         xs, ys, zs, mz, node_ids = build_polyline(elems, comp_i, comp_j)
501         if max(mz)-min(mz) ==0:
502             factormz = 0.1 * abs((max(xs) - min(xs)) / (max(mz) - 0))
503
504         else:
505             factormz = 0.1 * abs((max(xs) - min(xs)) / (max(mz) - min(mz)))
506         y_plot = mz*factormz ## 0.05 # moment scale
507         hover_text = [
508             f"Node {nid}<br>X = {x:.3f}<br>{force_key} = {v:.3f}<br>Z = {z:.3f}"
509             for nid, x, v,z in zip(node_ids, xs, mz,zs)
510         ]
511
512     fig_bmd.add_trace(go.Scatter3d(
513         x=xs,
514         y=y_plot,
515         z=zs,
516         mode='lines+markers',

```

```

518         line=dict(color="red", width=4),
519         marker=dict(size=3, color="red"),
520         #name=f"Girder {gid}",
521         showlegend=False,
522         text=hover_text,
523         hoverinfo="text"
524     ))
525
526
527     #baseline
528     fig_bmd.add_trace(go.Scatter3d(
529         x=[xs[0], xs[-1]],      # first and last node X
530         y=[0, 0],              # Mz = 0 baseline
531         z=[zs[0], zs[0]],      # same girder elevation
532         mode='lines',
533         line=dict(
534             color="green",
535             width=3,
536             dash='solid'
537         ),
538         showlegend=False,
539         hoverinfo='skip'
540     ))
541
542     fig_bmd.add_trace(go.Scatter3d(
543         x = [xs[np.argmax(mz)],xs[np.argmax(mz)]],
544         y = [0, max(mz)*factormz],
545         z = [zs[0]] * 2,
546         mode="lines",
547         line=dict(color="black", width=3),
548         legendgroup="max_lines",
549         showlegend=False,
550         visible=False,          # start OFF
551         hoverinfo="skip"
552     ))
553
554     fig_bmd.add_trace(go.Scatter3d(
555         x = [xs[np.argmin(mz)]] * 2,
556         y = [0, min(mz)*factormz],
557         z = [zs[0]] * 2,
558         mode="lines",
559         line=dict(color="black", width=3),
560         legendgroup="min_lines",
561         showlegend=False,
562         visible=False,          # start OFF
563         hoverinfo="skip"
564     ))
565
566

```

```

567 # ----- SUPPORT GEOMETRY -----
568 L = max(xs) - min(xs)
569 h = 0.0399 * L      # support height
570 w = 0.015 * L      # support half-width
571 r = 0.005 * L      # roller radius
572
573 # ----- PIN SUPPORT (START NODE) -----
574 x0, z0 = xs[0], zs[0]
575
576 fig_bmd.add_trace(go.Scatter3d(
577     x=[x0 - w, x0, x0 + w, x0 - w],
578     y=[-h, 0, -h, -h],
579     z=[z0, z0, z0, z0],
580     mode="lines",
581     line=dict(color="green", width=5),
582     showlegend=False,
583     hoverinfo="skip"
584 ))
585 n_hatch = 6
586 hatch_len = 0.15 * h
587
588 # ground line
589 fig_bmd.add_trace(go.Scatter3d(
590     x=[x0 - 1.3*w, x0 + 1.3*w],
591     y=[-h, -h],
592     z=[z0, z0],
593     mode="lines",
594     line=dict(color="green", width=4),
595     showlegend=False,
596     hoverinfo="skip"
597 ))
598
599 # hatch lines
600 xs_hatch = np.linspace(x0 - 1.2*w, x0 + 1.2*w, n_hatch)
601
602 for xh in xs_hatch:
603     fig_bmd.add_trace(go.Scatter3d(
604         x=[xh - 0.04*w, xh + 0.04*w],
605         y=[-h, -h - hatch_len],
606         z=[z0, z0],
607         mode="lines",
608         line=dict(color="green", width=2),
609         showlegend=False,
610         hoverinfo="skip"
611     ))
612
613 # ----- ROLLER SUPPORT (END NODE) -----
614 x1, z1 = xs[-1], zs[-1]
615
616 # Triangle

```

```

616     fig_bmd.add_trace(go.Scatter3d(
617         x=[x1 - w, x1, x1 + w, x1 - w],
618         y=[-h, 0, -h, -h],
619         z=[z1, z1, z1, z1],
620         mode="lines",
621         line=dict(color="green", width=5),
622         showlegend=False,
623         hoverinfo="skip"
624     ))
625     Yb = -h    # base of triangle
626
627
628     # Rollers (two wheels)
629     theta = np.linspace(0, 2 * np.pi, 60)
630
631     for dx in [-w / 2, w / 2]:
632         fig_bmd.add_trace(go.Scatter3d(
633             x = x1 + dx + r * np.cos(theta),
634             y = (Yb - r) + r * np.sin(theta),    # tangent to triangle base
635             z = [z1] * len(theta),
636             mode="lines",
637             line=dict(color="green", width=5),
638             showlegend=False,
639             hoverinfo="skip"
640         ))
641     n_hatch = 6
642     hatch_len = 0.15 * h
643
644     y_ground = Yb - 2*r    # tangent to roller bottom
645
646     # ground line
647     fig_bmd.add_trace(go.Scatter3d(
648         x=[x1 - 1.3*w, x1 + 1.3*w],
649         y=[y_ground, y_ground],
650         z=[z1, z1],
651         mode="lines",
652         line=dict(color="green", width=4),
653         showlegend=False,
654         hoverinfo="skip"
655     ))
656
657     # hatch lines
658     xs_hatch = np.linspace(x1 - 1.2*w, x1 + 1.2*w, n_hatch)
659
660     for xh in xs_hatch:
661         fig_bmd.add_trace(go.Scatter3d(
662             x=[xh - 0.04*w, xh + 0.04*w],
663             y=[y_ground, y_ground - hatch_len],
664             z=[z1, z1],

```

