Osdag Screening Task Report

Author: Rishav Kumar Sinha

Email: sinharishavkumar@gmail.com

Branch: Mechanical Engineering

Institution: National Institute of Technology, Silchar

Date: 6th April, 2025

GitHub Repository: https://github.com/RishavKumarSinha/osdag-screening-task

---

Abstract

This report outlines the development and implementation of two integrated tasks that demonstrate the use of Python-based tools in structural engineering design. The first task involves visualizing structural data by generating shear force and bending moment diagrams using PyPlot. The second task focuses on creating a detailed 3D computer-aided design (CAD) model of a laced compound column using PythonOCC. Together, these tasks highlight the potential for automation in the design and detailing processes of steel structures, showcasing the analytical and geometric modeling capabilities available to engineers today.

Introduction

Osdag is well-known for its use in the design and detailing of steel structures. In this project, the primary goal was to connect data visualisation with 3D modeling by utilizing Python's powerful libraries. The first task involved creating structural diagrams, specifically shear force and bending moment plots, based on experimental or analytical data stored in an Excel sheet. The second task focused on constructing a 3D CAD model of a laced compound column, a common feature in modern structural design due to its effective load distribution. Together, these tasks not only demonstrate the versatility of Python in engineering applications but also highlight an integrated approach to visual analysis and geometric construction.

Methodology

The project was executed in two distinct parts, each employing a different Python library to achieve the intended outcome.

Data Visualization Using PyPlot

The first component of the project involved reading structural data from an Excel spreadsheet using the Pandas library. The data, which comprised distance, shear force, and bending moment values, was processed and then visualized using Matplotlib's PyPlot module. The resulting plots provide an immediate graphical representation of the shear force and bending moment distribution along a structural member. The code reads the Excel file, extracts the relevant columns, and employs a two-panel subplot to display the diagrams side by side. This visualization aids in understanding the behavior of the structural element under load, providing critical insights into points of potential failure or stress concentration.

3D CAD Modeling Using PythonOCC

For the second task, the focus shifted to the creation of a three-dimensional model of a laced compound column using PythonOCC, which is the Python wrapper for the OpenCASCADE CAD kernel. The column design integrates two I-shaped sections, spaced 450mm apart, with end plates and lacing elements that provide additional stability. The modeling process involved generating simple geometric primitives to represent the I-sections and end plates, then applying translations and rotations to position the components accurately. Diagonal and horizontal lacing elements were introduced using calculated angles and offsets to replicate the physical configuration depicted in the reference drawings. Once assembled, the complete model was exported as a STEP file, a widely accepted format for CAD data exchange, ensuring compatibility with other CAD systems and further downstream processing.

Task 1: SFD and BMD Plotting Using PyPlot

Data Reading and Preparation

The process begins by reading structural data from an Excel file using the Pandas library. The following snippet illustrates the essential step of importing data and verifying its correctness:

```
import pandas as pd


# Reading the Excel file

df = pd.read_excel('data/SFS_Screening_SFDBMD.xlsx')

print(df.head())  # Confirming that data is loaded correctly
```

This step ensures that the columns for distance, shear force, and bending moment are correctly imported, setting the stage for plotting.

Plot Generation

Once the data is available, Matplotlib's PyPlot module is used to generate the diagrams. The code creates a two-panel plot where each panel is dedicated to one of the diagrams. Crucial elements include setting appropriate axis labels, titles, and grid lines to enhance readability:

```
import matplotlib.pyplot as plt


# Extracting data for plotting

x = df['Distance (m)']

shear_force = df['SF (kN)']

bending_moment = df['BM (kN-m)']


# Creating subplots for the diagrams

fig, axs = plt.subplots(1, 2, figsize=(12, 5))


# Plotting the Shear Force Diagram

axs[0].plot(x, shear_force, color='blue')

axs[0].set_title('Shear Force Diagram')

axs[0].set_xlabel('Distance (m)')

axs[0].set_ylabel('Shear Force (kN)')

axs[0].grid(True)


# Plotting the Bending Moment Diagram

axs[1].plot(x, bending_moment, color='green')

axs[1].set_title('Bending Moment Diagram')

axs[1].set_xlabel('Distance (m)')

axs[1].set_ylabel('Bending Moment (kNm)')

axs[1].grid(True)
```

```
plt.tight_layout()

plt.show()
```

Discussion

This approach effectively visualizes the internal force distributions along the structural member. The choice of subplotting not only optimizes space but also allows for an immediate side-by-side comparison of the two diagrams, aiding in structural analysis.

