FOSSEE Winter Internship Report

On

Development and Implementation of IRC-Based Bridge Design Modules in OsdagBridge

Submitted by

Sweta Pal

4th Year B.E Student, Department of Computer Science and Engineering

AMC Engineering College

Bengaluru, Karnataka

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Nidhi Khare

Aditya Donde

February 19, 2026

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

Chapter 1

Introduction

1.1 National Mission in Education through ICT

The National Mission on Education through ICT (NMEICT) is a scheme under the Department of Higher Education, Ministry of Education, Government of India. It aims to leverage the potential of ICT to enhance teaching and learning in Higher Education Institutions in an anytime-anywhere mode.

The mission aligns with the three cardinal principles of the Education Policy—**access, equity, and quality**—by:

- Providing connectivity and affordable access devices for learners and institutions.
- Generating high-quality e-content free of cost.

NMEICT seeks to bridge the digital divide by empowering learners and teachers in urban and rural areas, fostering inclusivity in the knowledge economy. Key focus areas include:

- Development of e-learning pedagogies and virtual laboratories.
- Online testing, certification, and mentorship through accessible platforms like EduSAT and DTH.
- Training and empowering teachers to adopt ICT-based teaching methods.

For further details, visit the official website: www.nmeict.ac.in.

1.1.1 ICT Initiatives of MoE

The Ministry of Education (MoE) has launched several ICT initiatives aimed at students, researchers, and institutions. The table below summarizes the key details:

No.	Resource	For Students/Researchers	For Institutions
Audio-Video e-content			
1	SWAYAM	Earn credit via online courses	Develop and host courses; accept credits
2	SWAYAMPRABHA	Access 24x7 TV programs	Enable SWAYAMPRABHA viewing facilities
Digital Content Access			
3	National Digital Library	Access e-content in multiple disciplines	List e-content; form NDL Clubs
4	e-PG Pathshala	Access free books and e-content	Host e-books
5	Shodhganga	Access Indian research theses	List institutional theses
6	e-ShodhSindhu	Access full-text e-resources	Access e-resources for institutions
Hands-on Learning			
7	e-Yantra	Hands-on embedded systems training	Create e-Yantra labs with IIT Bombay
8	FOSSEE	Volunteer for open-source software	Run labs with open-source software
9	Spoken Tutorial	Learn IT skills via tutorials	Provide self-learning IT content
10	Virtual Labs	Perform online experiments	Develop curriculum-based experiments
E-Governance			
11	SAMARTH ERP	Manage student lifecycle digitally	Enable institutional e-governance
Tracking and Research Tools			
12	VIDWAN	Register and access experts	Monitor faculty research outcomes
13	Shodh Shuddhi	Ensure plagiarism-free work	Improve research quality and reputation
14	Academic Bank of Credits	Store and transfer credits	Facilitate credit redemption

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

1.2 FOSSEE Project

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

1.2.1 Projects and Activities

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

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

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

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

For more details, visit the official FOSSEE website.

Figure 1.1: FOSSEE Projects and Activities

1.3 Osdag Software

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

1.3.1 Osdag GUI

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

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

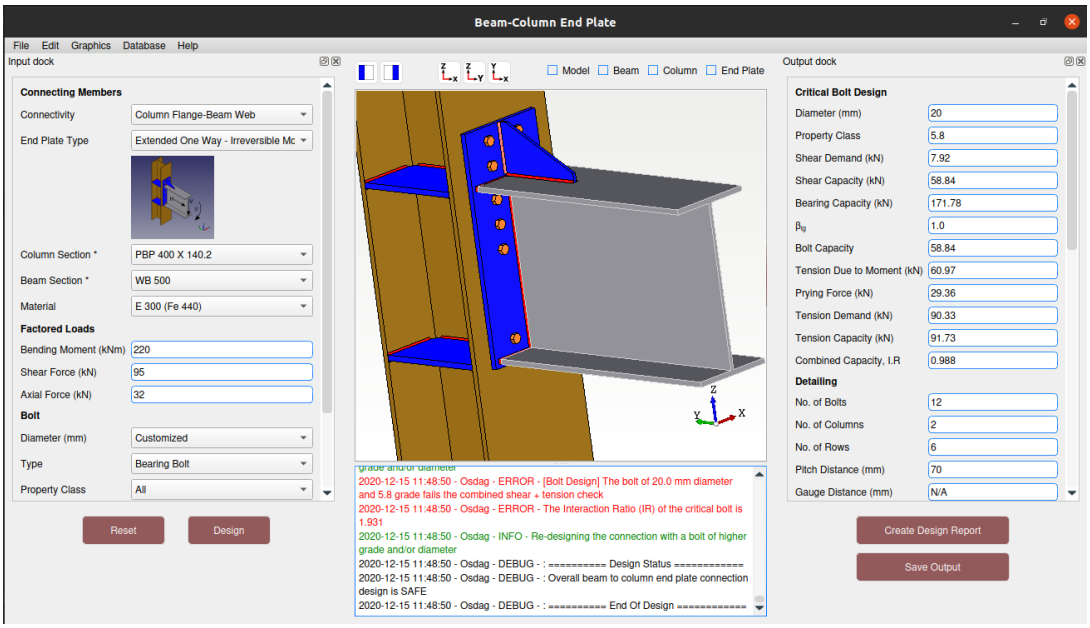

Figure 1.2: Osdag GUI

1.3.2 Features

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

For more details, visit the official Osdag website.

Chapter 2

Screening Task

2.1 Problem Statement

The objective of the screening task was to develop a Python script that reads force data from an Excel file and automatically generates a structured engineering PDF report using PyLaTeX. The report was required to include a beam image, a LaTeX-based input data table, and vector-based Shear Force and Bending Moment diagrams created using TikZ/pgfplots.

2.2 Tasks Done

The following tasks were performed to complete the screening assignment:

1. **Understanding the Requirements**

   The task requirements were analyzed to identify the expected report structure, input format, and visualization constraints. Special attention was given to the requirement that all tables and diagrams must be generated programmatically and not inserted as images.

2. **Input Data Handling**

   The force data containing position, shear force, and bending moment values was read from an Excel file using the `pandas` library. The data was extracted and stored in Python lists for further processing.

3. **Data Preparation for Plotting**

   The numerical values were converted into coordinate pairs suitable for TikZ/pgfplots. These coordinates were formatted as strings so that they could be directly embedded into the LaTeX plotting commands.

4. **Automated Report Generation using PyLaTeX**

   A Python script was developed using the PyLaTeX library to programmatically create a structured LaTeX document. The script generates all report sections, formatting, and layout automatically.

5. **Beam Description**

   An image of the simply supported beam was inserted into the report using the LaTeX figure environment to provide a visual representation of the structural system.

6. **Input Table Creation**

   The Excel data was recreated as a LaTeX tabular environment to ensure that the table content is selectable text rather than an image, satisfying the task requirement.

7. **Shear Force Diagram (SFD)**

   The shear force distribution along the beam length was plotted using TikZ/pgfplots. The diagram was generated using the programmatically created coordinate data.

8. **Bending Moment Diagram (BMD)**

   Similarly, the bending moment variation along the beam was plotted using TikZ/pgfplots to produce a scalable vector-based diagram.

9. **Automation and File Management**

   The script was designed to automatically overwrite any previously generated report and produce an updated PDF each time it is executed.

10. **Project Organization and Submission**

    The project files were organized into a structured directory. A GitHub repository was created, a demonstration video was recorded, and a ZIP file containing all relevant files was prepared for submission.

2.3 Folder Structure

The project files were organized as follows:

```
Pylatex_osdag/
|-- main.py
|-- ForceTable.xlsx
|-- beam.png
|-- Beam_Report.pdf
|-- README.md
```

2.4 Conclusion

The screening task was successfully completed by developing a Python-based automated workflow for generating an engineering report using PyLaTeX. The script reads force data from an Excel file, recreates the input table in LaTeX format, and generates vector-based Shear Force and Bending Moment diagrams using TikZ/pgfplots. The project demonstrates the integration of data processing, document automation, and engineering visualization using open-source tools.

Chapter 3

Seated Angle Connection Module Testing

3.1 Task 1: Problem Statement

The objective of this task was to perform functional testing of the **Seated Angle Connection** module in the Osdag desktop application. The purpose of testing was to verify the stability of the module, validate the input–design workflow, and identify any runtime or functional issues that could affect usability.

The testing involved executing the complete design process with different input configurations and checking the behavior of the software during file operations and result generation.

3.2 Task 1: Tasks Done

3.2.1 Understanding the Module Workflow

- Installed and configured the Osdag desktop application.
- Studied the input parameters and design workflow of the Seated Angle Connection module.
- Performed multiple trial runs with different input values to understand the module behavior.

3.3 Issues Identified During Testing

During the testing of the Seated Angle Connection module, two major functional issues were identified. The details of each issue are described below.

3.3.1 Bug 1: Application Crash During Design Execution

Summary

When using the Seated Angle Connection module, the application closes automatically after locking the inputs and clicking the *Design* button. No design output is generated, and the software exits abruptly without displaying any error message.

Steps to Reproduce

1. Open the *Seated Angle Connection* module in the Osdag GUI.
2. Import an existing `.osi` file or manually enter input values.
3. Enter a valid shear force (e.g., 50 kN).
4. Select bolt and angle sections (or keep default values).
5. Click the *Lock Inputs* button.
6. Click the *Design* button.

Observed Behavior

- A *Loading* dialog appears.
- After 1–2 seconds, the Osdag application closes automatically.
- No output is generated:
  - No design report
  - No PDF
  - No 3D model
  - No error message

Expected Behavior

The application should complete the design process successfully and generate the design output without crashing.

3.3.2 Bug 2: Imported `.osi` File Cannot Be Saved

Summary

When an existing `.osi` file is imported into the Seated Angle Connection module and modified, the file cannot be saved. The save operation fails, preventing the user from storing updated inputs.

Steps to Reproduce

1. Open the *Seated Angle Connection* module.
2. Import an existing `.osi` file.
3. Modify any input parameter.
4. Click the *Save* option.

Observed Behavior

- The file is not saved.
- Changes are not stored.
- No confirmation message is shown.

Expected Behavior

The modified `.osi` file should be saved successfully without errors.

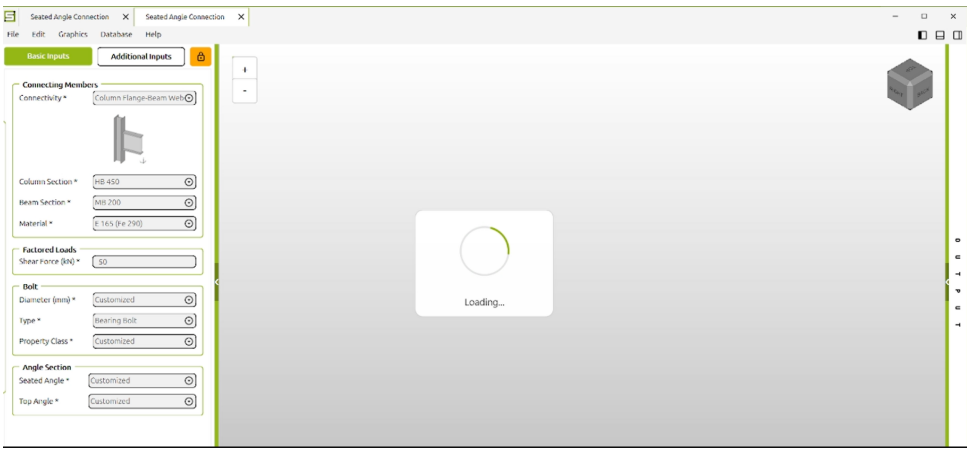

Figure 3.1: Error observed while saving imported .osi file in Seated Angle Connection module

3.3.3 Reporting and Documentation

- Both issues were raised on github and documented with detailed steps to reproduce.
- The observations were communicated to the development team for further investigation.