```

665         mode="lines",
666         line=dict(color="green", width=2),
667         showlegend=False,
668         hoverinfo="skip"
669     ))
670
671
672     #pad = 0.01*(max(xfull) - min(xfull))
673     #pad2 = 0.01*(max(mzfull) - min(mzfull))
674
675
676     max_idx = [i for i, t in enumerate(fig_bmd.data) if t.legendgroup == "max_lines"]
677     min_idx = [i for i, t in enumerate(fig_bmd.data) if t.legendgroup == "min_lines"]
678
679
680     fig_bmd.update_layout(
681     updatemenus=[
682         dict(
683             type="buttons",
684             direction="right",
685             x=0.5,
686             y=1.15,
687             showactive=True,
688             active=-1,
689             buttons=[
690                 # ----- MAX TOGGLE -----
691                 dict(
692                     label="MAX",
693                     method="update",
694
695                     # OFF      ON
696                     args=[{
697                         "visible": [
698                             True if t.legendgroup == "max_lines" else t.visible
699                             for t in fig_bmd.data
700                         ]
701                     }],
702
703                     # ON      OFF
704                     args2=[{
705                         "visible": [
706                             False if t.legendgroup == "max_lines" else t.visible
707                             for t in fig_bmd.data
708                         ]
709                     }],
710                 ),
711
712                 # ----- MIN TOGGLE -----
713                 dict(

```

```

714         label="MIN",
715         method="update",
716
717         # OFF      ON
718         args=[{
719             "visible": [
720                 True if t.legendgroup == "min_lines" else t.visible
721                 for t in fig_bmd.data
722             ]
723         }],
724
725         # ON      OFF
726         args2=[{
727             "visible": [
728                 False if t.legendgroup == "min_lines" else t.visible
729                 for t in fig_bmd.data
730             ]
731         }]
732     ),
733 ]
734 )
735 ],
736 title="Interactive 3D BMD",
737
738 scene=dict(
739
740     camera=dict(
741         #eye=dict(x=2.5, y=1, z=10), # pull camera along X
742         up=dict(x=0, y=0.5, z=0),
743         #center=dict(x=0, y=0, z=0),
744         #projection=dict(type="orthographic")
745     ),
746     xaxis=dict(
747         title = "girder length",
748         #range=[min(xfull) - pad, max(xfull) + pad],
749         showbackground=False, # removes YZ plane
750         showgrid=False,
751         zeroline=False,
752         visible=False
753     ),
754     yaxis=dict(
755         #range=[min(mzfull) - 0.25 * (max(mzfull) - min(mzfull)), max(mzfull) +
756             0.25 * (max(mzfull) - min(mzfull))],
757         title = "Mz values",
758         #range = [min(mzfull)-pad2, max(mzfull)+pad2],
759         showbackground=True, #
760         backgroundcolor="rgba(200,200,200,0.15)",
761         showgrid=False,
762         zeroline=False,

```

```

762         visible=False                                # showaxis line, keep plane
763     ),
764     zaxis=dict(
765         showbackground=False,    # removes XY plane
766         showgrid=False,
767         zeroline=False,
768         visible=False,
769         autorange="reversed"
770     ),
771     aspectmode='data',
772
773
774
775     ),
776
777
778     #legend=dict(title="Girders")
779 )
780
781 fig_bmd.write_html(TEMP_HTML, include_plotlyjs=True, full_html=True)
782
783 # =====
784 # BMD CONTOUR
785 def build_figure_bmd_contour(ds, force_key):
786
787     def find_component(name):
788         for c in ds["Component"].values:
789             if c.lower() == name.lower():
790                 return c
791         return None
792
793     comp_i_name, comp_j_name = FORCE_MAP[force_key]
794     comp_i = find_component(comp_i_name)
795     comp_j = find_component(comp_j_name)
796     def get_force(elem, comp):
797         return float(ds["forces"].sel(Element=elem, Component=comp).values)
798
799     # -----
800     # GIRDER GROUPING
801     # -----
802     Z_TOL = 3 # decimals for grouping (important!)
803
804     node_z = {}
805     for n in ops.getNodeTags():
806         z = float(ops.nodeCoord(n)[2])
807         node_z[int(n)] = round(z, Z_TOL)
808     from collections import defaultdict
809
810     girders = defaultdict(list)

```

