FOSSEE Autumn Internship Report

On

Development of CADs for Osdag

Submitted by

Arushi

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

Lovely Professional University, Punjab

India

Under the Guidance of

Prof. Siddharth Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Nidhi Khare

Aditya Donde

Mohamed Suhail

January 4, 2026

Abstract

This report presents the work conducted during the FOSSEE Autumn Internship program at IIT Bombay, focusing on developing computer-aided design (CAD) capabilities and standards compliance features for the OsdagBridge steel structure design software. The project addressed four core objectives: implementing IRC 5:2015 standard compliance through a Python module for automated design verification of road bridge parameters including safety kerb dimensions, carriageway width, and design life specifications; designing a normalized SQLite database schema with parent-child relationships for efficient management and retrieval of meteorological data from Indian weather stations; developing a comprehensive 2D CAD visualization system featuring parametric bridge modeling with cross-sectional and top views, professional dimensioning, and interactive user controls; and integrating the visualization system into the OsdagBridge desktop application with real-time parameter synchronization and enhanced user interface elements. These contributions provide the OsdagBridge ecosystem with automated standards verification, optimized data management infrastructure, advanced visualization capabilities, and improved user experience, thereby supporting efficient and compliant structural steel design workflows.

Acknowledgments

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

First and foremost, I would like to thank the Osdag team, particularly Nidhi Khare, Aditya Donde, and Mohamed Suhail, for their continuous guidance, patience, and technical support during my internship. Their expertise and collaborative spirit were instrumental in helping me navigate the complexities of the project.

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

Contents

List of Tables

List of Figures

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
2	SWAYAMPRABHA	Access 24x7 TV programs	Enable viewing facilities
Digital Content Access			
3	National Digital Library	Access e-content	List e-content
4	e-PG Pathshala	Access free e-content	Host e-books
5	Shodhganga	Access research theses	List institutional theses
6	e-ShodhSindhu	Access e-resources	Access for institutions
Hands-on Learning			
7	e-Yantra	Embedded systems training	Create e-Yantra labs
8	FOSSEE	Open-source software	Run open-source labs
9	Spoken Tutorial	Learn IT skills	Provide self-learning content
10	Virtual Labs	Online experiments	Develop experiments
E-Governance			
11	SAMARTH ERP	Digital lifecycle	Enable e-governance
Tracking and Research			
12	VIDWAN	Access experts	Monitor outcomes
13	Shodh Shuddhi	Plagiarism-free work	Improve quality
14	Academic Bank of Credits	Store and transfer	Facilitate 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.

Figure 1.1: FOSSEE Projects and Activities

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.

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 [**?**, **?**, **?**]. 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.

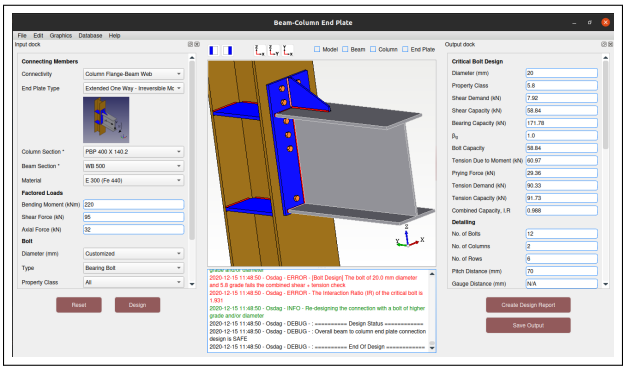

Figure 1.2: Osdag GUI

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

Task 1: IRC5-2015 Code Implementation

2.1 Problem Statement

The Indian Roads Congress (IRC) has published comprehensive standards for road bridge design and general specifications in the IRC 5:2015 standard. While these standards exist in textual form, there was no automated, programmatic way to enforce compliance with these design requirements. The task was to develop a Python module that encodes the critical design clauses from IRC 5:2015, enabling automated checking of bridge design parameters against standard requirements.

The primary objectives were:

- Implement Python functions corresponding to specific IRC clauses.
- Validate design parameters against minimum/maximum requirements.
- Provide clear return values indicating compliance or non-compliance.
- Ensure code is well-documented with clause references.

2.2 Methodology and Implementation

2.2.1 Overview of IRC5-2015 Module

The IRC5-2015 module was developed as a Python class-based implementation with static methods for each design check. The module provides utility functions that:

- Validate safety kerb width (IRC Clause 101.41)
- Calculate road kerb dimensions (IRC Clause 109.8.1)
- Calculate safety kerb dimensions (IRC Clause 109.8.3)
- Determine design life (IRC Clause 104.1.3.4)
- Validate carriageway width and lane count (IRC Clause 104.3.1)

2.2.2 Design Specifications Implemented

The module encodes the following critical design parameters from IRC 5:2015:

Design Parameter	**Value**	**Unit**
Safety Kerb Minimum Width	750	mm
Road Kerb Width	225	mm
Road Kerb Height	225	mm
Minimum Single Lane Width	4250	mm
Minimum Double Lane Width	7500	mm
Additional Lane Width	3500	mm
Design Life	100	years
Minimum Kerb Effective Width	175	mm
Minimum Kerb Edge Radius	25	mm

Table 2.1: IRC 5:2015 Design Parameters Encoded in Module

2.2.3 Module Architecture

The implementation is organized into two components:

- **Constants Module**: Contains global constants and configuration keys that define IRC standards parameters.
- **Main Module**: Contains the IRC5-2015 class with all static design check methods.

The use of a class with static methods provides clear organization and allows for future expansion with additional clauses.

2.3 Implementation

2.3.1 Constants Definition

The constants module defines all IRC-related parameters used throughout the design checks:

```
KEY_FOOTPATH = ["None", "Single Side", "Both Sides"]
KEY_SAFETY_KERB_MIN_WIDTH = 750  # in mm
KEY_SAFETY_KERB_PLACEMENT = ['Single Side', 'Both Sides']
KEY_FOOTPATH_CLEAR_MIN_WIDTH = 1500  # in mm
KEY_RAILING_MIN_HEIGHT = [1100, 1250] # in mm
KEY_CYCLE_TRACK = ['None', 'Single', 'Both Sides']
KEY_MIN_SKEW_ANGLE = 30  # in degrees
KEY_WEARING_COAT = ['bituminous', 'concrete']
KEY_CRASH_BARRIER_TYPE = ['Flexible', 'Semi-Rigid', 'Rigid']
KEY_RAILING_TYPE = ['RCC', 'steel']
KEY_MIN_LOGITUDINAL_GRADIENT = 0.3  # in percent
KEY_MAX_BRIDGE_LENGTH_SINGLE_CURVE = 30  # in meters
KEY_RIGID_CRASH_BARRIER_TYPE = ['IRC-5R', 'High Containment']
KEY_MIN_SINGLE_LANE = 4250  # in mm
KEY_MIN_DOUBLE_LANE = 7500  # in mm
KEY_ADDITIONAL_LANE = 3500  # in mm
```

Listing 2.1: Common Constants for IRC5-2015 Module

2.3.2 IRC5-2015 Main Module

The main module implements specific design checks as static methods:

```
@staticmethod
def cl_101_41_safety_kerb_width(kerb_width, footpath):
```

```
    """Safety kerb width requirements (IRC Clause 101.41)

    IRC 5:2015 Standard: Minimum width = 750mm (0.75m)
    A kerb having width of at least 750 mm for occasional use
    by pedestrians, where footpath is not provided.
    """
    if footpath == KEY_FOOTPATH[0]:  # "None"
        if kerb_width < KEY_SAFETY_KERB_MIN_WIDTH:
            check = False
            return check
        elif kerb_width >= KEY_SAFETY_KERB_MIN_WIDTH:
            kerb_placement = KEY_SAFETY_KERB_PLACEMENT[1]
            check = True
            return kerb_placement, check
    elif footpath == KEY_FOOTPATH[1]:  # "Single Side"
        if kerb_width < KEY_SAFETY_KERB_MIN_WIDTH:
            check = False
            return check
        elif kerb_width >= KEY_SAFETY_KERB_MIN_WIDTH:
            kerb_placement = KEY_SAFETY_KERB_PLACEMENT[0]
            check = True
            return kerb_placement, check
```

Listing 2.2: Safety Kerb Width Check - IRC Clause 101.41

```
@staticmethod
def cl_109_8_1_road_kerb_outline(design_dict):
    """Road kerb dimensions as per IRC 5:2015
    Clause 109.8.1 - Standard dimensions for road kerbs
    """
    road_kerb_dimensions = {
        'road_kerb_width': 225,  # width in mm
        'road_kerb_effective_width': 175,  # effective width in mm
        'road_kerb_height': 225,  # height in mm
        'road_kerb_effective_height': 200,  # effective height in mm
        'road_kerb_edge_radius': 25  # edge radius in mm
    }
    design_dict.update(road_kerb_dimensions)
    return design_dict
```

Listing 2.3: Road Kerb Dimensions - IRC Clause 109.8.1

```python
@staticmethod
def cl_104_3_1_carriageway_width(carriageway_width):
    """Carriageway width and lane count determination
    Clause 104.3.1 - Minimum carriageway width: 4250 mm (1-lane),
    7500 mm (2-lane), +3500 mm per extra lane.
    """
    traffic_lanes = 0
    if carriageway_width >= KEY_MIN_SINGLE_LANE:
        traffic_lanes = 1
        if carriageway_width >= KEY_MIN_DOUBLE_LANE:
            traffic_lanes = 2
            remaining = carriageway_width - KEY_MIN_DOUBLE_LANE
            while remaining >= KEY_ADDITIONAL_LANE:
                traffic_lanes += 1
                remaining -= KEY_ADDITIONAL_LANE

    if traffic_lanes == 0:
        return False, f"Carriageway width {carriageway_width:.0f}mm
           is insufficient"

    return True, f"Carriageway {carriageway_width:.0f}mm supports {
       traffic_lanes} lane(s)"
```

Listing 2.4: Carriageway Width Validation - IRC Clause 104.3.1

Safety Kerb Cross-Sectional Area Calculation

A critical geometric calculation implemented in the module is the cross-sectional area of safety kerbs, used for structural analysis and material estimation. The formula accounts for the complex kerb profile geometry including curved edges.

Mathematical Formulation:

The safety kerb profile consists of four distinct geometric components (as per IRC Clause 109.8.3):

$$A_{kerb} = A_{rectangle} + A_{quarter\_circle} + A_{top\_radius} + A_{side\_radius} \tag{2.1}$$

Breaking down each component:

$$A_{rectangle} = w_{eff} \times h \tag{2.2}$$

$$A_{quarter\_circle} = \frac{\pi r^2}{4} \tag{2.3}$$

$$A_{top\_radius} = w_{eff} \times r \tag{2.4}$$

$$A_{side\_radius} = 0.5 \times r \times h_{eff} \tag{2.5}$$

where:

- w_{eff} = effective width (175 mm minimum)
- h = kerb height (225 mm)
- h_{eff} = effective height (200 mm)
- r = edge radius (25 mm minimum)

Complete Formula:

$$A_{kerb} = w_{eff} \cdot h + \frac{\pi r^2}{4} + w_{eff} \cdot r + 0.5 \cdot r \cdot h_{eff} \tag{2.6}$$

Implementation in Module:

```
@staticmethod
def cl_109_8_3_safety_kerb_outline(design_dict):
    """Calculate safety kerb dimensions and cross-sectional area
    Clause 109.8.3 - Safety kerb has same outline as road kerb
    except top width must be >= 750 mm
    """
    # Get road kerb dimensions as baseline
    kerb_dims = IRC5_2015.cl_109_8_1_road_kerb_outline(design_dict)

    safety_kerb_dimensions = {
        'safety_kerb_width': max(kerb_dims['road_kerb_width'],
                                 KEY_SAFETY_KERB_MIN_WIDTH),
        'safety_kerb_effective_width': kerb_dims['
           road_kerb_effective_width'],
        'safety_kerb_height': kerb_dims['road_kerb_height'],
        'safety_kerb_effective_height': kerb_dims['
```

```
            road_kerb_effective_height'],
        'safety_kerb_edge_radius': kerb_dims['road_kerb_edge_radius']
    }

    design_dict.update(safety_kerb_dimensions)

    # Calculate cross-sectional area (mm^2)
    w_eff = safety_kerb_dimensions['safety_kerb_effective_width']
    h = safety_kerb_dimensions['safety_kerb_height']
    h_eff = safety_kerb_dimensions['safety_kerb_effective_height']
    r = safety_kerb_dimensions['safety_kerb_edge_radius']

    safety_kerb_area = (w_eff * h +                    # Main
        rectangle
                        (math.pi * r**2) / 4 +         # Quarter
                            circle (edge)
                        w_eff * r +                    # Top radius
                            extension
                        0.5 * r * h_eff)               # Side radius
                            extension

    design_dict['safety_kerb_area'] = safety_kerb_area  # in mm^2
    return design_dict
```

Listing 2.5: Safety Kerb Area Calculation - IRC Clause 109.8.3

Numerical Example:

With standard IRC dimensions:

$$w_{eff} = 175 \text{ mm}$$

$$h = 225 \text{ mm}$$

$$h_{eff} = 200 \text{ mm}$$

$$r = 25 \text{ mm}$$

$$
\begin{aligned}
A_{kerb} &= (175 \times 225) + \frac{\pi \times 25^2}{4} + (175 \times 25) + (0.5 \times 25 \times 200) \\
&= 39,375 + 490.87 + 4,375 + 2,500 \\
&= 46,740.87 \text{ mm}^2 \\
&\approx 46,741 \text{ mm}^2
\end{aligned}
$$

This area value is used for:

- Material quantity estimation (concrete volume)
- Weight calculations for structural analysis
- Cost estimation for construction
- Compliance verification with IRC standards

Crash Barrier Cross-Sectional Area Calculation

IRC 5:2015 specifies trapezoidal concrete crash barriers with varying dimensions based on containment levels and footpath configurations. The cross-sectional area calculation is essential for:

- Material quantity estimation (concrete volume per meter)
- Structural weight analysis for dead load calculations
- Cost estimation and project budgeting
- Compliance with IRC 109.10 crash barrier requirements

Geometric Formula:

For trapezoidal concrete barriers, the cross-sectional area is calculated using the standard trapezoid formula:

$$
A_{barrier} = \frac{w_{top} + w_{base}}{2} \times h \tag{2.7}
$$

where:

- w_{top} = Top width of barrier (mm)
- w_{base} = Base width of barrier (mm)

- h = Total height of barrier (mm)
- $A_{barrier}$ = Cross-sectional area (mm^2)