3.4 Outcome

This task helped in gaining familiarity with the Osdag interface, module workflow, and file handling system. It also provided practical exposure to software testing, bug identification, and issue reporting, which contributed to improving the reliability and usability of the Seated Angle Connection module.

Chapter 4

Implementation of IRC 5 Crash Barrier Geometry and Load Modules for OsdagBridge

4.1 Task 2: Problem Statement

Bridges require safety systems such as crash barriers, railings, and medians to protect users and structural components. IRC 5:2015 specifies standard cross-sectional geometries and loading requirements for these components through various figures and clauses.

The objective of this task was to develop Python modules within the OsdagBridge framework to implement the geometry and load calculations for crash barrier systems as per IRC 5:2015.

The developed modules were required to:

- Compute geometric properties such as dimensions and cross-sectional area.
- Implement different configurations based on bridge layout conditions.
- Calculate design loads where specified.
- Support multiple IRC figures for crash barriers and safety elements.
- Integrate seamlessly with the OsdagBridge design workflow.

4.2 Task 2: Tasks Done

4.2.1 Implementation of Fig. 1: Details of Concrete Crash Barrier & Railing for Two-Lane Bridge with Footpath

IRC Reference: Fig. 1(a) – With RCC Railing Fig. 1(b) – With Steel Railing

Objective

The objective of this module was to compute the accurate cross-sectional area and geometric properties of the rigid RCC crash barrier integrated with a footpath and railing system as specified in IRC 5:2015.

The implementation supports two configurations:

- RCC Railing
- Steel Railing

File Locations

The implementation is distributed across the following files:

```
src/osdagbridge/core/irc/irc5_2015.py
    - Contains IRC clause functions and geometric dimensions


src/osdagbridge/core/utils/codes/keyfile.py
    - Contains common constants such as:
        KEY_FOOTPATH
        KEY_CRASH_BARRIER_TYPE
        KEY_RAILING_TYPE


src/osdagbridge/core/bridge_components/super_structure/crash_barrier/geometry.py
    - Contains area calculation function:
        rigid_barrier_with_railing_area(railing)


src/osdagbridge/core/bridge_components/super_structure/crash_barrier/properties.py
```

- Contains load calculations functions:

    rcc_railing_load()

    steel_railing_load()

IRC Figure

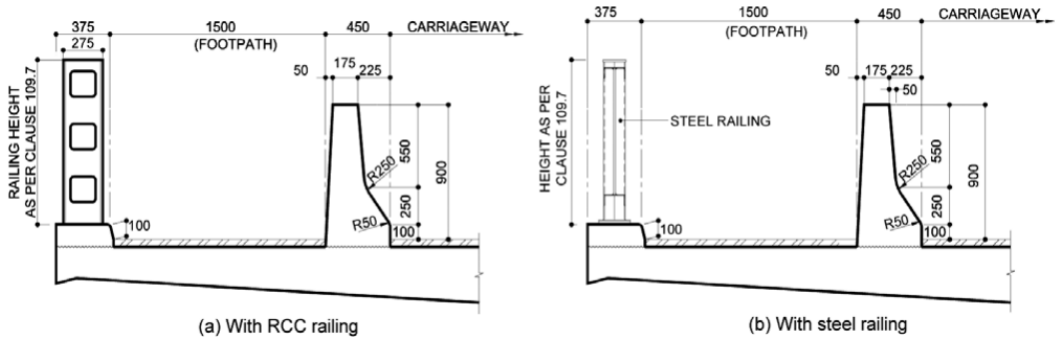

Figure 4.1: IRC 5:2015 Fig. 1(a) and Fig. 1(b): Details of Concrete Crash Barrier and Railing for Two-Lane Bridge with Footpath

Role of IRC5_2015.py

All IRC Clause 109.6.3 shape definitions are implemented in:

`IRC5_2015.cl_109_6_3_shapes()`

This function provides:

- Barrier height
- Top and bottom notch dimensions
- Middle section length
- Edge radii
- Wearing course thickness

These values are extracted based on:

- Barrier type = Rigid
- Footpath = Present
- Railing type = RCC / Steel

Use of Keyfile

The file `keyfile.py` contains all common parameters used across IRC modules. For Fig. 1 implementation, the following keys are used:

- `KEY_CRASH_BARRIER_TYPE[2]` – Rigid Barrier
- `KEY_FOOTPATH[1]` – Footpath Present
- `KEY_RAILING_TYPE[0]` – RCC Railing
- `KEY_RAILING_TYPE[1]` – Steel Railing

Area Calculation Methodology

The cross-section of the crash barrier is geometrically complex and consists of sloping faces and curved transitions. To accurately compute the area, the section was divided into simpler components:

1. Trapezoidal Segments The straight portions of the barrier were divided into three trapezoids:

- Upper trapezoid (height = 550 mm)
- Middle trapezoid (height = 250 mm)
- Bottom trapezoid (including wearing course thickness)

Area of each trapezoid is calculated as:

$$A = \frac{1}{2}(a + b) \times h$$

Total trapezoidal area:

$$A_{trap} = A_1 + A_2 + A_3$$

2. Curved Portion Calculation The upper and lower curved transitions are circular arcs with radii:

- $R_1 = 50$ mm (small curve)
- $R_2 = 250$ mm (large curve)

The curved area is computed using the circular segment formula derived from tangent geometry:

$$A_{segment} = R^2 \left(\tan \frac{\theta}{2} - \frac{\theta}{2} \right)$$

The net curved area is:

$$A_{curve} = A_{large} - A_{small}$$

3. Total Area

$$A_{total} = A_{trap} + A_{curve}$$

This method ensures high accuracy and matches manual engineering calculations.

Python Function

The area calculation is implemented in:

`rigid_barrier_with_railing_area(railing)`

The function:

- Accepts railing type (RCC / Steel)
- Fetches geometry from IRC clause function
- Computes trapezoidal and curved areas
- Returns total cross-sectional area

Functions were developed to compute geometric parameters and cross-sectional area of RCC crash barriers.

The following cases were implemented:

- Barrier with footpath
- Barrier without footpath
- IRC-5R standard barrier
- High containment barrier

The geometry includes:

- Barrier height
- Top and bottom widths
- Notch dimensions
- Edge radii
- Middle section length

The cross-sectional area is calculated using a combination of rectangular and curved segments.

Self-Weight Load Calculation

In addition to geometric area computation, the self-weight of the crash barrier was calculated to obtain the equivalent line load (kN/m) acting on the bridge deck.

This module performs the following:

- Imports cross-sectional area functions from `geometry.py`
- Converts area (mm^2) into load (kN/m)
- Provides separate functions for each IRC configuration

Material Density Assumptions

The following material densities are used:

- RCC density = 25 kN/m^3
- Steel density = 78 kN/m^3 (used for metallic barriers in later figures)

For Fig. 1, the crash barrier body is made of RCC. Even in the steel railing case, the self-weight of the RCC barrier is calculated based on RCC density.

Area to Load Conversion

The cross-sectional area obtained from the geometry module is in mm^2 per metre length.

Conversion steps:

$$\text{Area } (m^2) = \frac{\text{Area } (mm^2)}{10^6}$$

$$\text{Load (kN/m)} = \text{Area } (m^2) \times \text{Density}$$

This conversion is implemented using:

`load_from_area(area_mm2, density)`

Load Functions for Fig. 1

Two load functions were implemented:

- `rcc_railing_load()`

  Computes self-weight for Fig. 1(a) – Rigid barrier with RCC railing.

- `steel_railing_load()`

  Computes self-weight for Fig. 1(b) – Rigid barrier with steel railing.

Each function:

- Calls `rigid_barrier_with_railing_area()`
- Converts the area into kN/m using RCC density
- Returns the total barrier self-weight per metre length

Output

The functions return a dictionary containing:

- Barrier type (IRC configuration)
- RCC barrier load (kN/m)
- Total load per metre length

This load is used as a dead load input for subsequent structural analysis in the OsdagBridge workflow.

4.2.2 Implementation of Fig. 2: Details of Crash Barrier for Bridges without Footpath

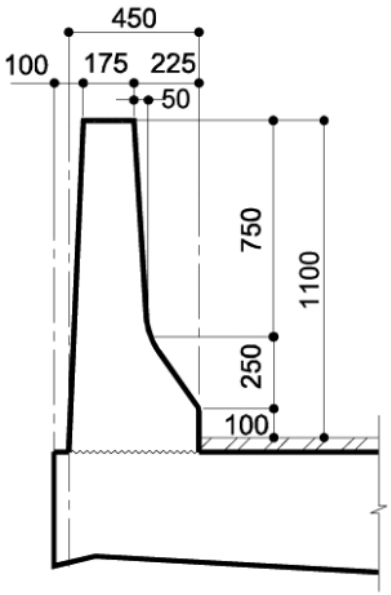

Fig. 2 Details of Crash Barrier for Bridges without Footpath

Figure 4.2: IRC 5:2015 Fig. 2: Details of Crash Barrier for Bridges without Footpath

Configuration

This configuration represents the standard IRC-5R rigid crash barrier used when no footpath is provided. The geometry is obtained from:

`IRC5_2015.cl_109_6_3_shapes()`

with:

- Barrier type: Rigid
- Footpath: None
- Crash barrier type: IRC-5R

File Locations

`src/osdagbridge/core/bridge_components/super_structure/crash_barrier/geometry.py`

`rigid_barrier_no_footpath_area()`

`src/osdagbridge/core/bridge_components/super_structure/crash_barrier/properties.py`

`rigid_barrier_no_footpath_load()`

Area Calculation Method

The cross-section is divided into three trapezoidal regions and curved transition portions.