```

811
812     for ele in ops.getEleTags():
813         n1, n2 = map(int, ops.eleNodes(ele))
814
815         z1 = node_z[n1]
816         z2 = node_z[n2]
817
818         # only longitudinal members (same Z at both ends)
819         if z1 == z2:
820             girders[z1].append(int(ele))
821
822
823     # -----
824     # BUILD GIRDER POLYLINE
825     # -----
826     def build_polyline(elem_list, comp_i, comp_j):
827         xs, ys, zs, mz, node_ids = [], [], [], [], []
828
829         for e in elem_list:
830             n1, n2 = members[e]
831             x1, y1, z1 = nodes[n1]
832
833             xs.append(x1)
834             ys.append(y1)
835             zs.append(z1)
836             mz.append(round(get_force(e, comp_i), 3))
837             node_ids.append(n1)
838
839             last_e = elem_list[-1]
840             n1, n2 = members[last_e]
841             x2, y2, z2 = nodes[n2]
842
843             xs.append(x2)
844             ys.append(y2)
845             zs.append(z2)
846             mz.append(round(get_force(last_e, comp_j), 3))
847             node_ids.append(n2)
848
849             return np.array(xs), np.array(ys), np.array(zs), np.array(mz), node_ids
850
851
852     xfull, mzfull = [], []
853     for elems in girders.values():
854         xs, ys, zs, mz, _ = build_polyline(elems, comp_i, comp_j)
855         xfull.extend(xs)
856         mzfull.extend(mz)
857
858
859

```

```

860     #moment_scale = 0.2 * abs(dx / dmz)    # visual-only scale
861
862     # -----
863     # FIGURE
864     # -----
865     fig = go.Figure()
866
867     for gid, elems in girders.items():
868         xs, ys, zs, mz, node_ids = build_polyline(elems, comp_i, comp_j)
869         if max(mz)-min(mz) ==0:
870             moment_scale = 0.1 * abs((max(xs) - min(xs)) / (max(mz) - 0))
871
872         else:
873             moment_scale = 0.1 * abs((max(xs) - min(xs)) / (max(mz) - min(mz)))
874         y_plot = mz * moment_scale
875
876         # ----- CONTOUR-STYLE BMD LINE -----
877         fig.add_trace(go.Scatter3d(
878             x=xs,
879             y=y_plot,
880             z=zs,
881             mode="lines",
882             line=dict(
883                 width=6,
884                 color=mz,                # color by Mz
885                 colorscale="Jet",
886                 cmin=min(mzfull),
887                 cmax=max(mzfull)
888             ),
889             showlegend=False,
890             hoverinfo="text",
891             text=[
892                 f"Node {nid}<br>X={x:.3f}<br>{force_key}={v:.3f}"
893                 for nid, x, v in zip(node_ids, xs, mz)
894             ]
895         ))
896
897         # ----- BASELINE (Mz = 0) -----
898         fig.add_trace(go.Scatter3d(
899             x=[xs[0], xs[-1]],
900             y=[0, 0],
901             z=[zs[0], zs[0]],
902             mode="lines",
903             line=dict(color="green", width=3),
904             hoverinfo="skip",
905             showlegend=False
906         ))
907
908         # ----- DROPLINES FROM BASELINE TO BMD -----

```

```

909     for xi, zi, mzi in zip(xs, zs, mz):
910
911         fig.add_trace(go.Scatter3d(
912             x=[xi, xi],
913             y=[0, mzi * moment_scale],    # baseline      BMD
914             z=[zi, zi],
915             mode="lines",
916             line=dict(
917                 width=4,
918                 color=[mzi, mzi],          # same color as contour
919                 colorscale="Jet",
920                 cmin=min(mzfull),
921                 cmap=max(mzfull)
922             ),
923             showlegend=False,
924             hoverinfo="skip"
925         ))
926
927     # ----- SUPPORT GEOMETRY -----
928     L = max(xs) - min(xs)
929     h = 0.0399 * L    # support height
930     w = 0.015 * L    # support half-width
931     r = 0.005 * L    # roller radius
932
933     # ----- PIN SUPPORT (FIRST NODE) -----
934     x0, z0 = xs[0], zs[0]
935
936     fig.add_trace(go.Scatter3d(
937         x=[x0 - w, x0, x0 + w, x0 - w],
938         y=[-h, 0, -h, -h],
939         z=[z0, z0, z0, z0],
940         mode="lines",
941         line=dict(color="green", width=5),
942         showlegend=False,
943         hoverinfo="skip"
944     ))
945
946     # ----- ROLLER SUPPORT (LAST NODE) -----
947     x1, z1 = xs[-1], zs[-1]
948
949     # Triangle
950     fig.add_trace(go.Scatter3d(
951         x=[x1 - w, x1, x1 + w, x1 - w],
952         y=[-h, 0, -h, -h],
953         z=[z1, z1, z1, z1],
954         mode="lines",
955         line=dict(color="green", width=5),
956         showlegend=False,
957         hoverinfo="skip"

```

```

958     ))
959
960     Yb = -h # base of triangle
961     theta = np.linspace(0, 2 * np.pi, 60)
962
963     # Two rollers
964     for dxi in [-w / 2, w / 2]:
965         fig.add_trace(go.Scatter3d(
966             x=x1 + dxi + r * np.cos(theta),
967             y=(Yb - r) + r * np.sin(theta),
968             z=[z1] * len(theta),
969             mode="lines",
970             line=dict(color="green", width=5),
971             showlegend=False,
972             hoverinfo="skip"
973         ))
974     n_hatch = 6
975     hatch_len = 0.15 * h
976
977     # ground line
978     fig.add_trace(go.Scatter3d(
979         x=[x0 - 1.3*w, x0 + 1.3*w],
980         y=[-h, -h],
981         z=[z0, z0],
982         mode="lines",
983         line=dict(color="green", width=4),
984         showlegend=False,
985         hoverinfo="skip"
986     ))
987
988     # hatch lines
989     xs_hatch = np.linspace(x0 - 1.2*w, x0 + 1.2*w, n_hatch)
990
991     for xh in xs_hatch:
992         fig.add_trace(go.Scatter3d(
993             x=[xh - 0.04*w, xh + 0.04*w],
994             y=[-h, -h - hatch_len],
995             z=[z0, z0],
996             mode="lines",
997             line=dict(color="green", width=2),
998             showlegend=False,
999             hoverinfo="skip"
1000         ))
1001
1002     y_ground = Yb - 2*r # tangent to roller bottom
1003
1004     # ground line
1005     fig.add_trace(go.Scatter3d(
1006         x=[x1 - 1.3*w, x1 + 1.3*w],

```