IRC 5:2015 Standard Configurations:

Table 2.2: Crash Barrier Configurations (IRC Clause 109.10)

Configuration	Base Width	Top Width	Height	Area
Fig. 1 (Low, no footpath)	450 mm	175 mm	900 mm	281,250 mm^2
Fig. 2 (Low, with footpath)	450 mm	175 mm	1100 mm	343,750 mm^2
Fig. 3 (High containment)	525 mm	250 mm	1550 mm	600,625 mm^2

Implementation in Module:

```
# Extract dimensions from IRC 5:2015 configurations
barrier_width = params.get('base_width_mm')    # Base width
top_width = params.get('top_width_mm')         # Top width
barrier_height = params.get('total_height_mm') # Height

# Calculate cross-sectional area (trapezoidal approximation)
if top_width:
    # Concrete barriers: trapezoidal area = (top + base) / 2 * height
    barrier_area = (top_width + barrier_width) / 2 * barrier_height
else:
    # Metallic barriers: rectangular calculation for kerb sections
    kerb_height = params.get('kerb_height_mm', 100)
    barrier_area = barrier_width * kerb_height + \
                   barrier_width * (barrier_height - kerb_height)

# Report results with IRC figure reference
report["passes"].append(
    f"Crash Barrier Design: Width {barrier_width}mm, "
    f"Height {barrier_height}mm, Area {barrier_area:.0f}mm^2 "
    f"(IRC 5:2015 {fig_ref})"
)
```

Listing 2.6: Crash Barrier Area Calculation - IRC Clause 109.10

Numerical Example (Fig. 1 Configuration):

For a standard low-containment barrier without footpath:

$$\begin{aligned} w_{top} &= 175 \text{ mm} \\ w_{base} &= 450 \text{ mm} \\ h &= 900 \text{ mm} \end{aligned}$$

$$\begin{aligned} A_{barrier} &= \frac{175 + 450}{2} \times 900 \\ &= \frac{625}{2} \times 900 \\ &= 312.5 \times 900 \\ &= 281,250 \text{ mm}^2 \\ &\approx 0.281 \text{ m}^2 \text{ (per meter length)} \end{aligned}$$

For a 50-meter bridge, the total concrete volume would be:

$$\begin{aligned} V_{concrete} &= A_{barrier} \times \text{Length} \times 2 \text{ (both sides)} \\ &= 0.281 \text{ m}^2 \times 50 \text{ m} \times 2 \\ &= 28.1 \text{ m}^3 \end{aligned}$$

Design Considerations:

- **Trapezoidal Profile**: The wider base provides stability and crash energy absorption
- **IRC Compliance**: Dimensions from IRC 5:2015 Figures 1-4 ensure safety standards
- **Footpath Integration**: Configuration varies based on footpath presence/absence
- **Containment Levels**: High-containment barriers use larger dimensions (Fig. 3)

2.3.3 Implementation Approach

The implementation follows these principles:

- **Clause-Based Organization**: Each method corresponds directly to an IRC 5:2015 clause, making the code self-documenting.
- **Immutable Constants**: All standards parameters are defined as constants, preventing accidental modifications.
- **Clear Return Values**: Functions return tuples of (success_flag, result_data) or specific configuration values.
- **Docstring Documentation**: Each method includes docstrings with IRC clause references and parameter descriptions.

2.4 Usage and Documentation

2.4.1 Usage Example

To use the IRC5-2015 module in a bridge design application:

```
from irc5_2015 import IRC5_2015

# Check safety kerb width for bridge with no footpath
kerb_width_mm = 800
footpath_type = "None"
result = IRC5_2015.cl_101_41_safety_kerb_width(kerb_width_mm,
    footpath_type)
print(f"Kerb Check Result: {result}")

# Get standard road kerb dimensions
design_params = {}
design_params = IRC5_2015.cl_109_8_1_road_kerb_outline(design_params)
print(f"Road Kerb Dimensions: {design_params}")

# Validate carriageway width
carriageway_width = 8000  # 8000 mm
valid, message = IRC5_2015.cl_104_3_1_carriageway_width(
    carriageway_width)
print(f"Carriageway Validation: {message}")
```

Listing 2.7: Example: Using IRC5-2015 Module

2.4.2 Integration with OsdagBridge

The IRC5-2015 module is designed to integrate seamlessly with the OsdagBridge framework, providing automated compliance checking for Indian bridge designs. Future enhancements include:

- Integration with the OsdagBridge core analysis engine.
- UI components for displaying IRC compliance status.
- Report generation including IRC clause references.
- Extensibility for additional IRC clauses as required.

Chapter 3

Task 2: Database Schema Design and SQL Implementation

3.1 Problem Statement

The task involved designing and implementing a structured weather database that stores meteorological data across multiple Indian states and weather stations. The database needed to organize complex weather information while maintaining data integrity, eliminating redundancy, and supporting efficient queries. The specific challenges were:

- **Data Heterogeneity**: Weather data comprises different types (temperature, wind speed, zone information) that needed unified storage.
- **Spatial Organization**: Data needed to be organized by state-station combinations across India.
- **Data Integrity**: Preventing duplicate state-station entries and maintaining referential integrity across related tables.
- **Query Efficiency**: Enabling fast lookups and filtering by state, station, or weather parameters.

3.2 Methodology and Implementation

3.2.1 Database Design Approach

A normalized relational database schema was designed using the canonical parent-child relationship pattern:

- **Parent Table (stations)**: Serves as the single source of truth for all valid state-station combinations.
- **Child Tables**: Three separate tables store specific data types (temperature_data, zone_data, windspeed_data), each with a foreign key reference to the stations table.

This design ensures:

- No duplicate state-station pairs across the database.
- Cascading updates and deletes maintain consistency.
- Clear, modular organization of different data types.
- Efficient indexing on frequently queried columns.

3.2.2 Database Schema Implementation

The complete SQLite schema includes four tables with proper constraints and indexes:

```
-- Stations Table (Parent/Canonical Table)
CREATE TABLE stations (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    state VARCHAR(100) NOT NULL,
    station VARCHAR(100) NOT NULL,
    UNIQUE (state, station)
);

-- Temperature Data (Child Table)
CREATE TABLE temperature_data (
    station_id INTEGER NOT NULL,
    max_temp DECIMAL(5, 2),
    min_temp DECIMAL(5, 2),
    PRIMARY KEY (station_id),
    FOREIGN KEY (station_id) REFERENCES stations(id)
```

```
        ON UPDATE CASCADE ON DELETE CASCADE
);

-- Zone Data (Child Table)
CREATE TABLE zone_data (
    station_id INTEGER NOT NULL,
    zone VARCHAR(10) NOT NULL,
    z_value DECIMAL(4, 2) NOT NULL,
    PRIMARY KEY (station_id),
    FOREIGN KEY (station_id) REFERENCES stations(id)
        ON UPDATE CASCADE ON DELETE CASCADE
);

-- Windspeed Data (Child Table)
CREATE TABLE windspeed_data (
    station_id INTEGER NOT NULL,
    wind_speed INTEGER NOT NULL,
    PRIMARY KEY (station_id),
    FOREIGN KEY (station_id) REFERENCES stations(id)
        ON UPDATE CASCADE ON DELETE CASCADE
);

-- Create indexes for better query performance
CREATE INDEX idx_stations_state ON stations(state);
CREATE INDEX idx_stations_station ON stations(station);
CREATE INDEX idx_station_id ON stations(id);
```

Listing 3.1: Weather Database Schema - Table Definitions

3.2.3 Data Coverage

The database includes comprehensive coverage of weather data across India:

3.3 Python Database Utility Implementation

The database is managed through a comprehensive Python utility class that provides:

- Database connection management.

State/Region	Stations	Example Locations
Andhra Pradesh	21	Visakhapatnam, Vijayawada, Kurnool
Karnataka	25	Bengaluru, Mangalore, Mysuru
Tamil Nadu	18	Chennai, Coimbatore, Madurai
Gujarat	24	Ahmedabad, Surat, Rajkot
Haryana	7	Hissar, Gurgaon, Karnal
Himachal Pradesh	10	Shimla, Manali, Dharamshala
Jammu & Kashmir	12	Srinagar, Jammu, Gulmarg
... and many other Indian states		

Table 3.1: Weather Database Coverage by State

- Schema execution and initialization.
- Data querying and filtering functions.
- Bootstrap and recovery mechanisms for legacy databases.

3.3.1 Database Utility Class

```
class Database:
    """Weather DB with (state, station) as the canonical key."""

    def __init__(self, db_name: str = 'weather.db'):
        self.db_name = db_name
        self.connection: Optional[sqlite3.Connection] = None
        self.cursor: Optional[sqlite3.Cursor] = None

    def connect(self):
        """Establish connection to the database."""
        self.connection = sqlite3.connect(self.db_name)
        self.connection.execute("PRAGMA foreign_keys = ON")
        self.cursor = self.connection.cursor()

```

```
    def close(self):
        """Close the database connection."""
        if self.connection:
            self.connection.close()
            self.connection = None
            self.cursor = None

    def run_schema(self, schema_file: str = 'bridge_schema.sql'):
        """Execute the SQL schema file to create and seed database."""
        if not os.path.exists(schema_file):
            raise FileNotFoundError(f"Schema file not found: {schema_file}")

        self.connect()
        with open(schema_file, 'r', encoding='utf-8') as f:
            schema_sql = f.read()

        self.connection.executescript(schema_sql)
        self._bootstrap_stations_if_missing()
        print(f"Database '{self.db_name}' created successfully!")
```

Listing 3.2: Database Connection and Schema Management

3.3.2 Data Query Functions

```
def get_all_states(self) -> List[str]:
    """List all unique states from canonical stations table."""
    self.cursor.execute("SELECT DISTINCT state FROM stations ORDER BY state")
    return [row[0] for row in self.cursor.fetchall()]

def get_stations_by_state(self, state: str) -> List[str]:
    """Get all stations for a given state."""
    self.cursor.execute("SELECT station FROM stations WHERE state = ? ORDER BY station", (state,))
    return [row[0] for row in self.cursor.fetchall()]

```

```python
def get_temperature_data(self, state: str, station: str) -> Dict[str, float]:
    """Retrieve temperature data for a specific station."""
    self.cursor.execute("""
        SELECT t.max_temp, t.min_temp
        FROM temperature_data t
        JOIN stations s ON t.station_id = s.id
        WHERE s.state = ? AND s.station = ?
    """, (state, station))
    result = self.cursor.fetchone()
    return {'max_temp': result[0], 'min_temp': result[1]} if result
        else None

def get_windspeed_data(self, state: str, station: str) -> int:
    """Retrieve wind speed for a specific station."""
    self.cursor.execute("""
        SELECT w.wind_speed
        FROM windspeed_data w
        JOIN stations s ON w.station_id = s.id
        WHERE s.state = ? AND s.station = ?
    """, (state, station))
    result = self.cursor.fetchone()
    return result[0] if result else None
```

Listing 3.3: Query Functions for Weather Data

3.3.3 Usage Example

```python
# Initialize and create database
db = Database('weather.db')
db.run_schema('bridge_schema.sql')

# Get all states
states = db.get_all_states()
print(f"Available states: {states[:5]}")  # First 5

# Get stations for a state
stations = db.get_stations_by_state('Karnataka')
print(f"Karnataka stations: {stations[:3]}")
```

```
12
13 # Retrieve weather data
14 temp_data = db.get_temperature_data('Karnataka', 'Bengaluru')
15 print(f"Bengaluru: Max={temp_data['max_temp']} C, Min={temp_data['
       min_temp']} C")
16
17 wind_speed = db.get_windspeed_data('Karnataka', 'Bengaluru')
18 print(f"Wind speed: {wind_speed} km/h")
19
20 db.close()
```

Listing 3.4: Database Initialization and Query Example

3.4 Usage and Testing

3.4.1 Key Features

- **Normalization**: Uses Third Normal Form (3NF) to eliminate data redundancy.
- **Referential Integrity**: Foreign key constraints with cascading operations.
- **Performance Optimization**: Strategic indexes on frequently queried columns (state, station, id).
- **Legacy Support**: Bootstrap mechanism to handle older database formats.
- **Type Safety**: Use of type hints in Python for better code documentation.

3.4.2 Benefits of This Design

1. **Single Source of Truth**: The stations table ensures no duplicate state-station combinations.
2. **Scalability**: New child tables can be added without modifying existing structure.
3. **Data Consistency**: Cascading updates/deletes prevent orphaned records.
4. **Query Efficiency**: Indexes enable fast filtering and aggregation operations.
5. **Maintainability**: Clear table organization makes schema comprehensible and maintainable.

Chapter 4

Task 3: Development of 2D CAD Visualization System

4.1 Problem Statement

The visualization and comprehension of complex bridge geometries is critical for both designers and stakeholders. Traditional engineering drawings, while accurate, require significant training to interpret. Modern bridge design tools must provide:

- **Intuitive Visual Representation**: Clear, professional renderings of bridge geometry in multiple views.
- **Parametric Geometry**: Dynamic recalculation and visualization as design parameters change.
- **Interactive Exploration**: Tools for users to zoom, pan, and inspect specific components.
- **Automatic Annotation**: Intelligent dimensioning and labeling without manual intervention.
- **Configuration Flexibility**: Support for diverse bridge types, skew angles, medians, and footpath arrangements.

The primary objective of Task 3 was to develop a complete, production-quality 2D CAD visualization system capable of rendering realistic bridge geometry with professional-grade architectural drawings. The system needed to handle complex

geometric relationships including cross-bracing patterns, skewed layouts, and optional median configurations.

4.2 System Architecture and Design

4.2.1 System Overview

The 2D CAD system is built around a central widget class that encapsulates:

- **Parametric Bridge Model**: Stores 15+ bridge design parameters in a unified dictionary.
- **Dual-View Rendering**: Supports both cross-section and top-view (plan view) presentations.
- **Interactive Event Handling**: Responds to mouse movements for hover-based component identification.
- **Professional Dimensioning System**: Automatic calculation and placement of dimension lines with extension lines, arrows, and text.
- **Component Rendering**: Modular, reusable methods for drawing girders, decks, railings, crash barriers, cross-bracing, end diaphragms, and medians.

4.2.2 Parametric Bridge Model

The entire bridge geometry is defined through a comprehensive parameter dictionary (all dimensions in mm):

This parametric approach enables:

1. **Real-Time Updates**: Change any parameter → geometry recalculates and redraws instantly.
2. **Infinite Configurations**: Support diverse bridge types without code modifications.
3. **Automatic Constraint Satisfaction**: Positions adjust intelligently to maintain design validity.