Trapezoidal Areas

$$A_1 = \frac{1}{2}(W_{top} + W_{mid})H_{top}$$

$$A_2 = \frac{1}{2}(W_{mid} + W_{base})H_{mid}$$

$$A_3 = W_{base} \times H_{bottom}^{eff}$$

Total trapezoidal area:

$$A_{trap} = A_1 + A_2 + A_3$$

Curved Portion The upper and lower curves with radii R_1 and R_2 are calculated using the circular segment expression:

$$A_{segment} = R^2 \left(\tan \frac{\theta}{2} - \frac{\theta}{2} \right)$$

Net curved area:

$$A_{curve} = A_{large} - A_{small}$$

Total Area

$$A_{total} = A_{trap} + A_{curve}$$

Self-Weight Load

The barrier self-weight is computed as:

$$w = \frac{A_{total}}{10^6} \times \gamma_{RCC}$$

where:

$$\gamma_{RCC} = 25\,\mathrm{kN/m^3}$$

4.2.3 Implementation of Fig. 3: Details of High Containment Crash Barrier

Configuration

This configuration represents the high containment rigid crash barrier provided for locations requiring enhanced vehicle restraint. The geometry is obtained from:

`IRC5_2015.cl_109_6_3_shapes()`

with:

- Barrier type: Rigid
- Footpath: None
- Crash barrier type: High Containment

File Locations

`src/osdagbridge/core/bridge_components/super_structure/crash_barrier/geometry.py`
`high_containment_barrier_area()`

`src/osdagbridge/core/bridge_components/super_structure/crash_barrier/properties.py`
`high_containment_barrier_load()`

Area Calculation Method

The cross-section is divided into three trapezoidal regions and curved transition portions.

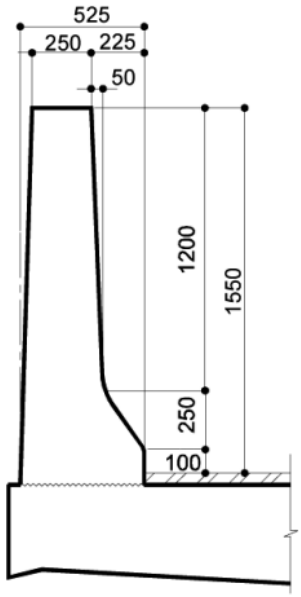

Fig. 3 Details of High Containment Crash Barrier

Figure 4.3: IRC 5:2015 Fig. 3: Details of High Containment Crash Barrier

Trapezoidal Areas

$$A_1 = \frac{1}{2}(W_{top} + W_{mid})H_{top}$$

$$A_2 = \frac{1}{2}(W_{mid} + W_{base})H_{mid}$$

$$A_3 = W_{base} \times H_{bottom}^{eff}$$

$$A_{trap} = A_1 + A_2 + A_3$$

Curved Portion The curved transitions with radii R_1 and R_2 are calculated using:

$$A_{segment} = R^2 \left(\tan \frac{\theta}{2} - \frac{\theta}{2} \right)$$

$$A_{curve} = A_{large} - A_{small}$$

Total Area

$$A_{total} = A_{trap} + A_{curve}$$

Self-Weight Load

The equivalent line load is calculated as:

$$w = \frac{A_{total}}{10^6} \times \gamma_{RCC}$$

where:

$$\gamma_{RCC} = 25\,\mathrm{kN/m^3}$$

The load computation is implemented in:

`high_containment_barrier_load()`

4.2.4 Implementation of Fig. 4: Details of Metallic Crash Barrier

IRC Reference: Fig. 4 – Details of Metallic Crash Barrier (a) With single W-beam (b) With double W-beam

Configuration

This configuration represents the semi-rigid edge metallic crash barrier system. Two cases are supported:

- Single W-beam
- Double W-beam

The geometric parameters are obtained from:

`IRC5_2015.cl_109_6_3_shapes()`

with barrier type = Semi-rigid.

File Locations

`src/osdagbridge/core/bridge_components/super_structure/crash_barrier/geometry.py`

`metallic_edge_barrier_area(barrier_type)`

`src/osdagbridge/core/bridge_components/super_structure/crash_barrier/properties.py`

`metallic_edge_barrier_load()`

IRC Figure

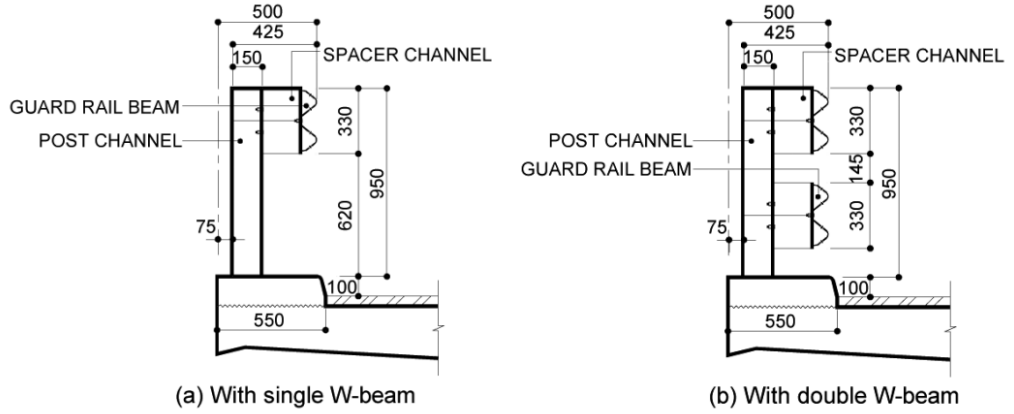

Figure 4.4: IRC 5:2015 Fig. 4: Details of Metallic Crash Barrier (Single and Double W-Beam)

Area Calculation

The total cross-sectional area is computed as the sum of the following components:

1. RCC Kerb Area The base kerb is calculated as a trapezoid:

$$A_{kerb} = \frac{1}{2}(B_{top} + B_{bottom}) \times H$$

2. Steel Post and Spacer Area The equivalent steel area per metre length is obtained using:

$$A_{post} = \frac{A_{section} \times H_{post}}{spacing}$$

including the spacer contribution.

3. W-Beam Area

$$A_{beam} = t \times L_{developed} \times n$$

where:

- t = beam thickness
- $L_{developed}$ = developed length
- n = number of beams (1 for single, 2 for double)

Total areas returned:

$$A_{steel} = A_{post} + A_{beam}$$

The function returns:

- Steel area (mm^2/m)
- RCC kerb area (mm^2/m)

Self-Weight Load

Loads are computed separately for steel and RCC:

$$w_{steel} = \frac{A_{steel}}{10^6} \times 78$$

$$w_{RCC} = \frac{A_{kerb}}{10^6} \times 25$$

Total load:

$$w_{total} = w_{steel} + w_{RCC}$$

The load function returns:

- Steel load (kN/m)
- RCC kerb load (kN/m)
- Total load (kN/m)

4.2.5 Implementation of Fig. 5: Median Details

IRC Reference: Fig. 5 – Median Details (a) With raised kerb (b) With RCC crash barrier (c) With metallic crash barrier (single or double W-beam)

File Locations

`src/osdagbridge/core/bridge_components/super_structure/median/geometry.py`

`src/osdagbridge/core/bridge_components/super_structure/median/properties.py`

IRC Figure

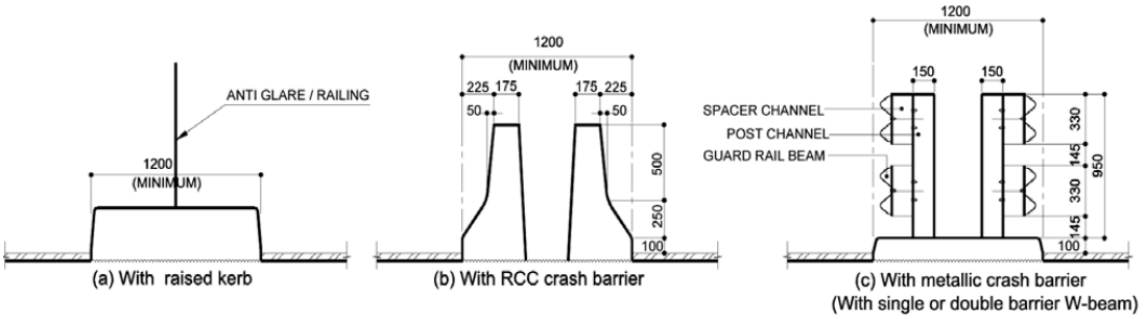

Fig. 5 Median Details

Figure 4.5: IRC 5:2015 Fig. 5: Median Details (Raised Kerb, RCC Barrier, and Metallic Barrier)

Fig. 5(a): Raised Kerb

The median kerb is modeled as a trapezoidal section.

$$A_{kerb} = \frac{1}{2}(B_{top} + B_{bottom}) \times H$$

Load calculation:

$$w = \frac{A_{kerb}}{10^6} \times 25$$

The functions return:

- Kerb area (mm^2/m)
- RCC load (kN/m)

Fig. 5(b): Median with RCC Crash Barrier

The RCC barrier profile is divided into three trapezoidal regions with curved transition correction using the circular segment formula:

$$A_{segment} = R^2 \left(\tan \frac{\theta}{2} - \frac{\theta}{2} \right)$$

Since the median has two barriers:

$$A_{total} = 2 \times A_{one\_side}$$

Load calculation:

$$w = \frac{A_{total}}{10^6} \times 25$$

The functions return:

- Area for one side and total area
- Total RCC load for both barriers (kN/m)

Fig. 5(c): Median with Metallic Crash Barrier

The total area consists of:

- RCC kerb area (trapezoidal)
- Steel post and spacer area
- W-beam area (single or double)

Since the median has two sides:

$$A_{steel,total} = 2 \times A_{steel}$$

$$A_{RCC,total} = 2 \times A_{kerb}$$

Load calculations:

$$w_{steel} = \frac{A_{steel,total}}{10^6} \times 78$$

$$w_{RCC} = \frac{A_{RCC,total}}{10^6} \times 25$$

$$w_{total} = w_{steel} + w_{RCC}$$

The functions return:

- Steel load (kN/m)
- RCC kerb load (kN/m)
- Total load (kN/m)

Railing Module

The configurations corresponding to Fig. 1(a) and Fig. 1(b) are also implemented within the railing module to support independent railing design and property calculations. Since the railing geometry is common to the crash barrier configuration with footpath, the same IRC provisions are reused.

File location:

```
src/osdagbridge/core/bridge_components/super_structure/railing/
    geometry.py
    properties.py
```

These modules provide geometric parameters and self-weight calculations for RCC and steel railings consistent with IRC 5:2015.

4.2.6 Integration Approach

The implementation follows a modular workflow consistent with the OsdagBridge architecture.

The overall flow of information is:

IRC5_2015 clauses → Keyfile constants → Geometry modules → Load (properties) modules

- **IRC5_2015.py** provides IRC Clause 109.6.3 geometric parameters for different barrier configurations.

- **keyfile.py** contains common constants such as barrier types, footpath options, railing types, and material properties.
- **Geometry modules** compute cross-sectional areas based on IRC dimensions.
- **Properties modules** convert the computed area into equivalent line loads (kN/m).

Each geometry function returns a dictionary containing the required geometric parameters or cross-sectional areas. The corresponding load functions use these values to compute RCC and/or steel self-weight, which can be directly used in higher-level structural analysis modules.

4.2.7 Directory Structure and File Locations

The modules developed for this task are organized within the OsdagBridge repository as follows:

```
src/osdagbridge/core/bridge_components/super_structure/

crash_barrier/
    geometry.py
    properties.py

median/
    geometry.py
    properties.py

railing/
    geometry.py
    properties.py

src/osdagbridge/core/irc/
    irc5_2015.py

src/osdagbridge/core/utils/codes/
```

`keyfile.py`

These modules are called by higher-level bridge components to generate geometry and load parameters based on user-selected configurations.