```

1007     y=[y_ground, y_ground],
1008     z=[z1, z1],
1009     mode="lines",
1010     line=dict(color="green", width=4),
1011     showlegend=False,
1012     hoverinfo="skip"
1013 ))
1014
1015 # hatch lines
1016 xs_hatch = np.linspace(x1 - 1.2*w, x1 + 1.2*w, n_hatch)
1017
1018 for xh in xs_hatch:
1019     fig.add_trace(go.Scatter3d(
1020         x=[xh - 0.04*w, xh + 0.04*w],
1021         y=[y_ground, y_ground - hatch_len],
1022         z=[z1, z1],
1023         mode="lines",
1024         line=dict(color="green", width=2),
1025         showlegend=False,
1026         hoverinfo="skip"
1027     ))
1028
1029 # LAYOUT (POST-PROCESSOR STYLE)
1030
1031 fig.update_layout(
1032     title="3D Bending Moment Diagram Contour View",
1033
1034     scene=dict(
1035
1036         camera=dict(
1037             #eye=dict(x=2.5, y=0.15, z=10), # pull camera along X
1038             up=dict(x=0, y=1, z=0),
1039             #center=dict(x=0, y=0, z=0),
1040             #projection=dict(type="orthographic")
1041         ),
1042         xaxis=dict(
1043             title = "girder length",
1044             #range=[min(xfull) - pad, max(xfull) + pad],
1045             showbackground=False, # removes YZ plane
1046             showgrid=False,
1047             zeroline=False,
1048             visible=False
1049         ),
1050         yaxis=dict(
1051             #range=[min(mzfull) - 0.25 * (max(mzfull) - min(mzfull)), max(mzfull) +
1052                 0.25 * (max(mzfull) - min(mzfull))],
1053             title = "Mz values",
1054             #range = [min(mzfull)-pad2, max(mzfull)+pad2],
1055             showbackground=True, #

```

```

1055         backgroundcolor="rgba(200,200,200,0.15)",
1056         showgrid=False,
1057         zeroline=False,
1058         visible=False           # showaxis line, keep plane
1059     ),
1060     zaxis=dict(
1061         showbackground=False,   # removes XY plane
1062         showgrid=False,
1063         zeroline=False,
1064         visible=False,
1065         autorange="reversed"
1066     ),
1067     aspectmode='data',
1068
1069
1070
1071     ),
1072 )
1073
1074 fig.write_html(TEMP_HTML, include_plotlyjs=True, full_html=True)
1075
1076 # =====
1077 # ===== QT WIDGET
1078 '''
1079 class PlotWidget(QWidget):
1080
1081     def __init__(self):
1082         super().__init__()
1083         self.setWindowTitle("Plate Girder Plots")
1084
1085         layout = QVBoxLayout(self)
1086
1087         top = QHBoxLayout()
1088         self.sfd = QRadioButton("SFD")
1089         self.bmd = QRadioButton("BMD")
1090         self.contour = QRadioButton("BMD Contour")
1091
1092         self.sfd.setChecked(True)
1093
1094         group = QButtonGroup(self)
1095         group.setExclusive(True)
1096         group.addButton(self.sfd)
1097         group.addButton(self.bmd)
1098         group.addButton(self.contour)
1099         group.buttonClicked.connect(self.update_plot)
1100
1101         top.addWidget(self.sfd)
1102         top.addWidget(self.bmd)
1103         top.addWidget(self.contour)

```

```

1104
1105     self.web = QWebEngineView()
1106
1107     layout.addLayout(top)
1108     layout.addWidget(self.web)
1109
1110     self.update_plot(self.sfd)
1111
1112     def update_plot(self, btn):
1113         if btn == self.sfd:
1114             build_figure_sfd()
1115         elif btn == self.bmd:
1116             build_figure_bmd()
1117         else:
1118             build_figure_bmd_contour()
1119
1120     self.web.load(QUrl.fromLocalFile(TEMP_HTML))
1121 '''
1122
1123 # ===== QT WIDGET
1124 # =====
1125 class PlotWidget(QWidget):
1126
1127     def __init__(self):
1128         super().__init__()
1129         self.setWindowTitle("Plate Girder Results")
1130
1131         layout = QVBoxLayout(self)
1132
1133         top = QHBoxLayout()
1134
1135         # ----- LOADCASE -----
1136         top.addWidget(QLabel("Load case:"))
1137
1138         self.combo = QComboBox()
1139         self.combo.addItems(LOADCASES)
1140         self.combo.currentTextChanged.connect(self.update_plot)
1141         top.addWidget(self.combo)
1142
1143         # ----- FORCE CHECKBOXES -----
1144         top.addWidget(QLabel("Result:"))
1145
1146         self.force_group = QButtonGroup(self)
1147         self.force_group.setExclusive(True) # only one selectable
1148
1149         self.force_checkboxes = {}
1150
1151         for key in FORCE_MAP.keys():
1152             cb = QCheckBox(key)

```