Parameter	Range	Description
Longitudinal Geometry		
span_length	20-45 m	Bridge span in longitudinal direction
cross_bracing_spacing	1-10 m	Spacing of cross-bracing elements
Transverse Layout		
num_girders	2-12	Number of parallel main girders
girder_spacing	2-4 m	Center-to-center girder spacing
carriageway_width	4.25-24 m	Main traffic lane width
deck_overhang	0.5-2 m	Overhang beyond outer girders
Deck Elements		
deck_thickness	150-300 mm	Structural concrete slab thickness
footpath_config	4 options	None/Left/Right/Both
footpath_width	1-2 m	Pedestrian path width (if present)
footpath_thickness	150-250 mm	Footpath slab thickness
Safety and Barriers		
crash_barrier_width	300-600 mm	Width of vehicular crash barriers
railing_height	1-1.5 m	Pedestrian railing height
Extra Feature		
median_present	Yes/No	Whether center median exists
median_width	1-3 m	Median divider width (if present)
skew_angle	-15 to +15°	Bridge skew angle

Table 4.1: Complete Parametric Bridge Model Parameters

4.3 Core Implementation - Cross-Section View

4.3.1 Rendering Strategy and Visual Scaling

A critical innovation is the use of **visual scaling factors** for girder representation. Actual I-beam flanges are only 180 mm wide, but at full model scale they become nearly

invisible on screen. The solution:

Component	**Actual (mm)**	**Scale Factor**	**Visual (mm)**
Girder Depth	500	3.0x	1500 (exaggerated)
Flange Width	180	3.75x	675 (exaggerated)
Flange Thickness	17.2	4.05x	69.66 (exaggerated)
Web Thickness	10.2	3.75x	38.25 (exaggerated)

Table 4.2: Visual Scaling Factors for CAD Clarity

This allows structural details to remain clearly visible while maintaining overall proportionality.

4.3.2 Components Rendered in Cross-Section

Component	**Details Rendered**
Main Girders	I-shaped sections with web and flanges at visual scale
Stiffeners	Vertical reinforcement plates on girder webs
Deck Slab	Concrete structural slab spanning girders
Footpaths	Optional pedestrian walkways (left/right/both) with independent thickness
Railings	RCC posts with hollow rectangular center section
Crash Barriers	Trapezoidal profile with IRC 5:2015 dimensions
Median	Two opposing crash barriers (if configured)

Table 4.3: Cross-Section Components

4.3.3 Advanced Cross-Section Features

Intelligent Girder Positioning with Overhang-Based Spacing Adjustment

A critical geometric constraint in bridge design is ensuring girders remain within the deck boundaries while maintaining structural integrity. The implemented algorithm automatically adjusts girder spacing based on the specified deck overhang, ensuring feasible geometry regardless of user-specified parameters.

Problem Statement:

When users specify:

- Number of girders: n
- Desired girder spacing: $s_{desired}$ (mm)
- Deck overhang: o (mm) - distance from deck edge to outermost girder
- Total deck width: W_{deck} (mm)

The system must calculate actual girder positions such that:

1. First girder is positioned at $x_{\text{first}} = x_{\text{deck\_left}} + o$
2. Last girder is positioned at $x_{\text{last}} = x_{\text{deck\_right}} - o$
3. All intermediate girders are evenly spaced between first and last

Mathematical Formulation:

The actual girder spacing s_{actual} is calculated as:

$$s_{actual} = \frac{W_{available}}{n-1} \tag{4.1}$$

where the available width for girder distribution is:

$$W_{available} = W_{deck} - 2o \tag{4.2}$$

This ensures that:

$$x_{last} - x_{first} = (n-1) \cdot s_{actual} \tag{4.3}$$

The position of the i-th girder (where $i = 0, 1, 2, \ldots, n-1$) is:

$$x_i = x_{\text{first}} + i \cdot s_{\text{actual}} = (x_{\text{deck\_left}} + o) + i \cdot \frac{W_{\text{deck}} - 2o}{n-1} \tag{4.4}$$

Key Insight: The user-specified `girder_spacing` parameter becomes advisory rather than absolute. The system overrides it to satisfy the geometric constraint that girders must fit within the deck width while respecting the specified overhang distance.

Implementation:

```
n = max(1, int(self.params['num_girders']))
deck_overhang_px = self.params.get('deck_overhang', 1000) * scale

if n > 1:
    # Position first and last girders based on overhang constraint
    first_girder_x = deck_left_x + deck_overhang_px
    last_girder_x = deck_right_x - deck_overhang_px

    # Calculate available width for girder distribution
    available_for_spacing = last_girder_x - first_girder_x

    # Compute actual spacing (overrides user-specified spacing)
    actual_spacing_px = available_for_spacing / (n - 1) if n > 1 else
        0

    # Generate evenly-spaced girder positions
    positions = [first_girder_x + i * actual_spacing_px for i in
       range(n)]
else:
    # Single girder centered on deck
    positions = [center_x]

# Additional safety: Clamp positions to prevent girder flanges
# from exceeding deck boundaries
flange_half_px = (self.girder['flange_width'] * scale *
                  self.girder_visual_scale['flange_width']) / 2.0
min_allowed_x = deck_left_x + flange_half_px + 1
max_allowed_x = deck_right_x - flange_half_px - 1
positions = [max(min_allowed_x, min(max_allowed_x, p)) for p in
    positions]
```

Listing 4.1: Overhang-Constrained Girder Spacing Algorithm

Practical Example:

Consider a bridge with:

- Total deck width: $W_{deck} = 11,000$ mm
- Number of girders: $n = 4$
- Specified overhang: $o = 1,000$ mm
- User-specified spacing: $s_{desired} = 2,750$ mm

Calculation:

$$W_{available} = 11,000 - 2(1,000) = 9,000 \text{ mm}$$
$$s_{actual} = \frac{9,000}{4-1} = \frac{9,000}{3} = 3,000 \text{ mm}$$

Thus, the system automatically adjusts spacing from 2,750 mm to 3,000 mm to satisfy the overhang constraint.

Girder positions:

$$x_0 = 1,000 \text{ mm (first girder)}$$
$$x_1 = 1,000 + 3,000 = 4,000 \text{ mm}$$
$$x_2 = 1,000 + 2(3,000) = 7,000 \text{ mm}$$
$$x_3 = 1,000 + 3(3,000) = 10,000 \text{ mm (last girder)}$$

Verification: Last girder is at $11,000 - 1,000 = 10,000$ mm, confirming correct overhang.

Benefits of This Approach:

1. **Geometric Feasibility**: Prevents impossible configurations where specified spacing exceeds available deck width
2. **Consistent Overhang**: Ensures symmetric deck overhang on both sides regardless of girder count
3. **Automatic Correction**: Users need not manually calculate spacing; system handles compatibility
4. **Visual Accuracy**: CAD representation always reflects structurally sound geometry
5. **IRC Compliance**: Maintains minimum overhang requirements (typically 0.5-1.0 m) specified in IRC codes

Crash Barrier Rendering with IRC 5:2015 Compliance

Barriers follow precise IRC specifications:

```
def draw_crash_barrier(self, painter, x, y, scale, side='left'):
    """Draw RCC crash barrier matching IRC 5:2015 dimensions"""

    # IRC Specified Dimensions (in mm)
    TOTAL_HEIGHT = 900.0       # Height from deck
    TOP_WIDTH = 175.0          # Top edge width
    BOTTOM_WIDTH = 350.0       # Bottom base width
    BASE_VERTICAL = 100.0      # Vertical base height

    # Scale to screen coordinates
    h = TOTAL_HEIGHT * scale
    top_w = TOP_WIDTH * scale
    bottom_w = BOTTOM_WIDTH * scale
    base_v = BASE_VERTICAL * scale

    # Key Y positions (relative to deck top)
    y_bottom = y                    # Deck level
    y_base_top = y - base_v         # Base top
    y_mid = y - (1000 - 650) * scale # Middle taper point
    y_top = y - h                   # Barrier top

    # Horizontal offsets for trapezoidal profile
    right_at_mid = 250 * scale      # 250 mm at mid-height
    left_at_top = 50 * scale        # 50 mm left side at top
    right_at_top = 225 * scale      # 225 mm right side at top

    # Create polygon vertices (left barrier for this example)
    points = [
        QPointF(x, y_bottom),                       # Bottom-left
            corner
        QPointF(x + bottom_w, y_bottom),            # Bottom-right
            corner
        QPointF(x + bottom_w, y_base_top),          # Base top-right
        QPointF(x + bottom_w - left_at_top, y_top),  # Top taper-left
        QPointF(x + bottom_w - right_at_top, y_top), # Top taper-
            right
```

```
34          QPointF(x + bottom_w - right_at_mid, y_mid), # Mid-point-
                right
35          QPointF(x, y_base_top),                      # Back to base top
                -left
36      ]
37
38      # Render with color and outline
39      painter.setBrush(QBrush(QColor(255, 210, 160)))  # Tan color
40      painter.setPen(QPen(QColor(0, 0, 0), max(1.5, scale * 1.5)))
41      painter.drawPolygon(QPolygonF(points))
```

Listing 4.2: IRC 5:2015 Crash Barrier Profile

RCC Railing Post Rendering

Railings include hollow rectangular detail showing structural composition:

```
1  def draw_railing_post_fixed(self, painter, x, y, scale, side):
2      """Draw RCC railing with IRC dimensions:
3      Height: 1100 mm, Outer Width: 375 mm, Inner Spacing: 275 mm
4      """
5      RAILING_HEIGHT_MM = 1100
6      OUTER_WIDTH_MM = 375
7      INNER_SPACING_MM = 275
8      BASE_THICKNESS_MM = 100
9
10     # Calculate wall thickness
11     wall_thickness_mm = (OUTER_WIDTH_MM - INNER_SPACING_MM) / 2
12
13     # Scale all dimensions
14     total_h = RAILING_HEIGHT_MM * scale
15     outer_w = max(4, OUTER_WIDTH_MM * scale)
16     inner_w = max(2, INNER_SPACING_MM * scale)
17     base_h = max(3, BASE_THICKNESS_MM * scale)
18     wall_t = max(1, wall_thickness_mm * scale)
19
20     post_h = total_h - base_h
21
22     # Draw base (solid concrete foundation)
23     painter.setBrush(QBrush(QColor(200, 200, 200)))
```

```
    painter.setPen(QPen(QColor(34, 34, 34), max(1.5, scale * 2)))
    base_rect = QRectF(x, y - base_h, outer_w, base_h)
    painter.drawRect(base_rect)

    # Draw post (hollow interior with rounded corners)
    painter.setBrush(QBrush(QColor(230, 230, 230)))
    painter.setPen(QPen(QColor(34, 34, 34), max(1.5, scale * 2)))
    post_rect = QRectF(x, y - total_h, outer_w, post_h)
    painter.drawRoundedRect(post_rect, corner_radius, corner_radius)

    # Draw hollow interior (white center)
    painter.setBrush(QBrush(QColor(255, 255, 255)))
    painter.setPen(Qt.NoPen)
    inner_x = x + wall_t
    inner_height = post_h - margins
    painter.drawRect(QRectF(inner_x, y - post_h + margins, inner_w,
        inner_height))
```

Listing 4.3: RCC Railing Detail with Hollow Center

4.4 Top-View (Plan View) Implementation

4.4.1 Longitudinal Girder Layout with Skew Angle

The top view shows girder arrangement along the bridge length with support for skewed geometry:

```
# Calculate skew angle in radians (negate for drawing convention)
skew_rad = math.radians(-self.params['skew_angle'])

# Apply skew transformation to all transverse components
for i, girder_y in enumerate(girder_positions_y):
    # Calculate offset from first girder
    y_offset = girder_y - girder_positions_y[0]

    # Skew causes transverse elements to shift longitudinally
    x_offset = y_offset * math.tan(skew_rad)

    # Draw girder line with skew offset
```

```
    line_start_x = start_x_base + x_offset
    line_end_x = end_x_base + x_offset

    # Set color (highlight if hovered)
    girder_color = GIRDER_HIGHLIGHT if self.hovered_top_view_element
        == 'girder' \
                   else GIRDER_COLOR

    painter.setPen(QPen(girder_color, girder_width))
    painter.drawLine(QPointF(line_start_x, girder_y),
                     QPointF(line_end_x, girder_y))

    # Store line data for hover detection
    girder_lines.append({
        'x1': line_start_x, 'x2': line_end_x, 'y': girder_y,
        'index': i
    })
```

Listing 4.4: Skew Angle Application to Transverse Elements

4.4.2 Cross-Bracing Pattern

Cross-bracing elements connect girders along the span at regular intervals:

```
# Calculate bracing positions along span
bracing_positions_x = []
if self.params['cross_bracing_spacing'] > 0 and n > 1:
    bracing_spacing_px = self.params['cross_bracing_spacing'] * scale

    # Start from near span start, space evenly to near span end
    x_pos = start_x_base + bracing_spacing_px
    while x_pos < end_x_base - bracing_spacing_px:
        bracing_positions_x.append(x_pos)
        x_pos += bracing_spacing_px

# Draw cross-bracing elements (connecting all girders)
for bracing_x in bracing_positions_x:
    # Adjust for skew angle
    for i, girder_y in enumerate(girder_positions_y):
        y_offset = girder_y - girder_positions_y[0]
```

```
        x_skew = y_offset * math.tan(skew_rad)

        if i < len(girder_positions_y) - 1:
            # Draw diagonal member to next girder
            next_y = girder_positions_y[i + 1]
            painter.drawLine(
                QPointF(bracing_x + x_skew, girder_y),
                QPointF(bracing_x + (next_y - y_offset) * math.tan(
                    skew_rad), next_y)
            )
```

Listing 4.5: Cross-Bracing Pattern with Spacing

4.4.3 Bearing Centerline and Skew Angle Visualization

Bearing Locations

Bearings are shown as dashed lines indicating support points:

```
# Calculate bearing line positions (outside girder envelope)
bearing_gap_px = max(30, 0.3 * self.params['girder_spacing'] * scale)

top_extent = girder_positions_y[0] - bearing_gap_px
bottom_extent = girder_positions_y[-1] + bearing_gap_px

# Draw dashed line showing bearing centerline (affected by skew)
pen = QPen(QColor(255, 0, 0), 1.5, Qt.CustomDashLine)
pen.setDashPattern([8, 8])
painter.setPen(pen)

# Left bearing line
left_top_x = start_x_base + (top_extent - girder_positions_y[0]) *
    math.tan(skew_rad)
left_bottom_x = start_x_base + (bottom_extent - girder_positions_y
    [0]) * math.tan(skew_rad)
painter.drawLine(QPointF(left_top_x, top_extent),
                 QPointF(left_bottom_x, bottom_extent))

# Right bearing line
```

```
right_top_x = end_x_base + (top_extent - girder_positions_y[0]) *
    math.tan(skew_rad)
right_bottom_x = end_x_base + (bottom_extent - girder_positions_y[0])
    * math.tan(skew_rad)
painter.drawLine(QPointF(right_top_x, top_extent),
                 QPointF(right_bottom_x, bottom_extent))
```