4.2.8 Python Code

Listing 4.1: Rigid Crash Barrier with Railing Area Calculation (IRC Fig. 1)

```
1
2  def rigid_barrier_with_railing_area(railing):
3      """
4      IRC Fig. 1(a) & 1(b)
5      Computes geometric area of rigid crash barrier with railing.
6      """
7
8      import math
9
10     if railing.lower() == "rcc":
11         railing_key = KEY_RAILING_TYPE[0]
12         barrier_name = "Rigid Barrier with RCC Railing"
13     elif railing.lower() == "steel":
14         railing_key = KEY_RAILING_TYPE[1]
15         barrier_name = "Rigid Barrier with Steel Railing"
16     else:
17         raise ValueError("railing must be RCC or Steel")
18
19     geom = IRC5_2015.cl_109_6_3_shapes(
20         barrier_type=KEY_CRASH_BARRIER_TYPE[2],
21         footpath=KEY_FOOTPATH[1],
22         railing_type=railing_key,
23         design_dict={},
24         crash_barrier_type=None
25     )
26
27     W   = geom["crash_barrier_width"]
28     T   = geom["crash_barrier_top_notch"]
29     M   = geom["crash_barrier_middle_length"]
30     B   = geom["crash_barrier_base_notch"]
```

```
    R1  = geom["crash_barrier_radius1"]
    R2  = geom["crash_barrier_radius2"]
    wearing = geom["wearing_course_thickness"]

    base_effective = B + wearing

    # Trapezoidal areas
    H1, H2, H3 = 550, 250, base_effective

    L1 = T + 50
    L2 = 225

    A1 = 0.5 * (T + L1) * H1
    A2 = 0.5 * (L1 + L2) * H2
    A3 = 0.5 * (L2 + W) * H3

    A_trapezoids = A1 + A2 + A3

    # Curved segments
    def segment_area(R, theta):
        return (R**2) * (math.tan(theta/2) - theta/2)

    x = math.atan(50/550)
    y = math.atan(175/250)

    theta_big = y - x

    A_big = segment_area(R2, theta_big)
    A_small = segment_area(R1, y)

    A_curve = A_big - A_small

    total_area = A_trapezoids + A_curve

    return {
        "type": barrier_name,
        "barrier_area": round(total_area, 3)
    }
```

4.2.9 Explanation of the Code

- IRC geometric dimensions are retrieved using the Clause 109.6.3 shape function.
- The cross-section is divided into trapezoidal regions to represent the sloping faces.
- Curved portions are calculated using circular segment equations based on tangent geometry.
- The total area is obtained by summing trapezoidal and curved areas.
- The function returns the barrier type and cross-sectional area, which is later used for load computation.

4.3 Task 2: Documentation

The developed modules were written following the standard OsdagBridge coding conventions to ensure consistency and maintainability.

Each function is documented with appropriate docstrings describing:

- The corresponding IRC figure or clause
- Input parameters and configuration options
- Output values such as cross-sectional area or load

The functions return results in dictionary format, enabling seamless use in higher-level analysis, design, and reporting modules within the OsdagBridge workflow.

The implementation is modular, allowing easy extension for additional IRC provisions or future enhancements.

Chapter 5

Implementation of IRC 6 Special Vehicle Load Model for OsdagBridge

5.1 Task 3: Problem Statement

The objective of this task was to implement the Special Vehicle load model specified in IRC 6:2017 (Clause 204.5 and 204.5.1) within the OsdagBridge framework.

The special vehicle consists of a prime mover connected to a 20-axle hydraulic trailer used for transporting heavy indivisible loads. The implementation required generation of axle loads and their spatial arrangement so that the vehicle configuration could be directly used for structural analysis and load evaluation.

The developed module computes:

- Axle load magnitudes as per IRC provisions
- Longitudinal positions of each axle
- Transverse wheel locations
- Total vehicle load
- A structured axle-wise load table for reporting and verification

The implementation supports direct integration into bridge load analysis workflows within OsdagBridge.

5.2 Task 3: Tasks Done

The Special Vehicle model was implemented based on IRC 6:2017 Clause 204.5 / 204.5.1. The configuration represents a prime mover followed by a hydraulic trailer consisting of multiple axles.

5.2.1 Vehicle Configuration

The following parameters were implemented:

- One steering axle (6 t)
- Two bogie axles (9.5 t each)
- Twenty trailer axles (18 t each)
- Longitudinal axle spacing as specified in IRC
- Transverse wheel spacing of $\pm$ 0.9 m

5.2.2 Load Generation

The implementation performs the following steps:

- Defines axle loads in tonnes and converts them to kN
- Generates cumulative longitudinal positions for all axles
- Assigns transverse wheel locations
- Computes total vehicle load
- Creates an axle-wise mapping of position and load
- Generates a formatted table for reporting

The function returns structured data that can be directly used for load placement in bridge analysis.

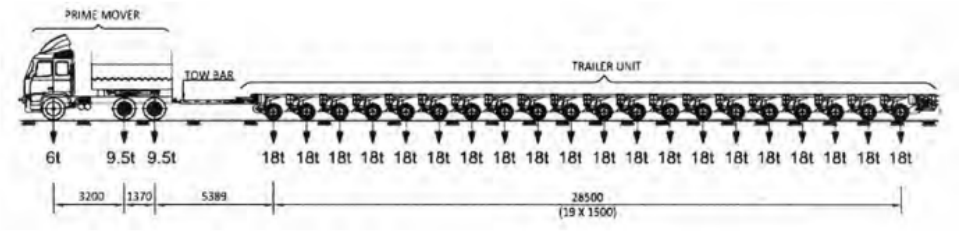

Fig. 6: Typical Axle Arrangement for Special Vehicle

Figure 5.1: IRC 6:2017 Special Vehicle Configuration (Clause 204.5.1)

5.2.3 Integration Approach

The Special Vehicle function is implemented as a static method within the IRC6 module:

`IRC6_2017.cl_204_5_1_special_vehicle()`

The workflow is as follows:

IRC6 Clause → Keyfile constants → Vehicle load generation → Structured output for analysis

The function returns a dictionary containing:

- Longitudinal axle positions (x)
- Transverse wheel locations (z)
- Wheel loads (kN)
- Total vehicle load (tonne and kN)
- Axle-wise load mapping
- Formatted axle load table

This structure allows seamless integration with bridge load placement, analysis modules, and reporting utilities.

5.2.4 Directory Structure and File Location

The implementation is located within the OsdagBridge repository as follows:

```
src/osdagbridge/core/utils/codes/
    irc6_2017.py
    keyfile.py
```

The function is part of the IRC6 module, which contains load models and provisions defined in IRC 6:2017.

5.2.5 Python Code

Listing 5.1: IRC 6 Special Vehicle Load Generation (Clause 204.5.1)

```
1
2  @staticmethod
3      def cl_204_5_1_special_vehicle():
4
5          # IRC:6-2017 Clause 204.5 / 204.5.1
6          # Special Vehicle (Prime mover + 20 axle hydraulic trailer)
7
8          # Longitudinal Spacing
9          dist12 = 3.200 * m
10         dist23 = 1.370 * m
11         dist34 = 5.389 * m
12         trailer_spacing = 1.500 * m
13
14         #  Axle Loads (tonnes)
15         axle_loads_tonne = (
16             [6.0] +            # 1 steering axle
17             [9.5, 9.5] +       # 2 bogie axles
18             [18.0] * 20        # 20 trailer axles
19         )
20
21         # Convert tonne      kN
22         wheel_loads = [ax * g for ax in axle_loads_tonne]
23
24         #  Longitudinal Positions
```

```
25          load_positions_x = [0.0]
26          load_positions_x.append(load_positions_x[-1] + dist12)
27          load_positions_x.append(load_positions_x[-1] + dist23)
28          load_positions_x.append(load_positions_x[-1] + dist34)
29
30          # 19 spacings     gives 20 trailer axle positions total
31          for _ in range(19):
32              load_positions_x.append(load_positions_x[-1] +
                trailer_spacing)
33
34          #  Transverse Positions
35          load_positions_z = [-0.9, 0.9]
36
37          #  Totals
38          total_load_tonne = sum(axle_loads_tonne)
39          total_load_kN = sum(wheel_loads)
40
41          #  Axle     Position Explicit Mapping
42
43          axle_load_map = []
44          location_table = []
45          header = f"{'Axle':>4} | {'X (m)':>7} | {'Load (t)':>8} | {'
                Load (kN)':>9}"
46          location_table.append(header)
47          location_table.append("-" * len(header))
48
49          for i in range(len(axle_loads_tonne)):
50              axle = {
51                  'axle_no': i + 1,
52                  'x': round(load_positions_x[i], 3),
53                  'load_tonne': round(axle_loads_tonne[i], 2),
54                  'load_kN': round(wheel_loads[i], 2)
55              }
56              axle_load_map.append(axle)
57
58              # printable line
59              location_table.append(
60                  f"{axle['axle_no']:>4} | {axle['x']:>7.3f} | {axle['
                        load_tonne']:>8.2f} | {axle['load_kN']:>9.2f}"
```

```
            )

        pretty_table = "\n".join(location_table)


        assert len(load_positions_x) == len(axle_loads_tonne), \
            "Axle count and position count mismatch"

        return {
            'x': load_positions_x,
            'z': load_positions_z,
            'wheel_loads': wheel_loads,
            'total_load_kN': round(total_load_kN, 3),
            'total_load_tonne': round(total_load_tonne, 3),
            'axle_load_map': axle_load_map,
            'pretty_axle_table': pretty_table
        }
```

5.2.6 Explanation of the Code

- Axle loads are defined according to IRC 6:2017, consisting of a steering axle, bogie axles, and 20 trailer axles.
- The loads are converted from tonne to kN using the gravitational constant obtained from the keyfile.
- Longitudinal axle positions are generated using cumulative spacing based on the specified distances between axles.
- Trailer axle locations are created using a loop with uniform spacing.
- Transverse wheel positions are assigned at ± 0.9 m from the vehicle centerline.
- The function computes the total vehicle load and creates an axle-wise mapping of position and load.
- The results are returned in dictionary format, enabling direct use in bridge load placement, analysis, and reporting within the OsdagBridge workflow.

5.3 Task 3: Documentation

The function is documented using descriptive docstrings specifying the corresponding IRC clause, vehicle configuration, input assumptions, and output parameters.

The returned data structure is designed in dictionary format to enable direct use in higher-level bridge analysis and reporting modules. The modular implementation allows easy extension for additional vehicle configurations or future IRC revisions.

Chapter 6

Implementation of IRC 22: Limit State Design Provisions for Composite Bridges in OsdagBridge

6.1 Task 4: Problem Statement

IRC 22 provides the Standard Specifications and Code of Practice for Composite Construction in Road Bridges based on the Limit State Design philosophy. The revised version of IRC 22 was developed to ensure compatibility with IRC 24 for steel bridges and IRC 112 for concrete bridges, thereby providing a unified framework for the design of composite bridge structures.