Task 2: Laced Compound Column CAD Modeling Using PythonOCC

Conceptual and Geometric Approach

The second task involved creating a 3D model of a laced compound column. This model includes two I-sections separated by a defined gap, end plates at the top and bottom, and connecting lacing elements that provide additional stability.

Crucial Transformations

A few helper functions are central to the CAD modeling process. For instance, the `translate` function applies a translation transformation to a shape. This abstraction simplifies the positioning of various components:

```
from OCC.Core.BRepBuilderAPI import BRepBuilderAPI_Transform

from OCC.Core.gp import gp_Trsf, gp_Vec


def translate(shape, dx=0, dy=0, dz=0):

    trsf = gp_Trsf()

    trsf.SetTranslation(gp_Vec(dx, dy, dz))

    return BRepBuilderAPI_Transform(shape, trsf, True).Shape()
```

Discussion

This function is used repeatedly to position the I-sections, end plates, and lacing elements accurately. It encapsulates the transformation logic, making the code modular and easier to maintain.

Assembly and STEP File Export

After constructing individual components, the model is assembled and then exported as a STEP file. A crucial section of the code handles the STEP export, ensuring that all parts of the model are correctly transferred:

```
from OCC.Core.STEPControl import STEPControl_Writer, STEPControl_AsIs

from OCC.Core.IFSelect import IFSelect_RetDone


# STEP file export section

step_writer = STEPControl_Writer()

step_writer.Transfer(i_section1, STEPControl_AsIs)

step_writer.Transfer(i_section2, STEPControl_AsIs)

step_writer.Transfer(top_plate, STEPControl_AsIs)

step_writer.Transfer(bottom_plate, STEPControl_AsIs)

for lace in laces:

    step_writer.Transfer(lace, STEPControl_AsIs)


status = step_writer.Write(output_file)

if status == IFSelect_RetDone:

    print(f"STEP file exported successfully to: {output_file}")

else:

    print("STEP export failed.")
```

Discussion

Exporting the model as a STEP file is critical for interoperability with other CAD systems. This section of the code ensures that all geometric entities are correctly written out, preserving the integrity of the assembled model.

Results and Discussion

The implementation of the shear force and bending moment diagrams was validated through direct comparison with reference plots. The diagrams not only correctly reflected the input data but also provided a clear visual insight into the structural behavior. Any discrepancies in the data handling process were minimal, largely due to the robust functionality of Pandas in managing spreadsheet inputs.

In parallel, the 3D CAD model generated via PythonOCC captured the essential features of a laced compound column. The model includes precise geometric transformations, which ensure that the relative positions of the I-sections, end plates, and lacing elements are consistent with the design specifications. Visual inspection using a PythonOCC viewer confirmed that the model met the required criteria, although it was noted that for production-level design, further refinement of the I-section profile and additional detailing might be necessary.

The integrated approach demonstrates how modern programming techniques can streamline the traditional tasks of structural analysis and CAD modeling. By automating both the data visualization and geometric construction processes, engineers can significantly reduce the time and effort required for preliminary design and validation.