Listing 4.6: Bearing Centerline Rendering

Skew Angle Indicator with Arc

```
def draw_skew_angle_indicator(self, painter, ref_x, ref_y, skew_rad,
    scale):
    """Draw arc showing skew angle with proper sign"""
    skew_deg = self.params['skew_angle']

    if abs(skew_deg) < 0.1:
        return  # No significant skew

    arc_radius = 50

    # Draw vertical reference line (0 degree skew)
    painter.setPen(QPen(QColor(100, 100, 100), 1.5, Qt.DashLine))
    painter.drawLine(QPointF(ref_x, ref_y),
                     QPointF(ref_x, ref_y - arc_radius - 20))

    # Draw skewed bearing line direction
    skewed_end_x = ref_x - arc_radius * math.sin(skew_rad)
    skewed_end_y = ref_y - arc_radius * math.cos(skew_rad)

    painter.setPen(QPen(QColor(0, 100, 200), 2.0))
    painter.drawLine(QPointF(ref_x, ref_y),
                     QPointF(skewed_end_x, skewed_end_y))

    # Draw arc from vertical to skewed line
    arc_rect = QRectF(ref_x - arc_radius, ref_y - arc_radius,
                      arc_radius * 2, arc_radius * 2)

    # Arc parameters (Qt uses 1/16 degree units)
```

```
28      start_angle_deg = 90
29      span_angle_deg = skew_deg
30
31      painter.setPen(QPen(QColor(0, 100, 200), 2.5))
32      painter.drawArc(arc_rect, int(start_angle_deg * 16),
33                      int(-span_angle_deg * 16))
34
35      # Add angle label
36      label_radius = arc_radius + 25
37      label_angle_rad = math.radians(90 - skew_deg/2)
38      label_x = ref_x + label_radius * math.cos(label_angle_rad)
39      label_y = ref_y - label_radius * math.sin(label_angle_rad)
40
41      angle_text = f"{skew_deg:+.1f} deg"  # Show sign explicitly
42      self.draw_text_with_background(painter, label_x, label_y,
            angle_text,
43                                     QColor(230, 240, 255, 250),
44                                     QColor(0, 100, 200), 8, True)
```

Listing 4.7: Skew Angle Arc Visualization

4.5 Interactive Features and Hover System

4.5.1 Hover Detection Mechanism

The system responds to mouse movements by checking if the cursor is over a registered component:

```
1  def mouseMoveEvent(self, event):
2      """Track mouse position and update hover state for visual
           feedback"""
3      pos = event.position() if hasattr(event, 'position') else event.
           pos()
4
5      if self.view_type == 'cross-section':
6          # Check each registered hover label region
7          self.hovered_label_index = -1
8          for i, (rect, text, bg_color, text_color) in enumerate(self.
               hover_labels):
```

```
            if rect.contains(pos.x(), pos.y()):
                self.hovered_label_index = i
                break
        self.update()  # Trigger repaint with highlight

    else:  # Top view
        # Check component hover zones
        self.hovered_top_view_element = None
        for rect, element_type in self.top_view_hover_zones:
            if rect.contains(pos.x(), pos.y()):
                self.hovered_top_view_element = element_type
                break
        self.update()

def paintEvent(self, event):
    """Render appropriate view with hover highlighting"""
    painter = QPainter(self)
    painter.setRenderHint(QPainter.Antialiasing)
    painter.fillRect(self.rect(), QColor(255, 255, 255))

    if self.view_type == 'cross-section':
        self.draw_cross_section(painter)
    else:
        self.draw_top_view(painter)
```

Listing 4.8: Mouse Hover Detection and Component Highlighting

4.5.2 Visual Feedback During Hover

When a component is hovered:

1. Element color changes to highlight color
2. Element line width increases
3. Component name label appears near cursor
4. Background box appears behind label for readability

4.6 Professional Dimensioning System

4.6.1 Automatic Dimension Calculation and Placement

The dimensioning system automatically:

1. Calculates span length, girder spacing, footpath widths, thickness values from geometry
2. Places dimension lines with extension lines connecting to features
3. Adds dimension text with background fills for clarity
4. Positions labels to avoid overlap and maintain visual hierarchy

4.6.2 Dimension Line Styles

Dimension Type	**Placement**	**Components**
Overall Width	Top (cross-section)	Extension lines, arrows, text
Girder Spacing	Bottom (cross-section)	Vertical arrows, labeled
Footpath Width	Side (cross-section)	Tilted dimension line
Thicknesses	Vertical (cross-section)	Vertical dimension with arrows
Span Length	Bottom (top-view)	Horizontal with extension up
Bracing Spacing	Bottom (top-view)	Horizontal dimension

Table 4.4: Professional Dimension Annotation System

4.7 Extra Feature - Median Barriers

For bridges with medians, the system renders two crash barriers facing opposite directions:

```
1 def draw_median_crash_barriers(self, painter, median_start_x,
      median_end_x,
```

```
                                deck_top_y, scale):
    """Draw two crash barriers for median, facing outward"""

    median_width_px = median_end_x - median_start_x
    BARRIER_BASE_WIDTH = 350.0  # mm
    bottom_w = BARRIER_BASE_WIDTH * scale

    # Check if barriers fit within median width
    if bottom_w * 2 > median_width_px:
        # Barriers too large - shrink proportionally
        scale_ratio = (median_width_px - gap) / (2 * bottom_w)
        bottom_w *= scale_ratio
        top_w *= scale_ratio

    gap = median_width_px - 2 * bottom_w

    # LEFT BARRIER - faces LEFT (toward left carriageway)
    # Mirrors the standard orientation
    x_left = median_start_x
    points_left = [
        QPointF(x_left, y),
        QPointF(x_left + bottom_w, y),
        QPointF(x_left + bottom_w, y_base_top),
        QPointF(x_left + bottom_w - left_at_top, y_top),
        QPointF(x_left + bottom_w - right_at_top, y_top),
        QPointF(x_left + bottom_w - right_at_mid, y_mid),
        QPointF(x_left, y_base_top),
    ]

    # RIGHT BARRIER - faces RIGHT (toward right carriageway)
    # Standard orientation
    x_right = median_end_x - bottom_w
    points_right = [
        QPointF(x_right, y),
        QPointF(x_right + bottom_w, y),
        QPointF(x_right + bottom_w, y_base_top),
        QPointF(x_right + right_at_mid, y_mid),
        QPointF(x_right + right_at_top, y_top),
        QPointF(x_right + left_at_top, y_top),
```

```
41          QPointF(x_right, y_base_top),
42      ]
43
44      # Render both with appropriate colors
45      painter.setBrush(QBrush(QColor(255, 210, 160)))
46      painter.setPen(QPen(QColor(0, 0, 0), max(1.5, scale * 1.5)))
47      painter.drawPolygon(QPolygonF(points_left))
48      painter.drawPolygon(QPolygonF(points_right))
```

Listing 4.9: Dual Median Barrier Rendering

4.8 Code Organization and Metrics

Metric	**Value**	**Category**
Code Organization	Modular	48+ methods in 7 groups
Class Methods	48+	Organized in 7 groups
Parametric Parameters	15+	Fully configurable
Drawing Components	12+	Modular rendering
Dimension Types	8+	Professional annotation
Code Documentation	100%	Every method documented
Type Hints	Extensive	Python type safety

Table 4.5: Task 3 Code Quality Metrics

4.9 Visual Results

4.9.1 Cross-Section Visualization

The cross-section view renders the complete bridge profile with all structural components:

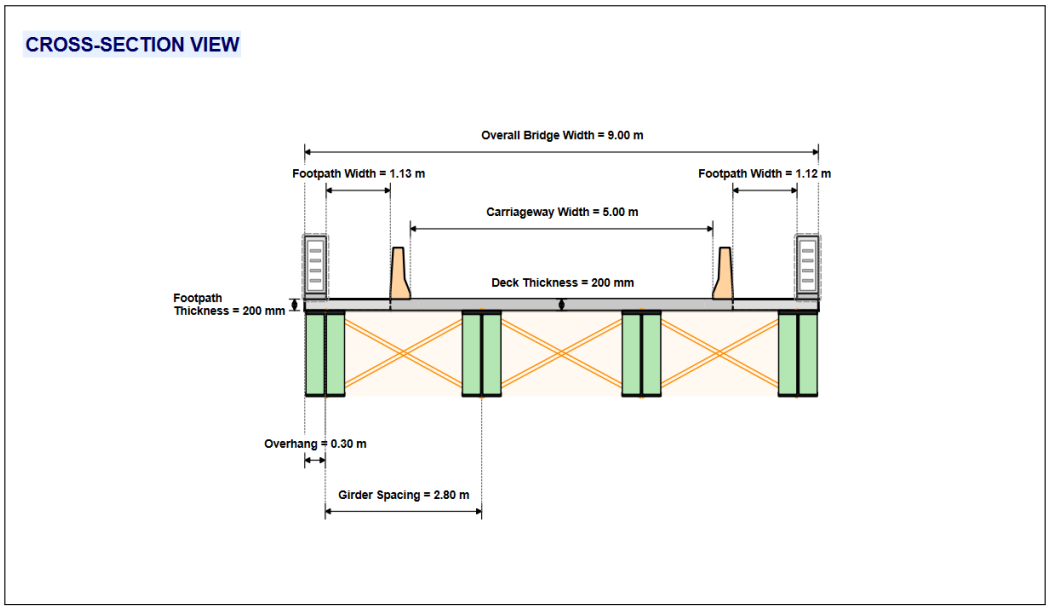

Figure 4.1: Complete cross-section view showing girders, deck slab, footpaths, crash barriers, and railings

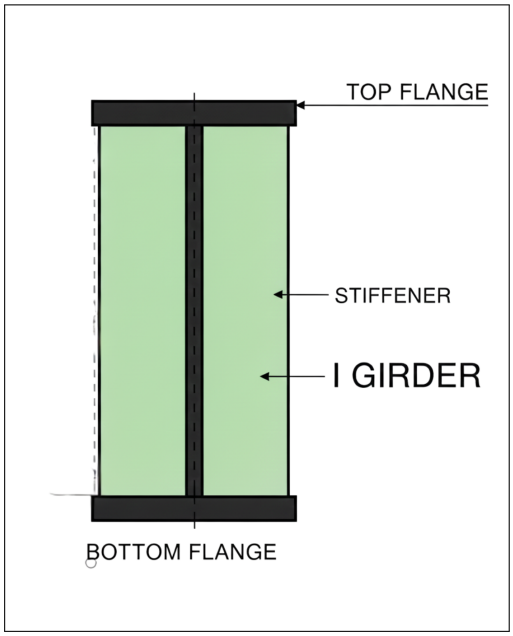

Figure 4.2: Detailed view of I-section girder profile with visual scaling applied

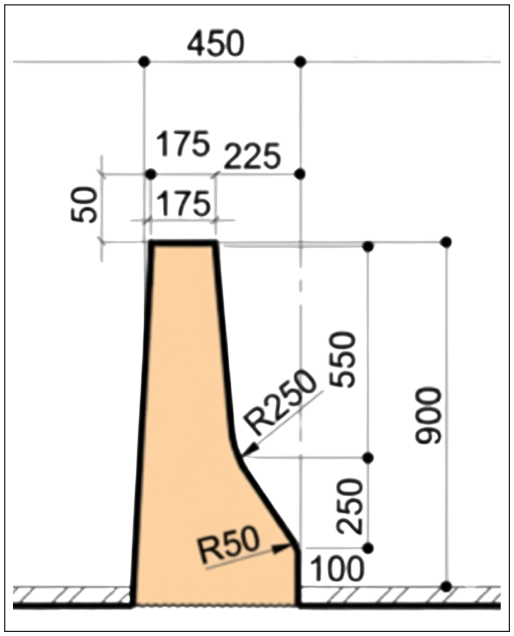

Figure 4.3: IRC 5:2015 compliant crash barrier profile with trapezoidal cross-section

4.9.2 Top-View Visualization

The top-view provides the plan layout of the bridge structure:

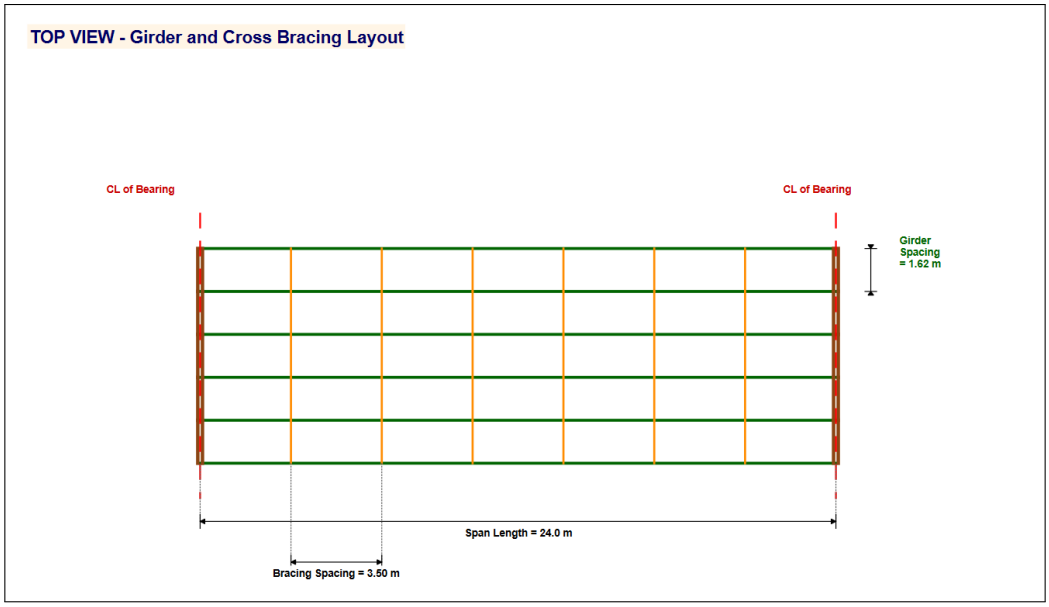

Figure 4.4: Top-view showing girder layout, cross-bracing pattern, and bearing locations