Composite bridges combine structural steel girders with reinforced concrete decks to achieve efficient structural performance and economy. The design involves evaluation of material properties, section classification, strength and serviceability checks under various limit states.

The objective of this task was to implement the provisions of IRC 22 within the OsdagBridge framework. The developed module translates the relevant clauses into computational functions that:

- Define material and section properties for composite members
- Implement limit state design provisions for strength and serviceability
- Perform classification and capacity checks as per IRC requirements

- Provide structured outputs for integration with higher-level bridge analysis and design modules

The implementation enables automated evaluation of composite bridge components in accordance with IRC 22 within the OsdagBridge environment.

6.2 Task 4: Tasks Done

6.2.1 Clause 601.4 – Material Partial Safety Factors

Clause 601.4 specifies the partial safety factors for different materials under Ultimate Limit State (ULS) and Serviceability Limit State (SLS). These factors are required for the design of structural steel, reinforcement, concrete, welds, and fasteners.

The values from Table 1 of IRC:22-2014 were implemented in the code as a dictionary structure for easy access by other design modules.

Table 1 Material Safety Factors (γ_m)

	Partial Safety Factor γ_m	
	Ultimate Limit	**Serviceability Limit**
Structural Steel against Yield Stress	1.10	1.00
Structural Steel against Ultimate stress	1.25	1.00
Steel Reinforcement (γ_s) against Yield Stress	1.15	1.00
Shear Connectors against Yield Stress	1.25	1.00
Bolts & Rivets for Shop & Site Fabrication against Yield Stress	1.25	1.00
Welds for Shop Fabrication	1.25	1.00
Welds for Site Fabrication	1.50	1.00
Concrete (γ_c) For Basic and Seismic Combinations	1.50	1.00
Concrete (γ_c) For Accidental Combinations	1.20	1.00

Note: Partial safety factors are not only given for f_y and f_{ck} but also for f_u

Figure 6.1: IRC 22:2014 Table 1 – Material Partial Safety Factors

The function `cl_601_4_material_safety_factors()` returns the partial safety factors for different materials for both ULS and SLS conditions.

]Clause 602 –

6.2.2 Clause 602 – Material Properties and Section Classification

Clause 602 of IRC:22-2014 specifies the material properties to be used for composite bridge design. Annex III provides standard values for structural steel, concrete, and reinforcement steel to be adopted in analysis and design.

The provisions of Annex III were implemented as modular functions to provide material properties for different design scenarios. The implementation covers:

- III.1 Structural Steel Properties
- III.2 Concrete Properties
- III.3 Reinforcement Steel Properties

Annex III – Material Properties

III.1 Structural Steel Properties

The following properties were implemented for structural steel, applicable to all grades:

- Young's Modulus, $E = 2.0 \times 10^5$ MPa
- Shear Modulus, $G = 0.77 \times 10^5$ MPa
- Poisson's Ratio, $\nu = 0.30$
- Coefficient of Thermal Expansion, $\alpha = 1.17 \times 10^{-5}/°\text{C}$

The function returns these values in dictionary form for use in stiffness and thermal load calculations.

Listing 6.1: IRC 22 Clause 602 Annex III – Structural Steel Properties

```
1
2  @staticmethod
3  def cl_602_annexIII_structural_steel_properties():
4      """
5      IRC:22-2014 Clause 602 | Annex-III | III.1
6      Returns fundamental structural steel properties.
7      """
8
9      properties = {
10         "E_MPa": E_STEEL_MPA,
11         "E_GPa": E_STEEL_MPA / 1000.0,
12         "G_MPa": G_STEEL_MPA,
13         "nu": POISSON_RATIO_STEEL,
14         "coefficient_of_thermal_expansion": COTE_STEEL_PER_C
15     }
```

```
16
17      return properties
```

Explanation The function returns the standard material constants for structural steel as specified in IRC:22 Annex III. These values are used in stiffness calculations, thermal load evaluation, and other structural analysis procedures within the OsdagBridge framework.

III.2 Concrete Properties

Material properties for concrete grades from M15 to M90 were implemented based on Annex III. The function returns characteristic compressive strength (f_{ck}), design compressive strength, tensile strength (f_{ctm}), and modulus of elasticity (E_c).

Aggregate-dependent modification factors were incorporated to adjust the modulus of elasticity. For structural applications, the minimum concrete grade was restricted to M25 as specified in IRC:22.

Table - III - 1: Properties of Concrete

Strength Class of Concrete

Strength Class	M 15	M 20	M 25	M 30	M 35	M 40	M 45	M 50	M 55	M 60	M 65	M 70	M 75	M 80	M 85	M 90
$(f_{ck})_{cu}$ (MPa)	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
$(f_{ck})_{cy}$ (MPa)	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72
f_{ctm} (MPa)	1.6	1.9	2.2	2.5	2.8	3.0	3.3	3.5	3.7	4.0	4.4	4.5	4.7	4.8	4.9	5.0
E_{cm} (GPa)	27	29	30	31	32	33	34	35	36	37	38	38	39	40	40	41

Note:-

$(f_{ck})_{cu}$ ---- characteristic compressive (cube) strength of concrete

$(f_{ck})_{cy}$ ---- characteristic compressive (cylinder) strength of concrete, given by 0.8 times 28 days cube crushing strength of concrete

f_{ctm} ---- mean tensile strength of concrete

E_{cm} ---- Secant Modulus of elasticity of concrete.

The values of E_{cm} given above are for quartzite/granite aggregates. They should be multiplied by the following factors as given below:
Limestone = 0.9; Sandstone = 0.7; Basalt = 1.2

The values have been taken from Table - 6.5 of IRC:112-2011

Figure 6.2: IRC 22:2014 Annex III – Properties of Concrete

III.3 Reinforcement Steel Properties

Reinforcement properties were implemented based on IS:1786-2008. The function stores characteristic yield strength, minimum ultimate strength, and elongation for different grades.

In addition, the IRC requirement for ductility was incorporated:

$$f_u \geq \max\left[(1 + p/100)f_y,\ f_{u,\min}\right]$$

The function computes the required ultimate strength and the corresponding strength ratio (f_u/f_y), which are used in limit state design checks.

Section Classification (Reference to IS 800:2007)

Clause 602 – Outstanding Compression Flange Classification

The outstanding element of a compression flange is classified based on the width-to-thickness ratio. The classification determines whether the flange behaves as Plastic, Compact, Semi-Compact, or Slender.

The limiting values and classification logic are directly obtained from the `IS800_2007` (Ref: IS 800:2007 Table 2(i)) module available in the Osdag core directory. The function returns the section class along with the calculated width-to-thickness ratio.

Clause 602 – Web Classification of I/H/Box Sections

The web of I, H, or box sections is classified based on the depth-to-thickness ratio to determine whether the section is Plastic, Compact, Semi-Compact, or Slender. The classification depends on the stress distribution condition and the position of the neutral axis.

The implementation directly calls the function `Table2_iii` from the `IS800_2007` module available in the Osdag core directory. The limiting width–thickness ratios are obtained from:

- Table 2(iii) – Web classification limits
- Table 1 / Table 3 – Material factor ($\epsilon = \sqrt{250/f_y}$) and stress parameters

The function returns the appropriate section classification based on the given web depth, thickness, and material strength.

6.2.3 Clause 603 - Steel Cross-Section Classification

Clause 603 requires classification of steel cross-sections based on width–thickness limits to determine their behaviour in limit state design. The classification categories include Plastic, Compact, Semi-Compact, and Slender sections.

The implementation evaluates the web slenderness ratio (d/t_w) and compares it with limiting values based on:

$$\epsilon = \sqrt{\frac{250}{f_y}}$$

The limits for different classes are computed as per IS 800 provisions.

The classification limits and reference checks are obtained from the `IS800_2007` module available in the Osdag core directory (outside OsdagBridge). The function returns the calculated slenderness ratio, limiting values, and the resulting section class.

6.2.4 Clause 603.3.1 – Positive Moment Capacity of Composite Section

Clause 603.3.1 specifies the calculation of positive bending moment resistance of composite beams at the ultimate limit state. The provisions of Annex I (I.1) were implemented assuming full shear interaction between the concrete slab and steel beam.

The effective width of the concrete slab (b_{eff}) is first evaluated. For non-compact sections, the effective width is restricted based on compactness limits of the steel flange and web. These compactness checks are performed using IS800:2007 Table 2 provisions through the `IS800_2007` module available in the Osdag core directory.

The design compressive force in concrete is calculated as:

$$C = 0.36\, f_{ck}\, b_{eff}\, d_c$$

The tensile force in steel is:

$$T = A_f\, f_y$$

The governing force is taken as:

$$F = \min(C, T)$$

The ultimate positive moment capacity is then obtained as:

$$M_p = F \times d_c$$

The function returns the effective width used, the moment capacity in kN-m, and the governing force (steel or concrete).

6.2.5 Clause 603.3.3 – Design of Structure

This clause covers the strength verification of composite steel sections during the construction and service stages. The implemented functions evaluate flexural buckling resistance, shear capacity, and the interaction between bending and shear, as per IRC:22-2014 with references to IS 800:2007 where applicable.

Clause 603.3.3.1 – Buckling Resistance Moment (Construction Stage)

This function evaluates the lateral torsional buckling resistance moment of steel sections during the construction stage as per Annex I (I.5) of IRC:22-2014. The procedure follows the design methodology of IS 800:2007 (Clause 8.2.1.2).

The implementation performs the following steps:

- Calculates the plastic design moment capacity (M_{pl}) using IS 800 provisions.
- Determines the section factor β_b:
  - $\beta_b = 1.0$ for plastic and compact sections
  - $\beta_b = Z_e/Z_p$ for semi-compact sections
- Assigns imperfection factor α_{LT}:
  - 0.21 for rolled sections
  - 0.49 for welded sections
- Computes the elastic critical moment M_{cr} using section properties (I_y, I_t, I_w) and unbraced length (L_{LT}).

- Evaluates non-dimensional slenderness λ_{LT} and reduction factor χ_{LT}.
- Determines the design buckling resistance moment:

$$M_b = \chi_{LT}\, M_{pl}$$

The function returns the buckling capacity along with intermediate parameters such as λ_{LT}, χ_{LT}, and M_{cr} for design transparency.

Clause 603.3.3.2 – Vertical Shear

This clause evaluates the shear resistance of steel sections in composite construction. The implementation covers plastic shear capacity and shear buckling resistance as per IRC:22-2014 with detailed procedures referenced from IS 800:2007.

Clause 603.3.3.2 (1) – Plastic Shear Resistance The plastic shear capacity is calculated based on the effective shear area of the section as per IS 800:2007 Clause 8.4.1.

The nominal shear strength is given by:

$$V_n = \frac{A_v f_{yw}}{\sqrt{3}}$$

$$V_d = \frac{V_n}{\gamma_{m0}}$$

where $\gamma_{m0} = 1.10$.

The shear area A_v is determined based on section configuration:

- Rolled I-section (major axis): $A_v = h\, t_w$
- Welded/plate girder: $A_v = d\, t_w$
- Minor axis bending: $A_v = 2 b_f t_f$

The function returns the shear area, nominal shear capacity, and design shear resistance.