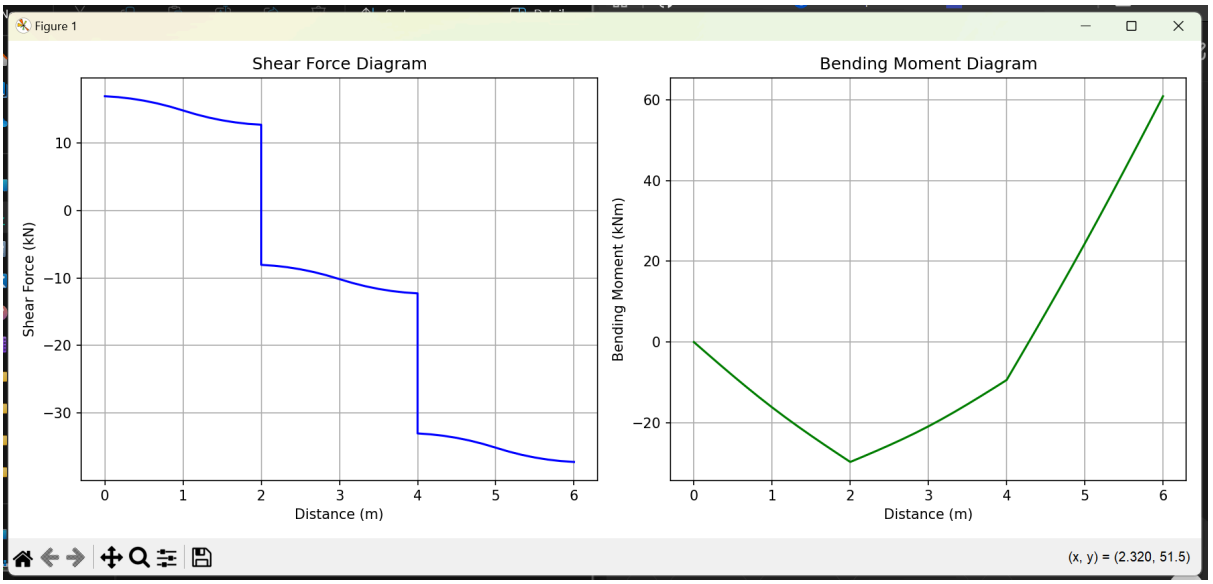
Conclusion

This project successfully merged two distinct aspects of structural engineering: data-driven analysis and geometric modeling. The PyPlot-based solution provided immediate insights into the structural performance through the shear force and bending moment diagrams, while the PythonOCC-based CAD model delivered a tangible representation of the laced compound column. The methods and results presented herein not only validate the approach but also pave the way for further enhancements, such as the integration of parametric design and real-time data processing in structural engineering applications.

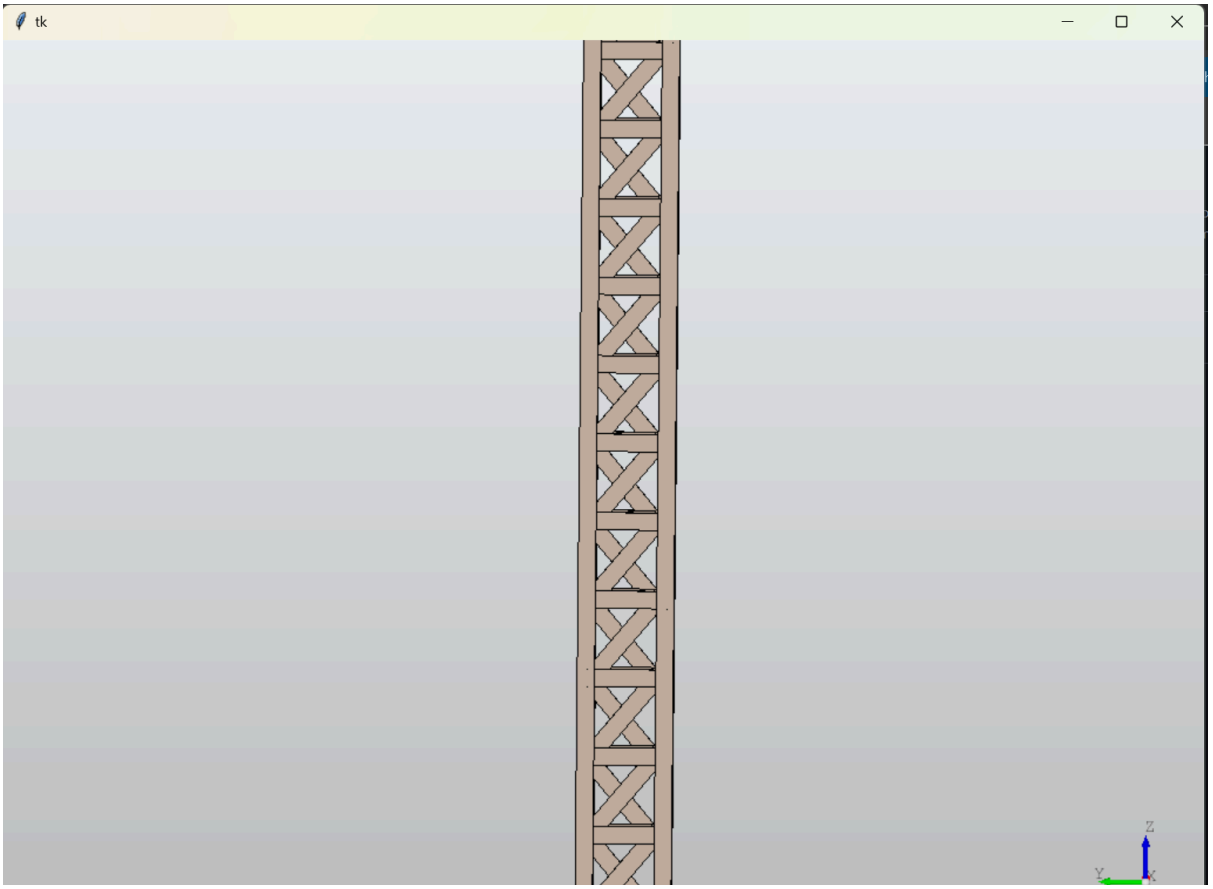
Future work may involve refining the CAD model to include more detailed cross-sectional properties and dynamic simulation of load responses, as well as integrating these models into a broader software framework for automated design verification.