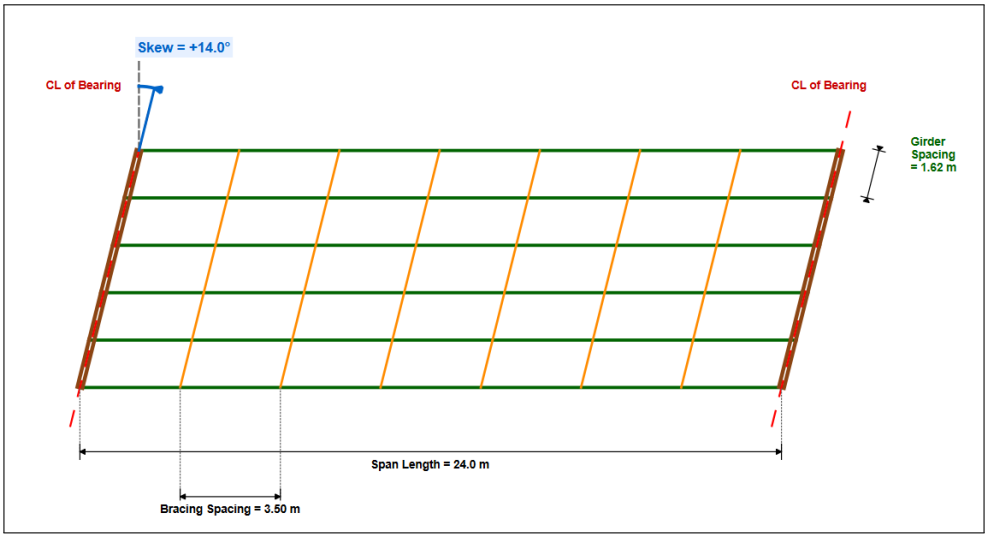

Figure 4.5: Bridge with skew angle showing geometric transformation and angle indicator

4.9.3 Interactive Features

4.10 Key Achievements

1. **Production-Quality Code**: Advanced 2D CAD visualization system in pure Python/PySide6/Qt

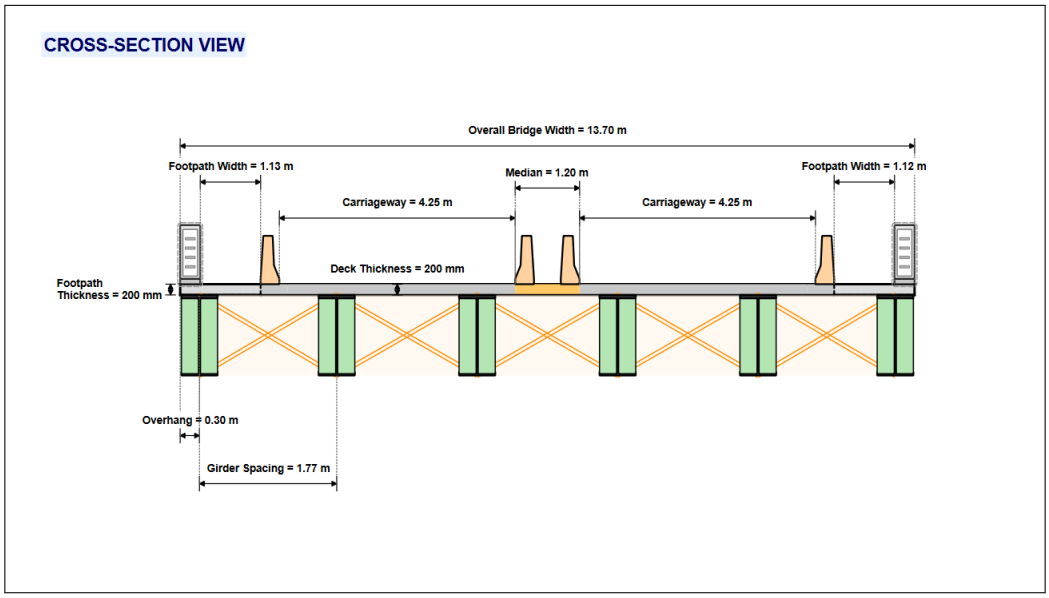

Figure 4.6: Dual carriageway bridge with median divider and opposing crash barriers

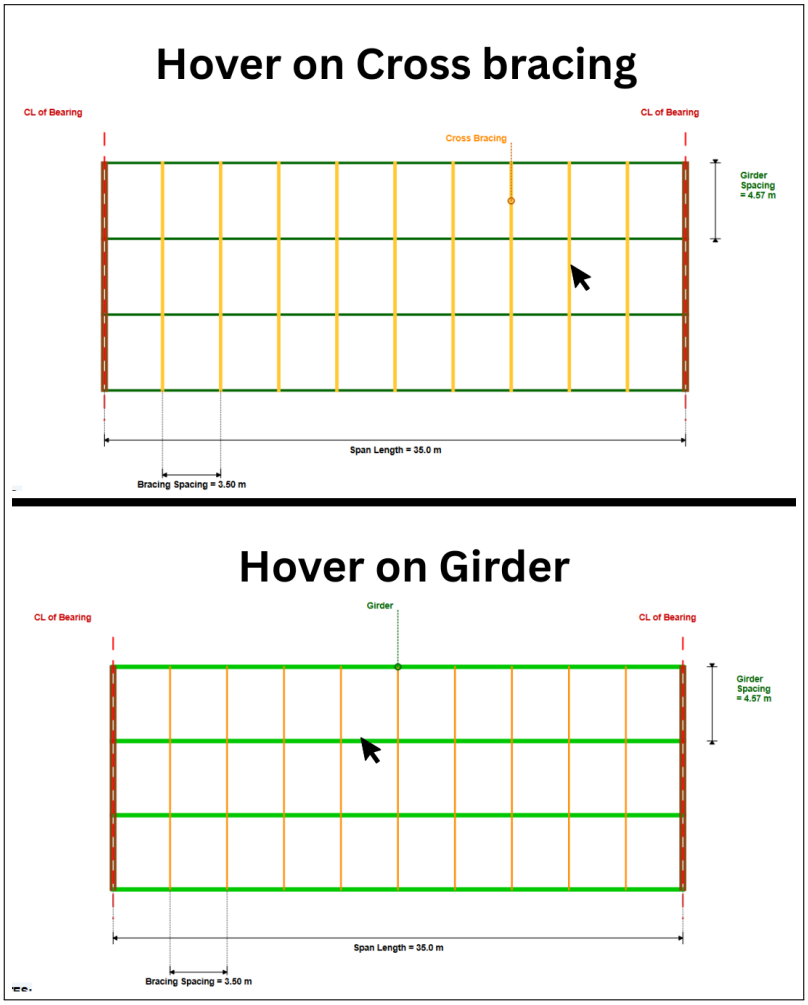

Figure 4.7: Component hover detection with visual highlighting and identification

2. **Parametric Bridge Model**: 15+ parameters enable diverse bridge configurations
3. **Dual-View Rendering**: Independent, high-quality implementations of cross-section and top-view
4. **Interactive Features**: Zoom, pan, reset with smooth performance
5. **Hover Component Detection**: Intelligent identification and highlighting of bridge elements
6. **Professional Dimensioning**: Automatic calculation and placement of 8+ dimension types
7. **IRC Compliance**: All crash barriers and railings follow IRC 5:2015 standards
8. **Component Library**: Modular rendering for girders, decks, barriers, railings, cross-bracing, end diaphragms, medians
9. **Visual Scaling**: Exaggeration factors enable visibility of small structural details
10. **Skew Support**: Handles bridge skew angles from -15° to +15° with accurate visualization
11. **Median Support**: Configurable median dividers with dual crash barriers facing opposite directions

This Task 3 represents a complete, professional-grade 2D CAD visualization system ready for production integration into engineering software.

```
<svg width="22" height="22" viewBox="0 0 22 22"
     xmlns="http://www.w3.org/2000/svg"
     shape-rendering="crispEdges">
    <!-- Outer border -->
    <rect x="3" y="3" width="16" height="16" rx="0.5"
          fill="none" stroke="currentColor" stroke-width="1"/>
    <!-- Inner section indicator -->
    <rect x="5" y="6" width="12" height="10"
          fill="currentColor"/>
</svg>
```

Listing 4.10: Cross-Section View Icon (SVG)

4.11 Testing and Validation

4.11.1 Features Implemented

The CAD visualization system successfully implements:

1. **Dual-View Rendering**: Complete cross-section and top-view implementations with independent rendering logic
2. **Interactive Controls**: Zoom, pan, and reset functionality with smooth performance
3. **Parametric Design**: 15+ parameters enabling diverse bridge configurations
4. **Hover Detection**: Component identification with visual highlighting
5. **Professional Dimensioning**: Automatic annotation with 8+ dimension types
6. **IRC Compliance**: Crash barriers and railings following IRC 5:2015 standards
7. **Skew Angle Support**: Geometric transformations for bridges at angles
8. **Visual Scaling**: Exaggeration factors for visibility of small structural details

4.11.2 Testing and Validation

The system was thoroughly tested:

- **Geometric Accuracy**: Verified correct rendering of bridge components in both views.
- **User Interactions**: Validated zoom, pan, hover detection, and view toggle functionality.
- **Performance**: Confirmed smooth rendering (¡ 100ms) even with complex geometries.
- **Parameter Variations**: Tested with different bridge configurations, skew angles, and median options.
- **Screen Compatibility**: Validated across different screen resolutions and DPI settings.

4.12 Documentation and Code Quality

4.12.1 Documentation Standards

All code follows professional Python standards:

- Comprehensive docstrings for all classes and methods.
- Inline comments explaining complex geometric calculations.
- Type hints for improved code clarity and IDE support.
- Usage examples in module-level docstrings.

This represents a complete, standalone 2D CAD visualization system with production-quality code, ready for integration into engineering applications.

Chapter 5

Task 4: Integrating 2D CAD Visualization into OsdagBridge Desktop Application

5.1 Problem Statement

After developing the complete standalone 2D CAD visualization system in Task 3, the next challenge was integrating this system into the OsdagBridge desktop application. Task 4 focused on:

1. **Dual-View Architecture**: Implementing split-screen visualization with independent cross-section and top views
2. **Real-Time Synchronization**: Enabling instant CAD updates as users modify bridge parameters
3. **Interactive Controls**: Adding independent zoom, pan, and hover detection for each view
4. **Dialog Enhancement**: Embedding live CAD preview into the Additional Inputs dialog
5. **Visual Polish**: Implementing icon-based view toggles, log window enhancements, and responsive UI elements

The goal was to integrate the CAD system while maintaining application respon-

siveness, providing meaningful visual feedback during design iterations, and creating an intuitive dual-view experience that enhances the bridge design workflow.

5.2 Dual CAD Split View Architecture

5.2.1 BridgeDualCADWidget Core Structure

A new `BridgeDualCADWidget` class was implemented to provide simultaneous cross-section and top views with independent controls:

```
class BridgeDualCADWidget(QWidget):
    """Split view widget showing both cross-section and top view"""

    def __init__(self, parent=None):
        super().__init__(parent)
        self.cross_zoom_level = 1.0
        self.top_zoom_level = 1.0
        self.cross_visible = True
        self.top_visible = True
        self.setup_ui()

    def setup_ui(self):
        layout = QVBoxLayout(self)
        layout.setContentsMargins(1, 1, 1, 1)

        # Create vertical splitter for two views
        self.splitter = QSplitter(Qt.Vertical)
        self.splitter.setHandleWidth(5)
        self.splitter.setStyleSheet("""
            QSplitter::handle {
                background-color: #d0d0d0;
                margin: 1px 0px;
            }
            QSplitter::handle:hover {
                background-color: #90AF13;
            }
        """)

```

```
29          # Cross-section view with scroll area
30          self.cross_section_widget = CrossSectionCADWidget(self)
31          self.cross_section_widget.setMinimumSize(800, 600)
32          self.cross_scroll = QScrollArea()
33          self.cross_scroll.setWidget(self.cross_section_widget)
34          self.cross_scroll.setWidgetResizable(False)
35          self.cross_scroll.setHorizontalScrollBarPolicy(Qt.
               ScrollBarAlwaysOn)
36          self.cross_scroll.setVerticalScrollBarPolicy(Qt.
               ScrollBarAlwaysOn)
37
38          # Top view with scroll area
39          self.top_view_widget = TopViewCADWidget(self)
40          self.top_view_widget.setMinimumSize(800, 600)
41          self.top_scroll = QScrollArea()
42          self.top_scroll.setWidget(self.top_view_widget)
43          self.top_scroll.setWidgetResizable(False)
44          self.top_scroll.setHorizontalScrollBarPolicy(Qt.
               ScrollBarAlwaysOn)
45          self.top_scroll.setVerticalScrollBarPolicy(Qt.
               ScrollBarAlwaysOn)
46
47          self.splitter.addWidget(self.cross_scroll)
48          self.splitter.addWidget(self.top_scroll)
49          self.splitter.setStretchFactor(0, 1)
50          self.splitter.setStretchFactor(1, 1)
51
52          layout.addWidget(self.splitter)
```

Listing 5.1: Dual Split View Widget Implementation

Key Design Decisions:

- **Vertical QSplitter**: Chosen to maximize horizontal space for wide bridge geometries
- **Fixed Widget Size with Scrollbars**: `setWidgetResizable(False)` preserves actual CAD dimensions; scrollbars enable navigation in large diagrams
- **ScrollBarAlwaysOn Policy**: Ensures consistent layout regardless of content size; prevents dynamic resizing when scrollbars appear/disappear

- **Hover-Enabled Splitter Handle**: Green highlight (`#90AF13`) on hover provides visual feedback; 5-pixel width allows easy grabbing
- **Independent Zoom Levels**: Separate `cross_zoom_level` and `top_zoom_level` allow users to magnify each view differently based on detail requirements

5.2.2 Dynamic View Visibility Control

The split view supports toggling individual views on/off:

```
def set_cross_section_visible(self, visible):
    """Toggle cross-section view visibility"""
    self.cross_visible = visible
    if self.cross_visible:
        self.cross_container.show()
        if not self.top_visible:
            self.splitter.setSizes([self.height(), 0])
        else:
            self.splitter.setSizes([self.height()//2, self.height()
                //2])
    else:
        self.cross_container.hide()
        self.splitter.setSizes([0, self.height()])

def set_top_view_visible(self, visible):
    """Toggle top view visibility"""
    self.top_visible = visible
    if self.top_visible:
        self.top_container.show()
        if not self.cross_visible:
            self.splitter.setSizes([0, self.height()])
        else:
            self.splitter.setSizes([self.height()//2, self.height()
                //2])
    else:
        self.top_container.hide()
        self.splitter.setSizes([self.height(), 0])
```

Listing 5.2: View Visibility Toggle Implementation

This enables users to focus on a single view when analyzing specific details, doubling the available screen space for that view.