Clause 603.3.3.2 (2) – Shear Buckling Resistance When the web is slender, shear resistance is governed by buckling behaviour. Two methods are implemented as per IS 800:2007 Clause 8.4.2.2.

(a) Simple Post-Critical Method This method is applicable when transverse stiffeners are provided only at supports or at specified spacing.

The procedure includes:

- Determination of shear buckling coefficient K_v from IS 800.
- Calculation of elastic shear buckling stress:

$$\tau_{cr} = \frac{K_v \pi^2 E}{12(1-\mu^2)} \left(\frac{t_w}{d}\right)^2$$

- Evaluation of web slenderness:

$$\lambda_w = \sqrt{\frac{f_{yw}}{\sqrt{3}\tau_{cr}}}$$

- Post-critical shear stress:

$$\tau_b = \begin{cases} \frac{f_{yw}}{\sqrt{3}}, & \lambda_w \leq 1.2 \\ \frac{1.2}{\lambda_w}\frac{f_{yw}}{\sqrt{3}}, & \lambda_w > 1.2 \end{cases}$$

The design shear resistance is obtained as:

$$V_{rd} = A_v \tau_b$$

(b) Tension Field Method This method is used when transverse stiffeners are provided at intermediate locations and post-buckling tension field action is mobilized.

The implementation directly utilizes the tension field formulation from IS 800:2007 Clause 8.4.2.2(b), which evaluates:

- Inclination of tension field (ϕ)
- Flange moment resistance (M_{fr})
- Effective tension field width
- Shear stress components
- Final tension field shear resistance (V_{tf})

The function returns intermediate parameters along with the design shear capacity for detailed design verification.

Listing 6.2: Plastic Shear Resistance – IRC 22 Clause 603.3.3.2

```
@staticmethod
def cl_603_3_3_2_plastic_shear_resistance(
    section_type,
    fyw,
    fabrication="rolled",
    h=None,
    d=None,
    tw=None,
    bf=None,
    tf=None
):
    gamma_m0 = 1.10

    if section_type == "i_major":
        if fabrication == "rolled":
            Av = h * tw
        else:
            Av = d * tw

    elif section_type == "i_minor":
        Av = 2 * bf * tf

    Vn = (Av * fyw) / math.sqrt(3)
    Vd = Vn / gamma_m0

    return {
        "Av_mm2": Av,
        "Vn_kN": Vn / 1000,
        "Vd_kN": Vd / 1000
    }
```

Clause 603.3.3.3 – Reduction in Bending Resistance under High Shear

Clause 603.3.3.3 – Reduction in Bending Resistance under High Shear

When the factored shear force is high, the bending resistance of the section is reduced as per IRC:22-2014. This provision is based on IS 800:2007 Clause 9.2.2(a) and (b).

If the applied shear force satisfies:

$$V \leq 0.6V_d$$

no reduction in bending resistance is required.

For higher shear levels:

$$V > 0.6V_d$$

the reduced bending resistance is calculated using:

$$\beta = \left(\frac{2V}{V_d} - 1\right)^2 \leq 1.0$$

$$M_{dv} = M_d - \beta(M_d - M_{fd})$$

where:

- M_d = design bending resistance
- M_{fd} = plastic bending strength excluding shear area
- V = factored shear force
- V_d = design shear strength

The implemented function evaluates the reduction factor (β), checks whether reduction is required, and returns the reduced bending capacity.

6.2.6 Clause 604 – Design for Serviceability Limit State

Clause 604.3 – Stresses and Deflection

IRC:22-2014 – Clause 604.3 (Modular Ratio)

Clause 604.3 specifies the modular ratio to be used for transformed section analysis in composite bridges.

The modular ratio is defined as:

$$m = \frac{E_s}{E_c}$$

where:

- E_s = modulus of elasticity of steel (2.0×10^5 MPa)
- E_c = modulus of elasticity of concrete (from Annex III)

The following cases were implemented:

- **Before 28 days:**

$$m = \frac{E_s}{E_{ci}}$$

- **Short-term loading:**

$$m = \frac{E_s}{E_c} \geq 7.5$$

- **Long-term loading (including creep):**

$$m = \frac{E_s}{K_c E_c} \geq 15.0$$

  where K_c is the creep factor (taken as 0.5).

The concrete modulus is obtained from the Annex III concrete property table implemented under Clause 602.

Clause 604.3.1 – Service Stress Limits

This clause specifies permissible service stresses for concrete, reinforcement, and structural steel.

Concrete:

$$\sigma_{c,max} \leq 0.48 f_{ck}$$

(as per IRC:112)

Reinforcement:

$$\sigma_{s,max} \leq 0.8 f_y$$

Structural Steel:

Equivalent stress is checked using combined stress criteria considering bending, axial and shear effects.

These limits are used to verify that stresses remain within acceptable levels under service load conditions.

Clause 604.3.2 – Deflection Limits

Serviceability deflection limits specified in IRC:22 were documented for design checks.

- Total deflection due to dead load, live load and impact:

$$\leq \frac{L}{600}$$

- Deflection due to live load and impact:

$$\leq \frac{L}{800}$$

- For cantilever portions:
  - Total deflection:

$$\leq \frac{L}{300}$$

  - Live load deflection:

$$\leq \frac{L}{400}$$

Clause 604.4 – Control of Cracking in Concrete

IRC:22-2014 – Clause 604.4
(Reference: IRC:112-2011 Clause 12.3.3)

Clause 604.4 specifies the minimum reinforcement required in the tensile zone of concrete to control cracking under service load conditions. The provisions of IRC:112-2011 Clause 12.3.3 were implemented for this purpose.

The minimum area of reinforcement is calculated using:

$$A_{s,min} = \frac{k_c \, k \, f_{ct,eff} \, A_{ct}}{\sigma_s}$$

where:

- A_{ct} = area of concrete in tension zone = $b_{eff} \times t$
- $f_{ct,eff}$ = effective tensile strength of concrete (taken as mean tensile strength f_{ctm}, assuming cracking after 28 days)
- σ_s = steel stress (taken as f_y unless specified)
- k_c = coefficient accounting for stress distribution (taken ≥ 0.5)
- k = restraint coefficient based on flange/web width

The restraint coefficient k is evaluated as:

- $k = 1.0$ for width ≤ 300 mm
- $k = 0.65$ for width > 800 mm
- Linear interpolation for intermediate widths

The function also compares the provided reinforcement with the required minimum and reports whether the section satisfies crack control requirements.

6.2.7 Clause 605 – Design for Fatigue Limit

Clause 605.2 – Fatigue Design

Clause 605.2 specifies fatigue design requirements for steel members subjected to repeated loading.

Thickness Correction Factor

For welded sections with plate thickness greater than 25 mm, a fatigue strength reduction factor is applied:

$$\mu_r = \left(\frac{25}{t_p}\right)^{0.25} \leq 1.0$$

where t_p is the thickness of the thicker plate. For rolled sections or welded plates with $t_p \leq 25$ mm, $\mu_r = 1.0$.

Low Fatigue Exemption

Fatigue assessment is not required if either of the following conditions is satisfied:

$$f < \frac{27}{\gamma_{mft}}$$

or

$$N_{sc} < 5 \times 10^6 \left(\frac{27/\gamma_{mft}}{\gamma_{ff}\, f} \right)^3$$

where:

- f = fatigue stress range (MPa)
- N_{sc} = number of stress cycles
- γ_{mft} = partial safety factor for fatigue strength (Table 3)
- γ_{ff} = partial safety factor for fatigue actions

The implemented function evaluates the thickness reduction factor, checks the exemption criteria, and determines whether fatigue design is required.

Clause 605.3 – Fatigue Strength

Clause 605.3 specifies the design fatigue strength of steel members subjected to repeated stress cycles.

The design fatigue stress range depends on the number of stress cycles (N_{sc}) and the type of stress.

Normal Stress Range

For normal stress range:

$$f_f = f_{fn} \left(\frac{5 \times 10^6}{N_{sc}} \right)^{1/3} \quad \text{for } N_{sc} \leq 5 \times 10^6$$

$$f_f = f_{fn} \left(\frac{5 \times 10^6}{N_{sc}} \right)^{1/5} \quad \text{for } 5 \times 10^6 < N_{sc} \leq 10^8$$

Shear Stress Range

$$\tau_f = \tau_{fn} \left(\frac{5 \times 10^6}{N_{sc}} \right)^{1/5}$$

where:

- N_{sc} = number of stress cycles
- f_{fn} = normal fatigue strength at 5×10^6 cycles
- τ_{fn} = shear fatigue strength at 5×10^6 cycles

Default values used in the implementation:

- $f_{fn} = 118$ MPa for rolled sections
- $f_{fn} = 92$ MPa for welded sections
- $\tau_{fn} = 59$ MPa

The function computes the allowable normal and shear fatigue stress ranges based on the specified number of cycles.

Clause 605.4 – Fatigue Assessment

Clause 605.4 specifies the procedure for fatigue assessment by comparing the applied stress ranges with the design fatigue strength of the member.

The design fatigue strength for the specified number of cycles is obtained from Clause 605.3 and modified using the thickness correction factor from Clause 605.2.

Design Fatigue Strength

The design fatigue strengths for normal and shear stresses are given by:

$$f_{fd} = \frac{\mu_r \, f_f}{\gamma_{mft}}$$

$$\tau_{fd} = \frac{\mu_r \, \tau_f}{\gamma_{mft}}$$

where:

- f_f = fatigue strength for normal stress (Clause 605.3)

- τ_f = fatigue strength for shear (Clause 605.3)
- μ_r = thickness correction factor (Clause 605.2)
- γ_{mft} = partial safety factor for fatigue strength

Stress Range Check

The structure satisfies fatigue requirements if:

$$f_{\text{range}} \leq f_{fd}$$

$$\tau_{\text{range}} \leq \tau_{fd}$$

where:

- f_{range} = actual normal stress range
- τ_{range} = actual shear stress range

Maximum Stress Check

In addition to stress range verification, the maximum stresses shall remain within elastic limits:

$$\sigma_{\max} \leq f_y$$

$$\tau_{\max} \leq \tau_y$$

The shear yield limit is taken as:

$$\tau_y = 0.43\, f_y$$

based on the elastic shear capacity recommended in IRC:24.

The implementation evaluates:

- Fatigue stress range adequacy
- Maximum stress limits
- Overall fatigue safety status

6.2.8 Clause 606: Shear Connectors

This clause covers the design strength, fatigue strength, and detailing requirements of shear connectors used in composite construction. The implementation includes calculation of stud connector strength based on material properties and concrete grade as per IRC:22 provisions.

Clause 606.3.1: Ultimate Strength of Stud Connectors

The ultimate design strength of headed stud shear connectors was calculated using Eq. (6.1) of IRC:22. The resistance is evaluated based on both steel strength and concrete strength, and the lesser value governs.