Acknowledgements

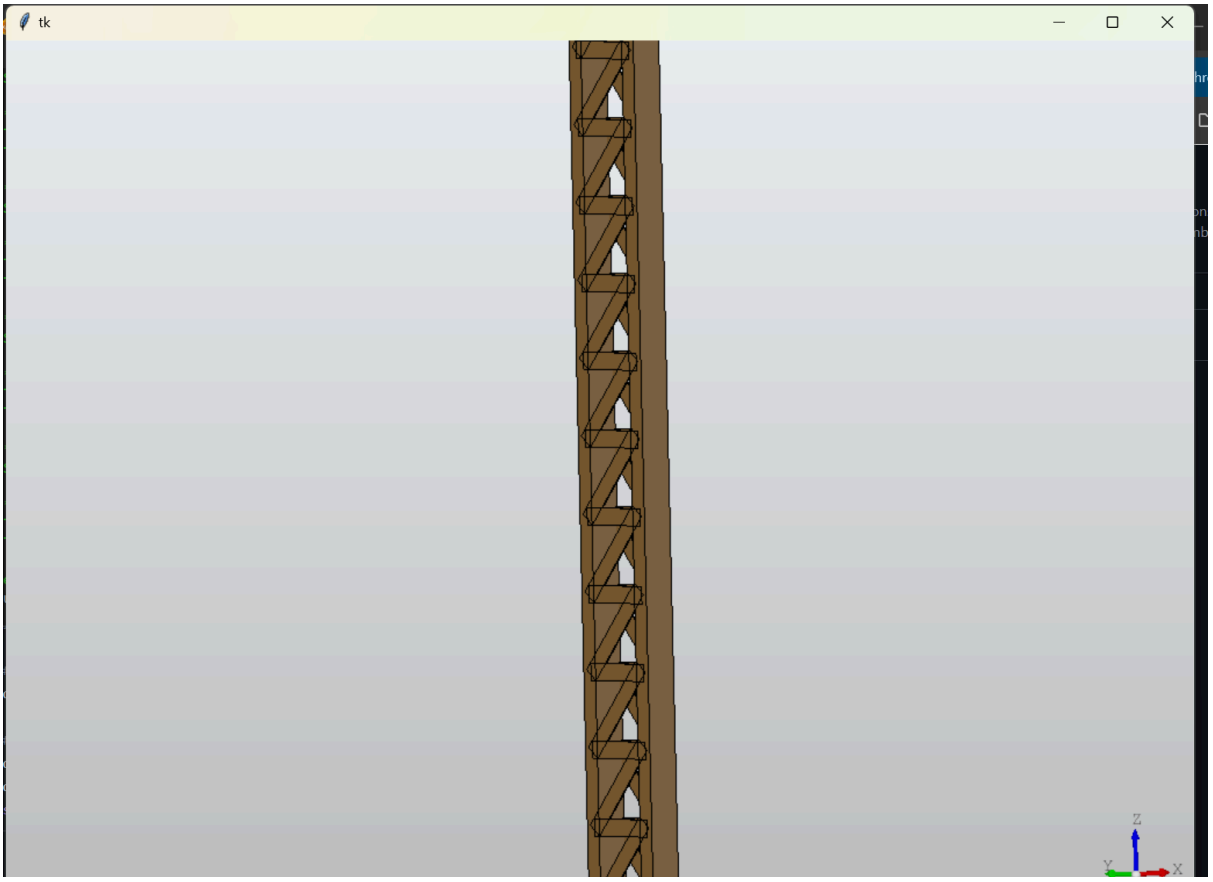
I would like to thank the FOSSEE team at IIT Bombay for providing this opportunity to explore and integrate modern computational tools in structural engineering. Their support and the open-source community have been invaluable in the successful execution of this project.



(Screenshot of the SFD and BMD plots)



(Front view of the CAD model of the laced compound column)



(Orthogonal view of the CAD model of the laced compound column)

References

- Osdag Documentation. (n.d.). Retrieved from [https://osdag.fossee.in/]
- OpenCASCADE. (n.d.). OpenCASCADE Technology.
- Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3), 90–95.
- McKinney, W. (2010). Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference.