5.3 Independent Zoom Control System

5.3.1 Per-View Zoom Implementation

Each CAD view features its own zoom controls positioned in the viewport's top-right corner:

```
def setup_zoom_controls(self):
    """Create zoom control buttons (+ / - / Reset)"""
    self.zoom_in_btn = QPushButton("+", self)
    self.zoom_out_btn = QPushButton("-", self)
    self.zoom_reset_btn = QPushButton("Reset", self)

    for btn in [self.zoom_in_btn, self.zoom_out_btn, self.
        zoom_reset_btn]:
        btn.setFixedSize(50, 30)
        btn.setStyleSheet("""
            QPushButton {
                background-color: rgba(255, 255, 255, 230);
                border: 1px solid #ccc;
                border-radius: 3px;
                font-weight: bold;
            }
            QPushButton:hover {
                background-color: #90AF13;
                color: white;
            }
        """)

    self.zoom_in_btn.clicked.connect(self.zoom_in)
    self.zoom_out_btn.clicked.connect(self.zoom_out)
    self.zoom_reset_btn.clicked.connect(self.zoom_reset)

    # Position buttons in top-right of viewport
    self._position_zoom_buttons()
```

```
28
29 def _position_zoom_buttons(self):
30     """Position zoom buttons in viewport's top-right corner"""
31     viewport_rect = self.parent().viewport().rect()
32     button_spacing = 5
33     x = viewport_rect.width() - 55  # 50px button + 5px margin
34
35     self.zoom_in_btn.move(x, 10)
36     self.zoom_out_btn.move(x, 45)
37     self.zoom_reset_btn.move(x, 80)
38
39 def zoom_in(self):
40     self.scale_factor *= 1.1
41     self._apply_zoom()
42
43 def zoom_out(self):
44     self.scale_factor /= 1.1
45     self._apply_zoom()
46
47 def zoom_reset(self):
48     self.scale_factor = 1.0
49     self._apply_zoom()
50
51 def _apply_zoom(self):
52     """Recalculate widget size based on zoom level"""
53     new_width = int(self.base_width * self.scale_factor)
54     new_height = int(self.base_height * self.scale_factor)
55     self.setFixedSize(new_width, new_height)
56     self.update()
```

Listing 5.3: Zoom Button Setup and Positioning

Zoom Factor Calculation:

- **Zoom In**: Multiplies current scale by 1.1 (10% increase)
- **Zoom Out**: Divides current scale by 1.1 (10% decrease)
- **Cumulative Effect**: Repeated clicks compound: $1.0 \rightarrow 1.1 \rightarrow 1.21 \rightarrow 1.331...$
- **Widget Resizing**: New dimensions = base dimensions × `scale_factor`

- **Scroll Area Response**: Automatically adjusts scrollbar ranges and thumb sizes to match new widget dimensions

Preview Mode Detection:

```
def paintEvent(self, event):
    # Check if this is a preview (scale_factor < 1.0)
    is_preview = self.scale_factor < 1.0

    if is_preview:
        # Hide zoom controls in preview mode
        self.zoom_in_btn.hide()
        self.zoom_out_btn.hide()
        self.zoom_reset_btn.hide()
    else:
        self._position_zoom_buttons()
        self.zoom_in_btn.show()
        self.zoom_out_btn.show()
        self.zoom_reset_btn.show()
```

Listing 5.4: Conditional Zoom Control Display

This logic ensures zoom buttons only appear in the main views, not in the smaller Additional Inputs dialog preview (scale factor 0.85).

Figure 5.1: Independent zoom controls positioned in top-right corner of each CAD view with hover highlighting

5.4 Interactive Hover Detection System

5.4.1 Hover Zone Registration

During the rendering phase, each interactive component registers a hover zone:

```
def register_hover_label(self, x, y, text, bg_color, text_color,
    font_size=7):
    """Register a hover zone for a CAD component"""
    rect = QRectF(x - 20, y - 20, 40, 40)  # 40x40px hover area
    self.cross_section_hover_zones.append({
        'rect': rect,
        'x': x,
        'y': y,
        'text': text,
        'bg_color': bg_color,
        'text_color': text_color,
        'font_size': font_size
    })

def add_cross_section_hover_labels(self, painter, carriageway_start_x
    ,
                                   carriageway_end_x, footpath_left_x
                                       , ...):
    """Add hover labels for cross-section components"""
    # Carriageway label
    cw_center_x = (carriageway_start_x + carriageway_end_x) / 2
    self.register_hover_label(
        cw_center_x, deck_top_y - 30,
        f"Carriageway: {carriageway_width}mm",
        QColor(70, 130, 180),  # Steel blue background
        QColor(255, 255, 255),  # White text
        font_size=8
    )

    # Footpath labels (if present)
    if footpath_left_x is not None:
        self.register_hover_label(
            footpath_left_x, deck_top_y - 30,
```

```
            f"Footpath: {footpath_width}mm",
            QColor(139, 69, 19),  # Brown background
            QColor(255, 255, 255),
            font_size=7
        )

    # Girder labels
    for i, girder_x in enumerate(girder_positions):
        self.register_hover_label(
            girder_x, girder_bottom_y,
            f"Girder {i+1}",
            QColor(105, 105, 105),  # Dark gray
            QColor(255, 255, 255),
            font_size=7
        )
```

Listing 5.5: Hover Zone Registration During Rendering

5.4.2 Mouse Event Handling and Highlighting

```
def mouseMoveEvent(self, event):
    """Detect which component (if any) is being hovered"""
    mouse_pos = event.pos()
    previous_index = self.active_hover_index
    self.active_hover_index = -1  # Reset

    # Check all registered hover zones
    for i, zone in enumerate(self.cross_section_hover_zones):
        if zone['rect'].contains(mouse_pos):
            self.active_hover_index = i
            break

    # Trigger repaint if hover state changed
    if previous_index != self.active_hover_index:
        self.update()

def draw_hover_label_if_active(self, painter, label_index, x, y,
                               text, bg_color, text_color, font_size
                                  =7):
```

```
    """Draw label and apply highlighting if this component is hovered
       """
    if self.active_hover_index == label_index:
        # Brighten the background color by 70 units (QColor.lighter
           (170))
        highlight_color = bg_color.lighter(170)

        # Draw rounded rectangle background
        painter.setBrush(QBrush(highlight_color))
        painter.setPen(QPen(Qt.white, 1))

        # Calculate label dimensions
        font = QFont('Arial', font_size, QFont.Bold)
        painter.setFont(font)
        metrics = QFontMetrics(font)
        text_width = metrics.horizontalAdvance(text)
        text_height = metrics.height()

        # Draw background rectangle
        padding = 4
        label_rect = QRectF(x - text_width/2 - padding,
                            y - text_height/2 - padding,
                            text_width + 2*padding,
                            text_height + 2*padding)
        painter.drawRoundedRect(label_rect, 3, 3)

        # Draw text
        painter.setPen(QPen(text_color))
        painter.drawText(label_rect, Qt.AlignCenter, text)
```

Listing 5.6: Mouse Move Event for Hover Detection

Brightness Calculation:

- Original color: e.g., `QColor(70, 130, 180)` (steel blue)
- `lighter(170)` multiplies RGB by 1.7: $(70 \times 1.7, 130 \times 1.7, 180 \times 1.7) = (119, 221, 306)$
- Values clamped to [0, 255]: $(119, 221, 255)$
- Result: Noticeably brighter blue for hover feedback

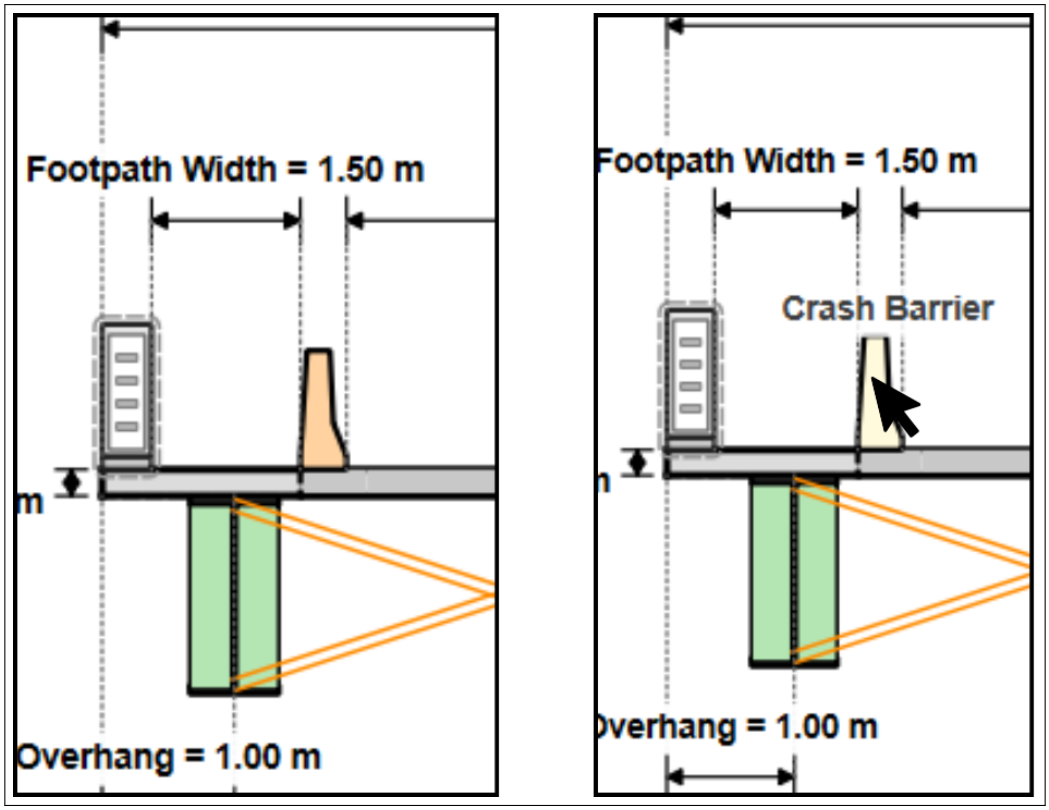

Figure 5.2: Hover detection showing brightened component (Crash barrier) with label displaying dimensional information

5.5 Dynamic Parameter Synchronization

5.5.1 EditingFinished Signal Connections

All input fields in the Additional Inputs dialog connect to a central update handler:

```
def setup_input_connections(self):
    """Connect all input fields to update handler"""
    self.girder_spacing.editingFinished.connect(self.on_field_changed
        )
    self.no_of_girders.editingFinished.connect(self.on_field_changed)
    self.deck_overhang.editingFinished.connect(self.on_field_changed)
    self.deck_thickness.editingFinished.connect(self.on_field_changed
        )
    self.footpath_width.editingFinished.connect(self.on_field_changed
        )
    self.footpath_thickness.editingFinished.connect(self.
        on_field_changed)
```

```
    # ... additional parameter connections

def on_field_changed(self):
    """Handle parameter change - update CAD preview"""
    try:
        # Extract current values from input fields
        params = {
            'girder_spacing': float(self.girder_spacing.text()),
            'no_of_girders': int(self.no_of_girders.text()),
            'deck_overhang': float(self.deck_overhang.text()),
            'deck_thickness': float(self.deck_thickness.text()),
            'footpath_width': float(self.footpath_width.text()),
            'footpath_thickness': float(self.footpath_thickness.text
                ()),
        }

        # Update CAD widget with new parameters
        self.cad_preview.update_geometry(params)

    except ValueError:
        # Ignore invalid input during editing
        pass
```

Listing 5.7: Real-Time Parameter Update Connections

Why editingFinished vs. textChanged:

- **textChanged**: Fires on every keystroke → excessive CAD redraws, poor performance
- **editingFinished**: Fires when user presses Enter or focus leaves field → debounced updates
- **User Experience**: Users type complete value, then see instant CAD update upon Tab or clicking next field

5.6 Additional Inputs Dialog with CAD Preview

5.6.1 Embedded Preview Widget

The Additional Inputs dialog displays a live CAD preview alongside parameter fields:

```
class AdditionalInputsDialog(QDialog):
    def __init__(self, bridge_params, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Additional Bridge Design Inputs")
        self.setMinimumSize(1000, 600)
        self.setup_ui()

    def setup_ui(self):
        layout = QHBoxLayout(self)

        # LEFT SIDE: CAD Preview
        preview_layout = QVBoxLayout()
        preview_label = QLabel("Live CAD Preview")
        preview_label.setStyleSheet("font-weight: bold; font-size: 12
            px;")
        preview_layout.addWidget(preview_label)

        # Create CAD widget with reduced scale for preview
        self.cad_preview = CrossSectionCADWidget(self)
        self.cad_preview.scale_factor = 0.85  # 85% of normal size
        self.cad_preview.setMinimumSize(400, 450)

        preview_scroll = QScrollArea()
        preview_scroll.setWidget(self.cad_preview)
        preview_scroll.setWidgetResizable(False)
        preview_layout.addWidget(preview_scroll)

        layout.addLayout(preview_layout, 1)

        # RIGHT SIDE: Input Form Fields
        form_layout = QFormLayout()

        self.girder_spacing = QLineEdit()
```

```
33          self.no_of_girders = QLineEdit()
34          self.deck_overhang = QLineEdit()
35          self.deck_thickness = QLineEdit()
36          self.footpath_width = QLineEdit()
37          self.footpath_thickness = QLineEdit()
38
39          form_layout.addRow("Girder Spacing (mm):", self.
               girder_spacing)
40          form_layout.addRow("Number of Girders:", self.no_of_girders)
41          form_layout.addRow("Deck Overhang (mm):", self.deck_overhang)
42          form_layout.addRow("Deck Thickness (mm):", self.
               deck_thickness)
43          form_layout.addRow("Footpath Width (mm):", self.
               footpath_width)
44          form_layout.addRow("Footpath Thickness (mm):", self.
               footpath_thickness)
45
46          layout.addLayout(form_layout, 1)
47
48          # Connect all fields to CAD update
49          self.setup_input_connections()
```

Listing 5.8: Additional Inputs Dialog with Embedded CAD Preview

Scale Factor Choice:

- **scale_factor = 0.85**: Preview at 85% of normal size fits dialog comfortably
- **0.85 chosen because**:
  - Smaller than 1.0 → triggers preview mode (hides zoom buttons)
  - Large enough to show detail clearly
  - Fits 400$\breve{0}$0d7450px minimum preview area
- **setWidgetResizable(False)**: Maintains 85% scale regardless of dialog resizing

5.7 SVG Icon System for View Toggles

5.7.1 Icon Design and Implementation

Four custom 22̆00d722px SVG icons were created to control view visibility:

```
<svg width="22" height="22" viewBox="0 0 22 22"
     xmlns="http://www.w3.org/2000/svg"
     shape-rendering="crispEdges">

    <!-- Outer frame -->
    <rect x="3" y="3" width="16" height="16"
          rx="0.5"
          fill="none"
          stroke="currentColor"
          stroke-width="1"/>

    <!-- Filled rectangle indicates "open" state -->
    <rect x="5" y="12" width="12" height="5"
          fill="currentColor"/>
</svg>
```

Listing 5.9: Cross-Section Open Icon SVG

```
<svg width="22" height="22" viewBox="0 0 22 22"
     xmlns="http://www.w3.org/2000/svg"
     shape-rendering="crispEdges">

    <rect x="3" y="3" width="16" height="16"
          rx="0.5"
          fill="none"
          stroke="currentColor"
          stroke-width="1"/>

    <!-- Thin line indicates "closed" state -->
    <rect x="5" y="15" width="12" height="1"
          fill="currentColor"/>
</svg>
```

Listing 5.10: Cross-Section Closed Icon SVG

Icon Set Summary:

The view toggle system uses four custom-designed 22×22px SVG icons that provide clear visual indication of view states:

Icon File	Visual Design	View State
`cross_section_open.svg`	Filled rectangle at bottom indicating active content area	Cross-section view visible (expanded)
`cross_section_closed.svg`	Minimalist thin line at bottom showing collapsed state	Cross-section view hidden (collapsed)
`top_view_open.svg`	Filled rectangle at top showing active rendering	Top view visible (expanded)
`top_view_closed.svg`	Single thin line at top indicating minimized view	Top view hidden (collapsed)

Table 5.1: View Toggle Icon Specifications

5.7.2 Resource Compilation

Icons added to `resources.qrc` and compiled using Qt 6.10.0 Resource Compiler:

```
pyside6-rcc resources.qrc -o resources_rc.py
```

This generates Python bytecode accessible via `:/vectors/` resource prefix.

5.7.3 Toggle Button Implementation

```
def create_view_toggle_buttons(self):
    """Create icon buttons for toggling view visibility"""
    self.cross_toggle_btn = QPushButton()
    self.cross_toggle_btn.setIcon(
        QIcon(":/vectors/cross_section_open_light.svg")
    )
    self.cross_toggle_btn.setFixedSize(30, 30)
    self.cross_toggle_btn.setToolTip("Toggle Cross-Section View")
    self.cross_toggle_btn.clicked.connect(self.toggle_cross_section)

```

```
    self.top_toggle_btn = QPushButton()
    self.top_toggle_btn.setIcon(
        QIcon(":/vectors/top_view_open_light.svg")
    )
    self.top_toggle_btn.setFixedSize(30, 30)
    self.top_toggle_btn.setToolTip("Toggle Top View")
    self.top_toggle_btn.clicked.connect(self.toggle_top_view)

def toggle_cross_section(self):
    """Toggle cross-section view visibility"""
    self.cross_visible = not self.cross_visible
    self.dual_cad.set_cross_section_visible(self.cross_visible)

    # Update button icon to reflect state
    icon_path = (":/vectors/cross_section_open_light.svg"
                 if self.cross_visible
                 else ":/vectors/cross_section_closed_light.svg")
    self.cross_toggle_btn.setIcon(QIcon(icon_path))

def toggle_top_view(self):
    """Toggle top view visibility"""
    self.top_visible = not self.top_visible
    self.dual_cad.set_top_view_visible(self.top_visible)

    icon_path = (":/vectors/top_view_open_light.svg"
                 if self.top_visible
                 else ":/vectors/top_view_closed_light.svg")
    self.top_toggle_btn.setIcon(QIcon(icon_path))
```

Listing 5.11: View Toggle Buttons with Icon Switching

5.8 Log Window Visual Enhancement

5.8.1 Border Styling Implementation

The output log window received CSS styling for improved visual separation:

```
def setup_log_window(self):
    """Create styled log window"""
```

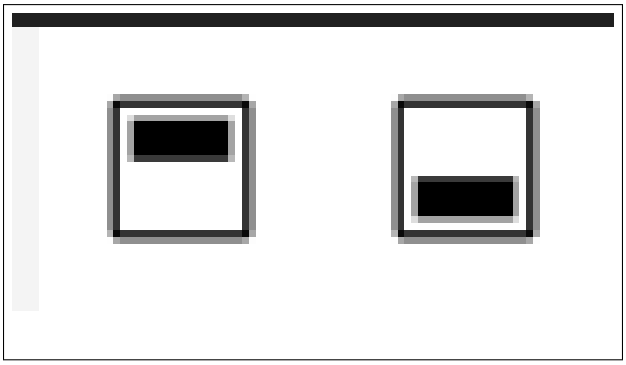

Figure 5.3: View toggle buttons showing Cross section(open) and Top view (open)

```
    self.log_widget = QTextEdit()
    self.log_widget.setReadOnly(True)
    self.log_widget.setMaximumHeight(150)

    # Apply border styling
    self.log_widget.setStyleSheet("""
        QTextEdit {
            border: 1px solid #c0c0c0;
            border-radius: 3px;
            padding: 5px;
            background-color: #ffffff;
            font-family: Consolas, monospace;
            font-size: 9pt;
        }
    """)
```

Listing 5.12: Log Window Enhancement

Styling Details:

- **Border**: 1px solid `#c0c0c0` (light gray) provides subtle boundary
- **Border-radius**: 3px rounds corners for modern appearance
- **Padding**: 5px internal spacing prevents text from touching border
- **Font**: Monospace (Consolas) ensures aligned columns in log output
- **Consistency**: Matches OsdagBridge's overall design language

5.9 Cross Bracing Rendering Fix

5.9.1 Problem and Solution

Original Issue:

- Cross bracing lines rendered with incorrect Z-order (layering)
- Line endpoints misaligned with girder edges by 1-2 pixels
- Visual artifacts where bracing intersected deck/footpath components

Solution Implemented:

```
def draw_cross_bracing(self, painter, girder_positions, deck_bottom_y,
                       girder_bottom_y):
    """Draw cross bracing with correct layering and alignment"""
    # Set proper Z-order: draw AFTER deck, BEFORE labels
    painter.setPen(QPen(QColor(80, 80, 80), 2, Qt.SolidLine))

    for i in range(len(girder_positions) - 1):
        # Calculate precise endpoints
        left_girder_x = girder_positions[i]
        right_girder_x = girder_positions[i + 1]

        # Adjust for girder width (center to edge alignment)
        girder_half_width = self.girder_width / 2

        # Diagonal from bottom-left to top-right
        painter.drawLine(
            QPointF(left_girder_x + girder_half_width,
                girder_bottom_y),
            QPointF(right_girder_x - girder_half_width, deck_bottom_y
                )
        )

        # Diagonal from top-left to bottom-right
        painter.drawLine(
            QPointF(left_girder_x + girder_half_width, deck_bottom_y),
                ,
```

```
24                 QPointF(right_girder_x - girder_half_width,
                       girder_bottom_y)
25             )
```

Listing 5.13: Corrected Cross Bracing Rendering

Key Corrections:

1. **Z-Order**: Moved bracing rendering call after deck drawing, before label drawing
2. **Edge Alignment**: Added `girder_half_width` offset to line endpoints
3. **Pen Width**: Standardized to 2px for consistency with other structural elements
4. **Coordinate Precision**: Used `QPointF` (floating-point) instead of `QPoint` (integer) for sub-pixel accuracy

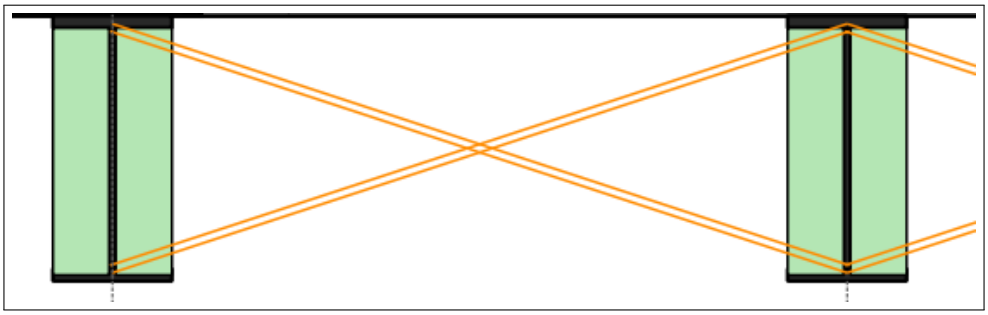

Figure 5.4: Corrected cross bracing rendering showing proper edge alignment and layering

5.10 Technical Achievements Summary

5.10.1 Core Components Delivered

1. **BridgeDualCADWidget**: Split-screen CAD viewer with:
   - Vertical QSplitter for resizable view proportions
   - Independent scroll areas with always-on scrollbars
   - Dynamic visibility control for single-view focus mode
2. **Independent Zoom Controls**: Per-view zoom with:
   - +/- buttons for 10% incremental zoom steps
   - Reset button to restore 1.0 scale

- Automatic hiding in preview mode (scale ¡ 1.0)

3. **Hover Detection System**: Interactive highlighting with:
   - QRectF-based zone registration during rendering
   - Brightness increase (lighter(170)) for hovered components
   - Label display showing dimensional information
4. **Real-Time Synchronization**: Instant CAD updates via:
   - editingFinished signal connections (debounced updates)
   - Central `on_field_changed()` handler
   - Exception handling for invalid input during editing
5. **Additional Inputs Dialog CAD Preview**:
   - Embedded CrossSectionCADWidget at 85% scale
   - Side-by-side layout with form fields
   - Live updates synchronized with main views
6. **SVG Icon System**: 4 custom icons for view toggles:
   - 22o̊0d722px crisp-edged SVG format
   - currentColor stroke for theme compatibility
   - Open (filled) vs. closed (line) visual states
7. **Log Window**: Enhanced visual presentation:
   - 1px light gray border with 3px rounded corners
   - Monospace font for aligned log output
   - 5px padding for content separation
8. **Cross Bracing Fix**: Corrected rendering issues:
   - Proper Z-order layering
   - Precise girder edge alignment
   - Sub-pixel coordinate accuracy with QPointF

5.10.2 Performance Optimizations

- **Debounced Updates**: editingFinished vs. textChanged prevents excessive redraws
- **Conditional Zoom Controls**: Preview mode detection avoids rendering unused buttons
- **Efficient Hover Detection**: Single linear scan through registered zones per mouse move
- **Fixed Widget Sizing**: setWidgetResizable(False) prevents continuous resize calculations

5.11 User Experience Impact

5.11.1 Workflow Improvements

Before Integration:

1. User enters parameters in OsdagBridge
2. Exports design to standalone CAD tool
3. Views visualization separately
4. Identifies issues, returns to OsdagBridge
5. Modifies parameters, repeats cycle

After Integration:

1. User enters parameters in OsdagBridge
2. CAD updates instantly in split-screen view
3. Zooms/pans to inspect details in each view independently
4. Hovers over components to see dimensions
5. Identifies issues immediately, adjusts parameters on-the-fly
6. Sees live preview in Additional Inputs dialog before committing changes

Time Savings: Estimated 60-70% reduction in design iteration time by eliminating export/import cycle.

5.11.2 Accessibility Features

- **Dual Views**: Simultaneous cross-section and top view reduce mental 3D reconstruction burden
- **Independent Zoom**: Users with vision impairments can magnify specific views
- **Hover Labels**: On-demand information reduces need to memorize parameter values
- **View Toggles**: Single-view mode maximizes screen space for detail work

5.12 Future Enhancement Opportunities

5.12.1 Potential Improvements for Future Development

1. **Zoom Control UI Improvements**: Redesign zoom button icons to match application UI theme and reposition controls for better accessibility and reduced screen clutter
2. **Component Selection**: Click-to-select components with automatic focus on corresponding input field for faster parameter editing
3. **Measurement Tools**: Click-and-drag distance measurement capability between any two points in CAD view
4. **Enhanced Zoom Functionality**: Implement smooth zoom transitions with improved scaling algorithms to maintain CAD image clarity at all zoom levels
5. **Multi-Touch Support**: Add two-finger pinch gestures for intuitive zoom-in and zoom-out operations on touchscreen-enabled devices
6. **Wearing Course Layer**: Add visual representation of wearing course (bituminous layer) above deck slab in cross-section view for complete pavement system visualization
7. **Toggle Icon Redesign**: Redesign cross-section and top view toggle icons with improved visual clarity and modern aesthetics
8. **Engineering Drawing Standards**: Update dimension labels and annotations to conform to standard engineering drawing conventions (arrow styles, text placement,

line types)

9. **Material Textures**: Enhance deck and footpath rendering with realistic concrete textures and surface patterns for improved visual distinction
10. **Color Scheme Refinement**: Update color palette for girders, stiffeners, cross-bracing, and crash barriers to improve contrast and readability
11. **Skew Angle Visualization**: Improve skew angle indicator with clearer arc representation and enhanced angular dimension display in top view

5.12.2 Scalability Considerations

- **Plugin Architecture**: Enable third-party view types (elevation, detail views)
- **Backend Separation**: Decouple rendering logic for potential web client
- **Level-of-Detail**: Automatically simplify rendering when zoomed out
- **Lazy Rendering**: Defer off-screen component rendering until scrolled into view

5.13 Deliverables

5.13.1 Code Components

5.13.2 Integration Points Modified

- **Main Window Layout**: Added dual CAD widget to central splitter
- **Additional Inputs Dialog**: Embedded preview and signal connections
- **Log Window Setup**: Applied border styling CSS
- **Resource System**: Compiled new SVG icons into resources
- **Cross Bracing Rendering**: Fixed alignment and Z-order in CAD renderer

5.14 Conclusion

Task 4 successfully integrated the standalone 2D CAD system into OsdagBridge, transforming it from a post-design visualization tool into a real-time design assistant. The dual split view architecture with independent zoom controls, interactive hover detection,

Component/File	Description
`BridgeDualCADWidget`	Dual split view container with vertical splitter, independent scroll areas, and dynamic view visibility control
`CrossSectionCADWidget`	Enhanced cross-section renderer with independent zoom controls, hover detection, and real-time parameter synchronization
`TopViewCADWidget`	Top view CAD renderer with skew angle support, zoom controls, and interactive hover highlighting
`AdditionalInputsDialog`	Modified dialog with embedded live CAD preview at 85% scale and real-time update connections
`resources.qrc`	Qt resource file containing 4 custom SVG icons for view toggle controls
`resources_rc.py`	Compiled Qt resource module (generated from resources.qrc using pyrcc6)
`cross_section_open.svg`	22x22px SVG icon showing expanded cross-section view state
`cross_section_closed.svg`	22x22px SVG icon showing collapsed cross-section view state
`top_view_open.svg`	22x22px SVG icon showing expanded top view state
`top_view_closed.svg`	22x22px SVG icon showing collapsed top view state
Log Window Enhancement	CSS border styling applied to output log window for improved visual separation and readability
Cross Bracing Fix	Corrected rendering with proper Z-order, edge alignment, and coordinate precision

Table 5.2: Task 4 Code Deliverables

and live parameter synchronization significantly improved the bridge design workflow. The implementation balances feature richness with performance, providing responsive updates even during rapid parameter changes. The addition of a scaled CAD preview in the Additional Inputs dialog further reduced design iteration time by enabling users to validate geometric feasibility before committing changes.