```

1153         self.force_group.addButton(cb)
1154         top.addWidget(cb)
1155         self.force_checkboxes[key] = cb
1156
1157         # Default selection
1158         self.force_checkboxes["Fy"].setChecked(True)
1159
1160         self.force_group.buttonClicked.connect(self.update_plot)
1161
1162
1163         # ----- CONTOUR -----
1164         self.contour = QCheckBox("Contour (Moments only)")
1165         self.contour.stateChanged.connect(self.update_plot)
1166         top.addWidget(self.contour)
1167
1168         top.addStretch()
1169
1170         self.web = QWebEngineView()
1171
1172         layout.addLayout(top)
1173         layout.addWidget(self.web)
1174
1175         self.update_plot()
1176
1177
1178
1179     def update_plot(self):
1180         loadcase = self.combo.currentText()
1181         # get selected checkbox
1182         force_key = None
1183         for key, cb in self.force_checkboxes.items():
1184             if cb.isChecked():
1185                 force_key = key
1186                 break
1187
1188         if force_key is None:
1189             return
1190
1191         ds = get_ds(loadcase)
1192
1193         # ----- FORCE TYPE -----
1194         is_force = force_key.startswith("F")    # Fx, Fy, Fz
1195         is_moment = force_key.startswith("M")   # Mx, My, Mz
1196
1197         # ----- UI LOGIC -----
1198         if is_force:
1199             # Forces      SFD, contour disabled
1200             self.contour.blockSignals(True)
1201             self.contour.setChecked(False)

```

```

1202         self.contour.setEnabled(False)
1203         self.contour.blockSignals(False)
1204
1205         build_figure_sfd(ds, force_key)
1206
1207         elif is_moment:
1208             # Moments      BMD (+ optional contour)
1209             self.contour.setEnabled(True)
1210
1211             if self.contour.isChecked():
1212                 build_figure_bmd_contour(ds, force_key)
1213             else:
1214                 build_figure_bmd(ds, force_key)
1215
1216         else:
1217             raise ValueError(f"Unsupported force: {force_key}")
1218
1219         # ----- UPDATE VIEW -----
1220         self.web.load(QUrl.fromLocalFile(TEMP_HTML))
1221
1222
1223 # ===== MAIN
1224 if __name__ == "__main__":
1225     app = QApplication(sys.argv)
1226     w = PlotWidget()
1227     w.resize(1200, 800)
1228     w.show()
1229     sys.exit(app.exec())
1230
1231
1232
1233 ##overview explanation
1234 '''
1235 # ----- MODEL CREATION -----
1236 bridge = BridgeGrillageModel()
1237 bridge.create_model()
1238 bridge.create_self_weight_load()
1239 bridge.create_deck_load()
1240 bridge.create_wearing_course_load()
1241 bridge.create_footpath_load()
1242 bridge.create_crash_barrier_load()
1243 bridge.create_railing_load()
1244 bridge.create_median_load()
1245 bridge.analyze()
1246
1247 results = bridge.model.get_results()
1248
1249 # ----- FORCE MAP -----
1250 FORCE_MAP = {

```

```

1251     "Fx": ("Vx_i", "Vx_j"),
1252     "Fy": ("Vy_i", "Vy_j"),
1253     "Fz": ("Vz_i", "Vz_j"),
1254     "Mx": ("Mx_i", "Mx_j"),
1255     "My": ("My_i", "My_j"),
1256     "Mz": ("Mz_i", "Mz_j"),
1257 }
1258
1259 # ----- LOAD CASES -----
1260 LOADCASES = [
1261     "girder self weight",
1262     "Deck slab load",
1263     "Wearing course self weight",
1264     "Footpath load",
1265     "Crash barrier load",
1266     "Railing load",
1267     "Median load",
1268 ]
1269
1270 def get_ds(loadcase):
1271     return results.sel(Loadcase=loadcase)
1272
1273 # ----- GEOMETRY -----
1274 nodes = {int(n): list(map(float, ops.nodeCoord(n))) for n in ops.getNodeTags()}
1275 members = {int(e): list(map(int, ops.eleNodes(e))) for e in ops.getEleTags()}
1276
1277 # ----- PLOT FUNCTIONS -----
1278 def build_figure_sfd(ds, force_key):
1279     fig = go.Figure()
1280     # SFD plotting logic (stepped diagram)
1281     fig.write_html("temp_plot.html")
1282
1283 def build_figure_bmd(ds, force_key):
1284     fig = go.Figure()
1285     # BMD plotting logic
1286     fig.write_html("temp_plot.html")
1287
1288 def build_figure_bmd_contour(ds, force_key):
1289     fig = go.Figure()
1290     # Contour plotting logic
1291     fig.write_html("temp_plot.html")
1292
1293 # ----- UI -----
1294 class PlotWidget(QWidget):
1295     def __init__(self):
1296         super().__init__()
1297
1298         layout = QVBoxLayout(self)
1299 
```