The following steps are implemented:

- Concrete properties (f_{ck} and E_{cm}) are obtained from Annexure III based on the specified concrete grade.
- Cylinder strength is taken as $0.8f_{ck}$.
- The reduction factor α is calculated based on the stud slenderness ratio (h_s/d).
- Stud strength is evaluated considering:
  - Steel failure: $0.8f_u \cdot \pi d^2/4$
  - Concrete failure: $0.29\alpha d^2\sqrt{f_{ck}E_{cm}}$
- The design strength is obtained after applying the partial safety factor $\gamma_v = 1.25$.

Additional features implemented:

- Automatic selection of governing failure mode (steel or concrete)
- Optional comparison with Table 7 values
- Linear interpolation for intermediate concrete grades
- Warning for stud heights greater than 100 mm, as per IRC recommendations

The function returns:

- Design stud strength (Q_u) in kN
- Governing failure mode
- Reduction factor α
- Optional Table 7 reference value

Clause 606.3.2: Fatigue Strength of Stud Connectors

The fatigue strength of stud shear connectors was implemented as per IRC:22 Clause 606.3.2. The fatigue shear stress range is calculated using the relation:

$$\tau_f = \tau_{fn} \left(\frac{5 \times 10^6}{N_{sc}} \right)^{1/5}$$

where:

- τ_{fn} = nominal fatigue strength (67 MPa for stud connectors)
- N_{sc} = number of stress cycles

The implementation performs the following:

- Computes the fatigue shear stress range for the specified number of cycles.
- Provides the design fatigue strength of the stud connector in terms of stress.
- Optionally retrieves nominal fatigue strength (Q_r) from Table 8 for standard stud diameters (16, 20, 22, and 25 mm).
- Logarithmic interpolation is used for intermediate cycle values between tabulated limits.

The function returns:

- Fatigue shear stress range (τ_f) in MPa
- Optional nominal fatigue strength from Table 8 (in kN)

Clause 606.4: Spacing and Design of Shear Connectors

Clause 606.4.1: Longitudinal Shear and Stud Spacing The longitudinal shear force between the concrete slab and steel girder was evaluated as per IRC:22 Clause 606.4.1. The design procedure includes:

- Calculation of modular ratio:

$$n = \frac{E_s}{E_{cm}}$$

  where E_{cm} is obtained from Annexure III.

- Determination of effective concrete compression area:

$$A_{ec} = n\, b_{eff}\, t_{eff}$$

  where $t_{eff} = \min(x_u, t_{slab})$.

- Computation of longitudinal shear per unit length:

$$V_L = \frac{V\, A_{ec}\, Y}{I_c}$$

- Required stud spacing:

$$s = \frac{n_s\, Q_u}{V_L}$$

  where Q_u is obtained from Clause 606.3.1.

The function returns:

- Modular ratio
- Longitudinal shear per unit length
- Required stud spacing

Clause 606.4.1.1: Full Shear Connection For full shear connection, the total longitudinal force to be transferred between the steel girder and concrete slab was calculated based on Clause 606.4.1.1.

The governing longitudinal force is taken as:

$$H = \min(H_1, H_2)$$

where

$$H_1 = \frac{A_s f_{yk}}{\gamma_m}$$

$$H_2 = 0.36\, f_{ck}\, A_{ec}$$

and

$$A_{ec} = b_{eff} \times \min(x_u, t_{slab})$$

The required stud spacing over the shear span L is:

$$s = \frac{n_s\, Q_u}{H/L}$$

where:

- Q_u = stud capacity (Clause 606.3.1)
- L = distance between zero moment and maximum moment sections

The function returns:

- Effective concrete area
- Longitudinal forces H_1 and H_2
- Governing force
- Required stud spacing for full shear connection

Clause 606.4.2: Stud Spacing under Fatigue (Serviceability Limit State) Stud spacing under fatigue loading was determined as per IRC:22 Clause 606.4.2. This check is performed for the Serviceability Limit State considering the shear range due to live load and impact.

The effective compression thickness of the slab is taken as:

$$t_{eff} = \min(x_u, t_{slab})$$

The transformed concrete compression area is:

$$A_{ec} = b_{eff} \times t_{eff}$$

The distance of the compression block centroid from the neutral axis is:

$$Y = x_u - \frac{t_{eff}}{2}$$

The longitudinal shear range per unit length is computed as:

$$V_r' = \frac{V_r \, A_{ec} \, Y}{I_c}$$

where I_c is the composite moment of inertia.

The required stud spacing for fatigue is then obtained from:

$$s = \frac{n_s \, Q_u}{V_r'}$$

where:

- Q_u = stud capacity (Clause 606.3.1)
- n_s = number of studs per section

The function returns:

- Effective concrete compression area and thickness
- Location of compression block centroid
- Longitudinal shear range per unit length
- Required stud spacing under fatigue

Clause 606.6: Detailing Requirements for Shear Connectors Detailing checks for stud shear connectors were implemented as per IRC:22 Clause 606.6 to ensure proper fabrication, anchorage, and durability.

The following requirements are verified:

- **Stud diameter limit**

$$d_s \leq 2t_f$$

  where t_f is the thickness of the steel flange.

- **Minimum stud height**

$$h_s \geq \max(4d_s,\ 100 \text{ mm})$$

- **Stud head diameter**

$$d_{head} \geq 1.5d_s$$

- **Edge distance** Minimum edge distance from the flange edge:

$$e \geq 25 \text{ mm}$$

  If not provided, it is computed as:

$$e = \frac{b_{tf} - s_{ts}(n_s - 1) - d_s}{2}$$

- **Projection above bottom reinforcement** The stud height shall satisfy:

$$h_s \geq c_{bottom} + d_{bar} + 40 \text{ mm}$$

- **Concrete clear cover** Minimum clear cover above the stud:

$$c_c = t_{slab} - h_s \geq 25 \text{ mm}$$

The function evaluates each requirement individually and returns:

- Required and provided detailing values
- Status of each individual check
- Overall compliance flag indicating whether all detailing requirements are satisfied

Clause 606.9: Limiting Criteria for Shear Connector Spacing The maximum and minimum spacing limits for shear connectors were implemented as per IRC:22 Clause 606.9.

The maximum permissible longitudinal spacing is governed by:

$$s_{max} = \min\left(600,\ 3t_{slab},\ 4h_s\right)$$

where t_{slab} = slab thickness (mm) h_s = stud height (mm)

The minimum spacing requirement is:

$$s_{min} = 75 \text{ mm}$$

The function returns:

- Individual limiting values (600 mm, $3t_{slab}$, $4h_s$)
- Governing maximum spacing
- Minimum spacing requirement
- Verification of adequacy if provided spacing is given

Clause 606.10: Transverse Shear Resistance of Slab The transverse shear capacity of the concrete slab along the longitudinal shear plane was evaluated as per IRC:22 Clause 606.10.

The applied longitudinal shear per unit length (V_L) is checked against two resistance expressions:

Concrete shear resistance

$$V_{c1} = 0.632\, L\sqrt{f_{ck}}$$

Combined concrete and reinforcement resistance

$$V_{c2} = 0.232\, L\sqrt{f_{ck}} + 0.1\, A_{st} f_{yk}\, n$$

where L = length of potential shear plane (m) f_{ck} = concrete strength (MPa) f_{yk}

= yield strength of transverse reinforcement (MPa) A_{st} = transverse reinforcement area (cm^2/m) n = number of reinforcement layers

The governing resistance is taken as:

$$V_{Rd} = \max(V_{c1}, V_{c2})$$

Additionally, minimum transverse reinforcement is checked:

$$A_{st,min} = \frac{2.5 V_L}{f_{yk}}$$

The function returns:

- Both shear resistance values and governing capacity
- Check status against applied shear
- Minimum reinforcement requirement
- Adequacy of provided reinforcement

6.2.9 Bug Identification and Correction in IS:800 Module

During the implementation and testing of IRC:22 Clause 603.3.3.1 (Buckling Resistance Moment), an issue was identified in the existing IS:800 module available in the Osdag core directory.

The function `cl_8_2_1_2_design_bending_strength()` contained a logical error in the determination of the parameter β_b. The earlier implementation used an incorrect conditional expression, due to which the intended classification-based behaviour was not executed correctly.

Additionally, the function did not return the computed bending strength value in certain execution paths, which prevented its use in higher-level calculations such as lateral torsional buckling resistance.

The following corrections were made:

- The conditional logic for determining β_b based on section class (Plastic/Compact/Semi-compact) was rewritten.

- Proper return statements were ensured so that the calculated design bending strength could be accessed by other modules.

These corrections enabled proper integration of IS:800 bending strength calculations within the IRC:22 buckling resistance implementation.

Figure 6.3 shows the corrected implementation.

```python
@staticmethod
def cl_8_2_1_2_design_bending_strength(section_class, Zp, Ze, fy, gamma_mo, support):
    # beta_b = 1.0 if section_class == KEY_Plastic or KEY_Compact else Ze/Zp
    beta_b = 1.0 if section_class == KEY_Plastic or section_class == KEY_Compact else Ze/Zp

    Md = beta_b * Zp * fy / gamma_mo
    if support == KEY_DISP_SUPPORT1 :
        if Md < 1.2 * Ze * fy / gamma_mo:
            return Md
        else:
            return 1.2 * Ze * fy / gamma_mo
    elif support == KEY_DISP_SUPPORT2 :
        if Md < 1.5 * Ze * fy / gamma_mo:
            return Md
        else:
            return 1.5 * Ze * fy / gamma_mo

    return Md
```

Figure 6.3: Correction made in IS:800 bending strength function for proper execution and return

6.3 Task 4: Documentation

The implemented IRC:22 modules were developed following the standard OsdagBridge coding structure and naming conventions. Each clause was implemented as a separate static function with clear input parameters and dictionary-based outputs for easy integration with higher-level design workflows.

Relevant provisions from IRC:22 were directly translated into computational form, and wherever required, supporting data was obtained from the Osdag core modules such as IS 800 and IRC material property tables. The functions include clause references to ensure traceability with the code provisions.

The implementation has been organized to support modular usage, future extensions, and seamless integration within the OsdagBridge design framework.

Chapter 7

OsdagBridge Plugin Integration

7.1 Task 5

7.1.1 Objective

The objective of this task was to study the feasibility of integrating **OsdagBridge** as an external plugin for Osdag and to develop a basic prototype architecture for communication between Osdag and the bridge module.

Since OsdagBridge has not yet been implemented, a preliminary investigation was carried out to understand possible integration approaches based on the existing Osdag architecture.

—

7.1.2 Background Study

Osdag is a Python-based application built using the Qt framework. Most modern plugin systems (such as those available for VS Code) are implemented in TypeScript or JavaScript and are not directly compatible with the Osdag environment.

Therefore, a Python-based plugin architecture was proposed with the following requirements:

- Non-blocking execution to avoid UI freezing
- Communication between Osdag and the plugin using Qt signals
- Ability to execute external processes or future bridge operations

- Modular and extendable design

—

7.1.3 Proposed Integration Architecture

The proposed structure consists of three components:

- **Osdag Core** – Loads and manages plugins
- **OsdagBridge Plugin** – Interface layer between Osdag and external services
- **Worker Thread** – Executes heavy computations or external communication

Thread-based execution ensures that long operations do not block the Osdag graphical interface.