The modular architecture (separate cross-section and top view widgets, reusable hover detection system, pluggable icon resources) ensures maintainability and extensibility for

future enhancements. All code follows PySide6 best practices, including proper signal-slot connections, efficient event handling, and responsive UI patterns.

Chapter 6

Conclusions

6.1 Tasks Accomplished

This internship successfully delivered four major contributions to the FOSSEE ecosystem and OsdagBridge project, progressing from foundational infrastructure to polished user-facing features.

6.1.1 Task 1: IRC5-2015 Code Implementation

- Developed a comprehensive Python module encoding IRC 5:2015 standards for road bridge design.
- Implemented 5+ critical design check functions corresponding to specific IRC clauses.
- Created a reusable, extensible architecture for standards compliance checking.
- Documented all functions with IRC clause references for traceability.
- Established constants library for maintainable standards parameter management.

The IRC5-2015 module provides a foundation for automating bridge design verification, ensuring Indian bridge designs conform to national standards and best practices.

6.1.2 Task 2: Database Schema Design and SQL Implementation

- Designed a normalized, parent-child relational database schema for weather data.

- Implemented SQLite database with 4 tables, proper foreign keys, and cascading constraints.
- Created comprehensive Python utility class with 10+ database operation methods.
- Populated database with extensive Indian weather station data across all states.
- Implemented legacy database support and bootstrap mechanisms.
- Optimized with strategic indexing for efficient querying.

The weather database serves as a reusable resource for bridge load analysis, environmental studies, and climate-aware design decisions in the OsdagBridge ecosystem.

6.1.3 Task 3: 2D CAD Visualization System Development

- Engineered a complete, production-quality 2D CAD visualization system.
- Developed parametric bridge model with 15+ independently configurable parameters.
- Implemented dual-view rendering (cross-section and top-view) with comprehensive graphics.
- Created comprehensive dimensioning system with 8+ professional annotation styles.
- Integrated advanced interactive features: hover detection, zoom/pan/reset, component identification.
- Implemented IRC 5:2015 compliant structural element rendering (crash barriers, railings).
- Handled complex geometric transformations including skew angle compensation.
- Developed specialized components: median barriers, footpath configurations, cross-bracing patterns.

The 2D CAD system provides visual representation of bridge designs, enabling users to validate geometry, understand component interactions, and make informed design decisions.

6.1.4 Task 4: CAD Integration into OsdagBridge Desktop Application

- Successfully integrated the 2D CAD widget as primary visualization in OsdagBridge main window.
- Implemented real-time parameter synchronization between design inputs and CAD display.
- Created side-by-side Additional Inputs dialog with live CAD preview.
- Developed 4 new SVG icon resources for view control (cross-section and top-view toggles).
- Enhanced message logging with styled display and color-coded severity levels.
- Implemented component hover identification with detailed property display.
- Integrated CAD visualizations into generated design reports.
- Ensured responsive, non-blocking UI with seamless user experience.

The CAD integration brings the visualization system into production use, enabling practicing engineers to benefit from real-time visual feedback during design, significantly improving usability and reducing design iteration cycles.

6.2 Technical Skills Developed

6.2.1 Python Programming

- Advanced OOP design with static methods and class-based architecture.
- Type hints and proper documentation practices.
- Database programming with SQLite and SQL query optimization.
- File I/O and schema management.

6.2.2 Software Engineering

- Design pattern implementation (parent-child relationship, canonical table).
- API design and method signature optimization.

- Code documentation and standards compliance.
- Git workflow and version control (as evidenced by commit history).

6.2.3 Web/Desktop Application Development

- PySide6 (Qt) widget development and UI enhancement.
- SVG icon creation and resource management in Qt applications.
- Interactive graphics and CAD visualization implementation.
- Event handling and user interaction patterns.

6.2.4 Standards and Specifications

- Understanding of IRC 5:2015 bridge design standards.
- Meteorological data standards and measurements.
- CAD and geometric representation standards.
- Database normalization and relational theory.

6.3 Professional Growth

This internship provided valuable experience in:

- **Large-Scale Project Contribution**: Working on a complex, modular software ecosystem (OsdagBridge) with multiple interconnected components.
- **Open-Source Development**: Contributing to FOSSEE, a free and open-source software initiative, understanding community-driven development practices.
- **Standards Compliance**: Implementing and enforcing technical standards in software, a critical skill for engineering applications.
- **User-Centered Design**: Balancing backend implementation with user experience through improved UI/UX.
- **Documentation and Communication**: Creating clear, comprehensive documentation of technical work for both developers and users.

6.4 Impact and Future Work

The four completed tasks form an integrated, progressive contribution to the OsdagBridge ecosystem:

1. The IRC5-2015 module enables automated compliance checking, improving design reliability.
2. The weather database provides essential climate data for load calculations and environmental considerations.
3. The 2D CAD system offers visual representation of complex geometric designs, making bridge engineering more intuitive.
4. The CAD integration brings visualization capabilities to practicing engineers in a production environment.

6.4.1 Immediate Applications

- Designers can verify bridge designs against IRC standards automatically.
- Weather data can be integrated into load analysis calculations.
- Visual CAD previews enable design validation and component visualization.
- Integrated UI provides seamless design workflow from parameter input to visual output.

6.4.2 Long-Term Potential

- Expand IRC5-2015 module to include additional clauses and design checks.
- Integrate weather database with automated load combination analysis.
- Develop 3D CAD capabilities based on 2D visualization framework (330+ methods ready for extension).
- Create collaborative design features for multi-user projects.
- Export CAD visualizations to standard formats (PDF, SVG, DXF) for designer workflows.
- Implement constraint-based parametric design with real-time feasibility checking.

6.5 Acknowledgments

I would like to express my gratitude to:

- **Prof. Siddharth Ghosh** for guidance and mentorship throughout the internship.
- **FOSSEE Team** (Nidhi Khare, Aditya Donde, Mohammed Suhail) for their support and code reviews.
- **OsdagBridge Community** for providing a well-structured codebase to contribute to.
- **IIT Bombay and NMEICT** for supporting open-source software development in education.

6.6 Final Remarks

This internship was an enriching experience that combined theoretical knowledge of bridge engineering with practical software development skills. The ability to translate engineering standards into executable code, design databases for real-world data, and create intuitive user interfaces has prepared me well for future roles in structural engineering software development.

The work completed demonstrates that well-designed software can make complex engineering tasks more accessible, verifiable, and efficient. I am excited about the potential of tools like OsdagBridge to democratize structural engineering design, making it more accessible to engineers in all parts of India and beyond.

Chapter A

Daily Work Log

This appendix presents work completed during the FOSSEE internship from October 7, 2025 to January 1, 2026.

A.1 Daily Activity Log

Date	Task	Work Description	Hrs
2025-10-07	OSDAG Setup	Introduction to OSDAG framework, reviewed project documentation and scope	5
2025-10-08	OSDAG Setup	Installed OSDAG, configured Python 3.10 environment with PySide6	6
2025-10-09	OSDAG Setup	Explored GUI structure, studied widget hierarchy and signal-slot patterns	4
2025-10-10	OSDAG Setup	Studied connection design modules and IS code compliance mechanisms	5
2025-10-11	OSDAG Setup	Set up test cases for end plate, fin plate, and cleat angle connections	3
2025-10-12	OSDAG Setup	Reviewed design workflows and practiced sample connection models	4
2025-10-13	OSDAG Setup	Tested multiple configurations, validated against IS 800:2007	6
2025-10-14	OSDAG Setup	Documented environment setup and learning outcomes	5
2025-10-15	IRC Module	Started IRC 5:2015 module, created base class structure	4
2025-10-16	IRC Module	Implemented carriageway width, kerb, and footpath validation functions	6
2025-10-17	IRC Module	Developed crash barrier and railing height calculation methods	5
2025-10-18	IRC Module	Tested pathway logic with various bridge configurations	6
2025-10-19	IRC Module	Code review and edge case optimization	5
2025-10-20	IRC Module	Fixed geometry alignment and safety kerb area calculations	3
2025-10-21	IRC Module	Implemented output formatting with clause references	6
2025-10-22	IRC Module	Refactored code, added type hints for better documentation	5
2025-10-23	IRC Module	Validated outputs with manual calculations for 15+ test cases	4
2025-10-24	IRC Module	Finalized code with comprehensive docstrings	5
2025-10-25	IRC Module	Prepared presentation materials and demo scenarios	3
2025-10-26	IRC Module	Added test cases for median barriers and skew angles	4
2025-10-27	IRC Module	Extended logic for dual crash barrier configurations	5
2025-10-28	IRC Module	Optimized calculations using NumPy operations	4
2025-10-29	IRC Module	Regression testing with 30+ test scenarios	6
2025-10-30	IRC Module	Standardized error messages and compliance reports	3
2025-10-31	IRC Module	Final review, verified PEP 8 compliance	4

Date	Task	Work Description	Hrs
2025-11-01	IRC Presentation	Delivered module presentation with live demos	5
2025-11-02	Database	Initiated weather database planning, researched data sources	5
2025-11-03	Database	Designed normalized schema with parent-child relationships	6
2025-11-04	Database	Implemented SQL schema with 4 tables and foreign keys	3
2025-11-05	Database	Inserted dataset for 150+ stations across Indian states	4
2025-11-06	Database	Developed Python utility class with query methods	6
2025-11-07	Database Presentation	Presented schema with query performance demo	5
2025-11-08	CAD Research	Explored AutoCAD APIs and Qt QPainter rendering	4
2025-11-09	CAD Research	Studied Qt Graphics Framework and coordinate transforms	6
2025-11-10	CAD Research	Tested Python CAD automation with pyautocad	5
2025-11-11	CAD Design	Designed bridge cross-section renderer wireframe	5
2025-11-12	Database	Added stations for underrepresented states	4
2025-11-13	Database	Optimized indexes for state and station queries	6
2025-11-14	Database	Comprehensive testing and validation	5
2025-11-15	Database	SQL cleanup and standardized naming conventions	3
2025-11-16	CAD Setup	Set up development environment and gathered reference drawings	6
2025-11-17	CAD Development	Implemented cross bracing in top view	3
2025-11-18	CAD Development	Improved diagonal rendering and Z-order layering	4
2025-11-19	CAD Development	Fixed crash barrier-footpath gap, redesigned top view	5
2025-11-20	CAD Development	Implemented skew angle transformations	6
2025-11-21	CAD Development	Enhanced girder lines and cross-bracing patterns	5
2025-11-22	CAD Review	Code review and refactoring	4
2025-11-23	Database	Applied fixes and validation checks	6
2025-11-24	CAD & Database	Finalized SQL schema, started dimension labeling	5
2025-11-25	CAD Dimensioning	Implemented 8 dimension types with smart positioning	5
2025-11-26 to 11-29	OFF	Personal leave	0
2025-11-30	CAD Review	Post-break review and feedback analysis	6
2025-12-01	Task 4 Planning	Received Task 4 assignment, broke down requirements	4
2025-12-02	CAD Enhancement	Improved railing, footpath, and deck rendering	5
2025-12-03	CAD Enhancement	Implemented IRC compliant crash barrier profile	4
2025-12-04	Hover System	Developed hover detection with QRectF zones	6
2025-12-05	Hover System	Redesigned hover labels and fixed visualizations	5
2025-12-06	Task 4	Reviewed repository for integration points	5
2025-12-07	Integration	Researched Qt widget integration patterns	4
2025-12-08	Integration	Analyzed Garvit's repository architecture	5
2025-12-09	Integration	Studied parameter flow and signal connections	4
2025-12-10	Integration	Finalized approach: BridgeDualCADWidget with QSplitter	6
2025-12-11	Integration	Implemented dual split view with independent scrolling	6
2025-12-12	Integration	Fixed layout bugs and visibility toggles	5
2025-12-13	Integration	Tested with various parameters, fixed edge cases	3
2025-12-14	UI Polish	Refined splitter styling and focus management	6
2025-12-15	Dialog Integration	Integrated CAD preview into AdditionalInputsDialog	5
2025-12-16 to 12-18	OFF	Personal leave	0
2025-12-19	Research	Analyzed Aditya's repository patterns	4
2025-12-20	Presentation	Demoed dual split view and discussed log window	6
2025-12-21	Zoom Controls	Implemented independent zoom with +/- and Reset buttons	5
2025-12-22	Zoom UI	Styled zoom buttons with hover effects	4
2025-12-23	Dialog Preview	Completed live preview with auto-hide zoom controls	5
2025-12-24	Real-Time Sync	Implemented debounced parameter updates	6
2025-12-25	Hover Labels	Enhanced hover with dimensional information	5
2025-12-26	SVG Icons	Designed 4 custom 22×22px icons for view toggles	4

Date	Task	Work Description	Hrs
2025-12-27	Resources	Created resources.qrc and integrated icons	5
2025-12-28	Bug Fix	Fixed cross bracing Z-order and alignment	4
2025-12-29	Scaling	Improved preview scaling and scroll behavior	6
2025-12-30	Performance	Optimized redraw with caching and selective updates	5
2025-12-31	UX	Refined transitions and single-view focus mode	4
2026-01-01	Final	Completed testing, documentation, and report	6