```

1300     self.combo = QComboBox()
1301     self.combo.addItem(LoadCases)
1302
1303     self.web = QWebEngineView()
1304
1305     layout.addWidget(self.combo)
1306     layout.addWidget(self.web)
1307
1308     self.combo.currentTextChanged.connect(self.update_plot)
1309     self.update_plot()
1310
1311     def update_plot(self):
1312         loadcase = self.combo.currentText()
1313         ds = get_ds(loadcase)
1314
1315         build_figure_bmd(ds, "Mz")
1316         self.web.load(QUrl.fromLocalFile("temp_plot.html"))
1317
1318 # ----- MAIN -----
1319 if __name__ == "__main__":
1320     app = QApplication(sys.argv)
1321     w = PlotWidget()
1322     w.show()
1323     sys.exit(app.exec())
1324
1325 %----- end code -----
1326 '''

```

3.3.3 Explanation of the Code

- **Model Initialization:** The script initializes a bridge grillage model and applies multiple load cases using OpenSeesPy.
- **Data Processing:** Structural results are extracted and filtered based on load cases and force/moment components.
- **SFD Implementation:** Shear force diagrams are generated using step-wise plotting techniques.
- **BMD Implementation:** Bending moment diagrams are plotted in 3D with proper scaling and node mapping.
- **Contour Visualization:** Bending moment contour plots are created using color gradients to represent variation across the structure.

- **User Interface:** A PySide6-based GUI allows users to interactively select parameters and update visualizations in real time.
- **Visualization Rendering:** Plotly is used to generate interactive HTML plots, which are displayed within the application using a web engine.

3.4 Task 1: Documentation

This section presents the contribution made towards the Osdag Developer/User Manual. The work involves the implementation of a Python-based interactive visualization module for bridge grillage analysis, including Shear Force Diagram (SFD), Bending Moment Diagram (BMD), and Bending Moment Contour visualization.

3.4.1 Directory Structure (`src/osdagbridge/core/bridge_types/plate_girder`)

```
plate_girder
├── analyser.py
├── widget_plot.py
├── bridge_grillage_model
├── plot_functions
└── ui_components
```

Description of Files

- **analyser.py:** Contains the implementation of the bridge grillage model using OpenSeesPy, including load application and structural analysis.
- **widget_plot.py:** Implements the interactive visualization tool, including SFD, BMD, and BMD contour plotting functions along with the GUI.
- **Bridge Grillage Model:** Defines nodes, elements, and load cases required for structural analysis.
- **Plot Functions:** Includes functions for generating 3D SFD, BMD, and contour plots using Plotly.

- **UI Components:** Contains PySide6-based widgets such as dropdowns and checkboxes for user interaction.

3.4.2 Program Execution

The visualization module can be executed through the Python script containing the widget interface. Since the module resides within the `src` directory, it should be run using Python's module execution mode. Navigate to the `src` directory and execute the following command:

```
$ python -m osdagbridge.core.bridge_types.plate_girder.widget_plot
```

3.4.3 Workflow Description

The workflow of the module is as follows:

- The bridge grillage model is created and analyzed using OpenSeesPy.
- Structural results are extracted and organized based on load cases and force/moment components.
- The user selects the desired load case and result type (force or moment) using the graphical interface.
- Based on the selection:
 - Shear forces are visualized as 3D SFD plots.
 - Bending moments are visualized as 3D BMD plots.
 - Contour visualization is generated for bending moments (BMD contour).
- The plots are rendered as interactive HTML visualizations using Plotly and displayed within the application.

Chapter 4

Steel Design and Deck Design Output Dialog Implementation

4.1 Task 2: Problem Statement

The bridge design application required a structured output interface to display design results computed by the engineering backend. Specifically, engineers needed to inspect detailed structural outputs — member properties, dimensional details, shear connector specifications, section properties, stiffener data, analysis results, and design checks — for both the steel girder and the concrete deck slab after a design run. Previously, no such output dialog existed in the desktop application; results were unavailable.

The task was to design and implement two modal output dialogs — **Steel Design** and **Deck Design** — that present computed results in a consistent, readable UI matching the existing application design language. The dialogs needed to be launched from the output dock, populated from the application’s `cad_state` data dictionary, and follow the same visual patterns established by the existing Additional Inputs dialogs (such as the Girder Details sub-tab).

4.2 Task 2: Tasks Done

The following components were designed and implemented:

- **Steel Design Dialog** (`steel_design.py`): A `QDialog` of size 800×800 px with a

branded green header and a `QTabWidget` containing three tabs — Details, Analysis Results, and Design Check.

- **Steel Design Details Tab (`steel_design_details.py`):** Displays member info (member ID, grade, type), dimensional details (depth, web/flange thickness and width, restraint types, effective slab width), shear connector details (material, diameter, height, spacing, stud count), section properties (mass, area, moments of inertia, radii of gyration, elastic and plastic moduli, torsion and warping constants), and a stiffener details table (intermediate, longitudinal, bearing stiffeners). Each section is enclosed in a styled card frame matching the `GirderDetailsTab` visual pattern — `QFrame#girderCard` with `border: 1px solid #cfcfcf` and `border-radius: 10px`. The stiffener table is wrapped in a `QFrame` border container to prevent Qt's viewport from clipping the left border, and a `NoScrollTable` subclass is used to pass wheel events to the parent scroll area so the page remains scrollable when the cursor is over the table.
- **Steel Design Analysis Tab (`steel_design_analysis.py`):** Displays member and load combination selection dropdowns, global result fields (R_A , R_B , $M_{\max}$, $V_{\max}$, $D_{\max}$), a diagram placeholder, and position-based fields (M_x , V_x , D_x) aligned beside the diagram zones. The member combo is dynamically populated via `set_girder_count(n)` driven by the `no_of_girders` value in `cad_state`.
- **Steel Design Check Tab (`steel_design_check.py`):** Placeholder tab reserved for structured design check output.
- **Deck Design Dialog (`deck_design.py`):** A standalone `QDialog` of size 800×800 px displaying deck properties (grade of material, thickness), a reinforcement details table (position, material yield strength, diameter, spacing, clear cover, area for top and bottom layers), and a design check text area. The same card frame pattern, table border fix, `NoScrollTable`, and `QTimer`-based height fitting are applied.
- **Integration:** The dialogs are launched from `output_dock.py` button signals, routed through `template_page.py` methods (`open_steel_design` and `open_deck_design`), which instantiate the dialogs, call `load_data(cad_state)`, and execute them modally.

Several UI consistency issues were identified and resolved during implementation:

- Spacing between section title and first field was inconsistent — fixed by adopting uniform card layout margins of (18, 16, 18, 16) and spacing of 10px across all sections.
- Table left border was invisible due to Qt's `QAbstractScrollArea` viewport painting over the widget border — fixed by wrapping the table in a `QFrame` that owns the border, with the table itself set to `border: none`.
- Double border on the right and bottom edges of tables — fixed by setting `gridline-color` for internal lines only, suppressing `border-right` on the last header section via `QHeaderView::section:last`, and using `setViewportMargins(0, 0, -1, -1)` to tuck the last gridlines under the frame border.
- Page scroll blocked when hovering over table — fixed by subclassing `QTableWidget` as `NoScrollTable` with `wheelEvent` calling `event.ignore()`, allowing events to bubble up to the `StyledScrollArea`.
- Reinforcement table height not fitting content — fixed using `QTimer.singleShot(0, ...)` to measure actual rendered header and row heights after Qt layout and apply `setFixedHeight` at runtime.

4.3 Task 2: Documentation

4.3.1 File Structure

osdagbridge/desktop/ui/

dialogs/

steel_design.py # Main Steel Design dialog

deck_design.py # Deck Design dialog

tabs/

steel_design_details.py # Details tab

steel_design_analysis.py # Analysis Results tab

steel_design_check.py # Design Check tab (placeholder)

docks/

output_dock.py # Button signals (modified)