—

7.1.4 Plugin Interface Implementation

A basic plugin class was developed to simulate how OsdagBridge can be integrated.

OsdagBridge Plugin Class

```
from PyQt5.QtCore import QObject, pyqtSignal, QThread

class OsdagBridgePlugin(QObject):
    """
    Basic plugin interface for OsdagBridge integration
    """

    result_ready = pyqtSignal(dict)
    error_occurred = pyqtSignal(str)
    status_update = pyqtSignal(str)

    def __init__(self):
        super().__init__()
        self.worker_thread = None
```

```
    def run_bridge_task(self, input_data):
        self.status_update.emit("Starting OsdagBridge task...")

        self.worker = BridgeWorker(input_data)
        self.worker_thread = QThread()

        self.worker.moveToThread(self.worker_thread)

        self.worker_thread.started.connect(self.worker.run)
        self.worker.result_ready.connect(self.result_ready)
        self.worker.error_occurred.connect(self.error_occurred)
        self.worker.finished.connect(self.worker_thread.quit)

        self.worker_thread.start()
```

—

7.1.5 Worker Thread

The worker executes the actual operation independently of the UI thread.

```
class BridgeWorker(QObject):

    result_ready = pyqtSignal(dict)
    error_occurred = pyqtSignal(str)
    finished = pyqtSignal()

    def __init__(self, input_data):
        super().__init__()
        self.input_data = input_data

    def run(self):
        try:
            # Simulated processing
            result = {
```

```
                "status": "success",
                "received_data": self.input_data
            }

            self.result_ready.emit(result)

        except Exception as e:
            self.error_occurred.emit(str(e))

        finally:
            self.finished.emit()
```

—

7.1.6 Plugin Manager (Osdag Side)

A simple plugin manager was created to demonstrate how Osdag can load and access the plugin.

```
class PluginManager:

    def __init__(self):
        self.plugins = {}

    def load_plugin(self, plugin_name, plugin_class):
        plugin = plugin_class()
        self.plugins[plugin_name] = plugin
        return plugin

    def get_plugin(self, plugin_name):
        return self.plugins.get(plugin_name)
```

—

7.1.7 Usage Example

```
manager = PluginManager()

bridge_plugin = manager.load_plugin(
    "OsdagBridge",
    OsdagBridgePlugin
)

bridge_plugin.result_ready.connect(print)
bridge_plugin.status_update.connect(print)

bridge_plugin.run_bridge_task({
    "module": "Composite Beam",
    "code": "IRC22"
})
```

—

7.1.8 Observations

- Thread-based execution prevents UI blocking.
- Qt signal-slot mechanism enables safe communication with the Osdag interface.
- The architecture supports future integration with external tools, APIs, or cloud services.

—

7.1.9 Future Scope

The proposed structure can be extended to support:

- Automatic plugin discovery
- Standardized plugin API for Osdag
- External solver integration

- File-based or API-based communication
- Logging and configuration management

—

Chapter 8

Conclusions

8.1 Tasks Accomplished

During the course of the internship, multiple modules related to bridge design were developed and integrated within the OsdagBridge framework. The work focused on implementing codal provisions from various Indian Roads Congress (IRC) standards and developing a structured approach for future expansion.

The major tasks accomplished are summarized below:

- **IRC 5:2015 Implementation** Geometry and self-weight load calculation modules were developed for bridge superstructure components such as crash barriers, railings, and medians. Cross-sectional areas were computed using analytical geometric decomposition methods (trapezoids, circular segments, and composite shapes), and corresponding loads were obtained using material densities. The implementation covered:
  - RCC crash barriers (with and without footpath)
  - High containment barriers
  - Metallic crash barriers (single and double beam)
  - Bridge railings
  - Median configurations (raised kerb, RCC barrier, metallic barrier)
- **IRC 6:2017 Special Vehicle Loading** A complete computational model was developed for the Special Vehicle (prime mover with multi-axle hydraulic trailer).

The module generates axle loads, longitudinal and transverse positions, total load, and formatted axle distribution tables for analysis.

- **IRC 22:2014/2015 Composite Bridge Design** Comprehensive implementation of composite construction provisions was carried out, including:
  - Material properties (Annexure III)
  - Section classification and references to IS 800:2007
  - Positive moment capacity and buckling resistance
  - Shear resistance and shear buckling methods
  - Reduced bending under high shear
  - Serviceability checks (modular ratio and crack control)
  - Fatigue design, strength, and assessment
  - Shear connector design, spacing, fatigue, and detailing

  Cross-referencing with existing IS 800 modules in the Osdag core ensured consistency and reuse of validated functions. During testing, a functional issue was identified and corrected in the IS 800 bending strength routine to enable proper integration.

- **OsdagBridge Plugin Integration Study** A preliminary study was conducted to explore methods for integrating OsdagBridge as an external plugin to the Osdag application. Various extension architectures were evaluated, and a Python-based approach using Qt signals/slots and background worker threads was proposed to ensure safe communication with the user interface and non-blocking execution.

Overall, the internship resulted in the development of structured, reusable, and codal-compliant modules that significantly expand the bridge design capabilities within the Osdag ecosystem.

8.2 Skills Developed

The internship provided extensive exposure to both technical and professional competencies relevant to structural engineering software development.

Technical Skills

- Interpretation and implementation of IRC design codes (IRC 5, IRC 6, IRC 22)
- Development of engineering computation modules using Python
- Geometric modelling and load calculation for structural components
- Composite bridge design concepts including strength, stability, fatigue, and serviceability
- Integration of multiple standards (IRC and IS 800) within a unified framework
- Use of Git for version control, branch management, and collaborative workflows
- Code structuring, modular programming, and documentation practices
- Debugging and validation of engineering software

Software and Development Skills

- Working with a large open-source engineering codebase (Osdag)
- Understanding plugin architecture and software extensibility concepts
- Basic exploration of UI–backend communication using Qt mechanisms
- Preparation of technical documentation and structured reports using LaTeX

Professional Skills

- Systematic interpretation of engineering standards and technical literature
- Problem-solving through iterative testing and validation
- Independent learning and research for new development tasks
- Maintaining coding standards and organized project structure

This internship provided valuable experience in bridging structural engineering principles with software development, and contributed toward the development of scalable bridge design functionality within the Osdag platform.

Chapter A

Appendix

A.1 Work Reports

Date	Day	Task	Work Hours
1 Dec 2025	Monday	Did initial setup for Osdag desktop	3hrs
2 Dec 2025	Tuesday	Did setup for Osdag Web	2 hrs
3 Dec 2025	Wednesday	Fixed few errors in setup	3.5hrs
4 Dec 2025	Thursday	Contacted team members for help in inital setup and debugged	2 hrs
5 Dec 2025	Friday	Gmeet - Osdag Mass testing	1 hr
8 Dec 2025	Monday	worked on testing for Seated Angle	2 hrs
9 Dec 2025	Tuesday	worked on testing for Seated Angle	3 hrs
10 Dec 2025	Wednesday	worked on testing for Seated Angle	1 hr
11 Dec 2025	Thursday	still was working on testing as there were few errors while running osdag	2 hrs
12 Dec 2025	Friday	uploaded testing video on youtube and submitted	1.5 hrs
13 Dec 2025	Saturday	Gmeet - For IRC 5 explanation	1 hr
14 Dec 2025	Sunday	Went through codebase	2 hr
15 Dec 2025	Monday	Went through clauses of IRC5	2 hrs
16 Dec 2025	Tuesday	Gmeet	1 hr
17 Dec 2025	Wednesday	coded for crash barrier	2.5 hrs
18 Dec 2025	Thursday	Gmeet	1 hr
19 Dec 2025	Friday	wrote clauses for railings	2.5 hrs
21 Dec 2025	Sunday	Gmeet	1 hr
22 Dec 2025	Monday	worked on comments for IRC5 codes	2 hr
23 Dec 2025	Tuesday	segregated the code into irc5 file, geometry and loads	2 hr
24 Dec 2025	Wednesday	Gmeet	1hr
25 Dec 2025	Thursday	segregated the code into irc5 file, geometry and loads	1.5 hrs
26 Dec 2025	Friday	Gmeet	1 hr
27 Dec 2025	Saturday	worked on IRC6 special vehicle clause	2 hrs
29 Dec 2025	Monday	Gmeet	1 hr
30 Dec 2025	Tuesday	tried different ways for area of crash barrier by dividing fig into portions	3.5 hrs
31 Dec 2025	Wednesday	Gmeet	1 hr
1 Jan 2026	Thursday	Gmeet	1 hr
2 Jan 2026	Friday	Reduced error for area of crash barrier by dividing fig into portions	3 hrs
3 Jan 2026	Saturday	Gmeet	1 hr
4 Jan 2026	Sunday	worked on the comments for area	2.5 hrs
5 Jan 2026	Monday	Gmeet	1 hr
6 Jan 2026	Tuesday	Leave	-
8 Jan 2026	Thursday	Gmeet	1 hr
9 Jan 2026	Friday	created a PR with all the changes for IRC5	1.5 hrs
10 Jan 2026	Saturday	Gmeet	1 hr
11 Jan 2026	Sunday	IRC22 clauses	2 hrs
12 Jan 2026	Monday	created a draft PR with corrections - for median	1.5 hrs
13 Jan 2026	Tuesday	leave	-
14 Jan 2026	Wednesday	leave	-
15 Jan 2026	Thursday	leave	-
16 Jan 2026	Friday	leave	-
19 Jan 2026	Monday	leave	-
20 Jan 2026	Tuesday	finished all clauses of IRC22	9 hrs
21 Jan 2026	Wednesday	Gmeet	1 hr
22 Jan 2027	Thursday	setting up Osdag path fortesting IRC22 clauses	3.5 hrs
23 Jan 2026	Friday	Gmeet + testing of IRC22	2.5 hrs
24 Jan 2026	Saturday	Reseached for plugin integration	3.5 hrs
25 Jan 2026	Sunday	Reseached for plugin integration + docs we can refer	3 hrs
26 Jan 2026	Monday	Gmeet	1 hr
27 Jan 2026	Tuesday	planned the way plugins can be integrated	2.5 hrs
28 Jan 2026	Wednesday	Gmeet	1.5 hrs
29 Jan 2026	Thursday	worked on ppt for outputs of IRC22 clauses	3 hrs
30 Jan 2026	Friday	Gmeet	1 hr
1 Feb 2026	Sunday	Gmeet + corrections	1 hr
2 Feb 2026	Monday	Gmeet	1 hr
3 Feb 2026	Tuesday	making changes to shift IRC used terms in key file from common	1.5 hrs
4 Feb 2026	Wednesday	Gmeet	1 hrs
5 Feb	Thursday	CAD for loading tab in additional inputs + corrected few clauses of IRC6	3 hrs
6 Feb 2026	Friday	Gmeet	1 hr
7 Feb 2026	Saturday	Gmeet + CAD	1.5 hrs
9 Feb 2026	Monday	Gmeet	1 hr
10 Feb 2026	Tuesday	testing of IRC22 clauses for edgecases	2.5 hrs
11 Feb 2026	Wednesday	Gmeet	1 hr

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.