```

    utils/
        styled_scroll_area.py          # Shared scroll area utility
core/
    template_page.py                  # Dialog launch methods (modified)

```

4.3.2 Data Flow

Results are passed from the engineering backend into a `cad_state` dictionary on `template_page`. When a dialog is opened, `load_data(cad_state)` distributes values to all fields and table cells. The `cad_state` keys follow a consistent naming convention, for example `stiff_intermediate_grade`, `rebar_top_yield`, `shear_diameter`, and so on.

4.3.3 Key Design Decisions

- **Card frame pattern:** All sections use `QFrame#girderCard` matching the existing `GirderDetailsTab`, ensuring visual consistency across the entire application.
- **Table border isolation:** A `QFrame` wrapper owns the outer border of every table, decoupling it from Qt's internal viewport rendering which causes the left-border clipping issue.
- **NoScrollTable:** A lightweight subclass that prevents accidental table scroll while preserving page scroll, without disabling scrollbars programmatically (which would block parent scroll events).
- **Runtime height fitting:** Table heights are computed after render using `QTimer.singleShot` to handle variable header heights caused by multi-line header text across different DPI settings.
- **Dynamic member selection:** The Analysis tab member combo is driven by `set_girder_count(n)`, keeping it in sync with the number of girders defined in the input dock — no hardcoding.

Chapter 5

Conclusions

5.1 Tasks Accomplished

The work carried out during the fellowship involved the development of both analytical and user interface components for bridge design applications.

The first task focused on the implementation of a Python-based structural visualization module capable of generating interactive 3D representations of shear force diagrams (SFD), bending moment diagrams (BMD), and bending moment contour plots. The system integrated structural analysis using OpenSeesPy with interactive visualization through Plotly and provided a widget-based interface for dynamic user selection of load cases and result types.

The second task involved the design and implementation of structured output dialogs for steel design and deck design within a desktop application environment. This included the development of modular dialog interfaces, integration with backend data through the `cad_state` dictionary, and resolution of multiple UI challenges related to layout consistency, table rendering, and scroll behavior. The implemented solution ensured a consistent and user-friendly presentation of design outputs.

5.2 Skills Developed

The fellowship contributed to the development of a strong foundation in both technical and practical aspects of engineering software development. Technical skills include structural analysis using OpenSeesPy, data processing and visualization using Plotly, and user

interface development using PySide6 within a desktop application framework.

In addition, significant experience was gained in handling real-world software challenges such as UI consistency, event handling, layout management, and integration between frontend interfaces and backend computational modules. The work also strengthened problem-solving abilities, debugging skills, and the ability to design scalable and maintainable software components aligned with existing application architecture.

Chapter A

Appendix

A.1 Work Reports

Internship Work Report

Name: Md Arbaz Khan

Project: Osdag Bridge

Internship: Osdag Winter Intern 2025

DATE	DAY	TASK	Hours Worked
11-Dec-2025	Thursday	Set up development environment, cloned OsdagBridge repository, explored codebase structure	4
12-Dec-2025	Friday	Studied existing PySide6 UI architecture, reviewed InputDock and OutputDock modules	5
13-Dec-2025	Saturday	Explored OpenSeesPy grillage model and BridgeGrillageModel analyser class	5
14-Dec-2025	Sunday	Studied xarray dataset structure returned by get_results() for internal forces	4
15-Dec-2025	Monday	Began Task 1: designed PlotWidget UI with load case combo and force component checkboxes	5
16-Dec-2025	Tuesday	Implemented FORCE_MAP and girder grouping logic by Z-coordinate tolerance	6
17-Dec-2025	Wednesday	Built 3D SFD (Shear Force Diagram) using Plotly Scatter3d with stepped polylines	6
18-Dec-2025	Thursday	Added pin and roller support geometry (triangle, rollers, hatch lines) to SFD	5
19-Dec-2025	Friday	Implemented 3D BMD (Bending Moment Diagram) with baseline, MAX/MIN toggle buttons	6
20-Dec-2025	Saturday	Added contour BMD view with Jet colorscale and droplines from baseline to BMD curve	6
21-Dec-2025	Sunday	Integrated Plotly HTML output with QWebEngineView, implemented TEMP_HTML file routing	5
22-Dec-2025	Monday	Connected force checkbox group to update_plot(), handled force vs moment diagram switching	6
23-Dec-2025	Tuesday	Tested all load cases and force components, fixed scaling issues for near-zero force ranges	5
24-Dec-2025	Wednesday	Refined camera angles and scene layout for 3D diagrams, removed redundant axis labels	4
25-Dec-2025	Thursday	Holiday	0
26-Dec-2025	Friday	Reviewed Task 1 with mentor, incorporated feedback on hover text and support geometry sizing	5
27-Dec-2025	Saturday	Began Task 2: studied GirderDetailsTab card frame and inner box UI patterns	5
28-Dec-2025	Sunday	Designed SteelDesign QDialog (800x800) with green header and QTabWidget (3 tabs)	5
29-Dec-2025	Monday	Implemented SteelDesignDetailsTab: member info, dimensional details, shear connector sections	6
30-Dec-2025	Tuesday	Added section properties grid and stiffener details QTableWidgetItem to details tab	6

Internship Work Report

Name: Md Arbaz Khan

Project: Osdag Bridge

Internship: Osdag Winter Intern 2025

DATE	DAY	TASK	Hours Worked
31-Dec-2025	Wednesday	Implemented SteelDesignAnalysisTab with selection panel, results fields, diagram placeholder	6
01-Jan-2026	Thursday	Holiday	0
02-Jan-2026	Friday	Built DeckDesign QDialog with deck properties, reinforcement table, and design check sections	6
03-Jan-2026	Saturday	Integrated Steel Design and Deck Design dialogs into output_dock.py and template_page.py	5
04-Jan-2026	Sunday	Applied girderCard frame pattern (border: 1px solid #cfcfcf, border-radius: 10px) to all sections	6
05-Jan-2026	Monday	Fixed table left border clipping by wrapping QTableWidgetItem in QFrame border container	6
06-Jan-2026	Tuesday	Implemented NoScrollTable subclass with wheelEvent ignore for page scroll preservation	5
07-Jan-2026	Wednesday	Fixed double right/bottom border using setViewportMargins(0,0,-1,-1) and section:last CSS	6
08-Jan-2026	Thursday	Fixed reinforcement table height using QTimer.singleShot for runtime header height measurement	6
09-Jan-2026	Friday	Resolved spacing inconsistency between section titles and first fields across all cards	5
10-Jan-2026	Saturday	Standardised all helpers: _create_card_frame, _create_label, _create_small_label, _make_grid	5
11-Jan-2026	Sunday	Applied same card/table rules to DeckDesign dialog, verified visual consistency	5
12-Jan-2026	Monday	Tested dialogs with sample cad_state data, verified load_data() populates all fields correctly	6
13-Jan-2026	Tuesday	Fixed Member Info card horizontal auto-expansion using QSizePolicy.Maximum	5
14-Jan-2026	Wednesday	Reviewed complete UI for both dialogs with mentor, incorporated final feedback	5
15-Jan-2026	Thursday	Wrote unit tests for load_data() methods and verified set_girder_count() dynamic combo	5
16-Jan-2026	Friday	Final code cleanup, removed unused helpers, added docstrings to all public methods	5
17-Jan-2026	Saturday	Prepared documentation for both tasks, drafted internship report sections	6
18-Jan-2026	Sunday	Wrote LaTeX documentation for Task 1: problem statement, tasks done, technical details	5
19-Jan-2026	Monday	Wrote LaTeX documentation for Task 2: problem statement, tasks done, file structure	5

Internship Work Report

Name: Md Arbaz Khan

Project: Osdag Bridge

Internship: Osdag Winter Intern 2025

DATE	DAY	TASK	Hours Worked
20-Jan-2026	Tuesday	Reviewed full internship report with mentor, incorporated feedback	5
21-Jan-2026	Wednesday	Final testing of complete OsdagBridge output module integration	5
22-Jan-2026	Thursday	Prepared final presentation of both tasks for mentor review	5
23-Jan-2026	Friday	Final review meeting with mentor, discussed internship outcomes and improvements	4
24-Jan-2026	Saturday	Compiled all deliverables and submitted final internship report	5
25-Jan-2026	Sunday	Buffer day for any remaining fixes and documentation updates	4
26-Jan-2026	Monday	Final code submission and repository cleanup	4
27-Jan-2026	Tuesday	Prepared handover documentation for future contributors	4
28-Jan-2026	Wednesday	Internship closing: final meeting and submission of all work	4

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.