METU ELECTRICAL AND ELECTRONICS ENGINEERING DEPARTMENT

EE 374

FUNDAMENTALS OF POWER SYSTEMS AND ELECTRICAL EQUIPMENT

Transmission Line Tower Design Project

Group 19

İbrahim Kaan UZUN 2444065

Ömer Oğuzhan BULUT 2442622

Introduction

In this project, our primary goal is to design a Python-based graphical user interface (GUI) application to calculate key parameters for transmission lines supported by different types of transmission towers. Transmission lines form the backbone of electrical power systems, connecting generation stations to distribution networks and delivering electricity to end users. Given the significance of efficient and reliable transmission, accurately determining the line parameters—such as resistance, inductance, capacitance, and power capacity—is crucial for ensuring optimal performance and safety.

This project specifically focuses on developing a tool that can handle various tower types, conductor configurations, and design constraints. By inputting relevant geometric and electrical specifications, users can obtain precise calculations for:

- Line resistance (R)
- Line inductance (L)
- Line charging capacitance (C)
- Line capacity (MVA)

To achieve these calculations, the project involves several steps, including understanding tower specifications, configuring conductor arrangements, and applying mathematical models to derive the desired parameters. The GUI aims to be user-friendly, allowing for easy input of variables and a clear presentation of results. This tool will be invaluable for engineers and designers involved in the planning and maintenance of transmission networks.

Methodology and Assumptions

The methodology for this project involves several key steps and calculations to ensure the accurate determination of transmission line parameters using a Python-based graphical user interface (GUI). Below are the detailed steps and the corresponding formulas used in the calculations:

1. **Given Parameters:**
    - There are three different tower types: Type-1: Narrow Base Tower, Type-2: Single Circuit Delta Tower, Type-3: Double Circuit Vertical Tower, and five different conductors: Hawk, Drake, Cardinal, Rail, and Pheasant. Each tower has height and horizontal distance restrictions for phases, Circuit number restrictions, different voltage levels, and restrictions on the maximum number of conductors in the bundle. Each conductor has Diameter, Conductor GMR, AC resistance, and Current Capacity parameters which can be seen in Table 1.

Table 1: Conductor Parameters

Conductor Name	Diameter (mm)	Conductor GMR (mm)	AC resistance (Ω/km)	Current Capacity (A)
Hawk	21.793	8.809	0.132	659
Drake	28.143	11.369	0.080	907
Cardinal	30.378	12.253	0.067	996
Rail	29.591	11.765	0.068	993
Pheasant	35.103	14.204	0.051	1187

2. User Interface Design:

- The GUI, seen in Figure 1 below, was developed using PySide6 to allow users to input various parameters related to transmission towers and conductors. The interface includes dropdowns for selecting tower types and conductor types, text inputs for coordinates and line lengths, and buttons for executing calculations and clearing inputs. Also, depending on the tower type selected in the GUI, a photo of that tower appears. This helps users understand which tower type it is.

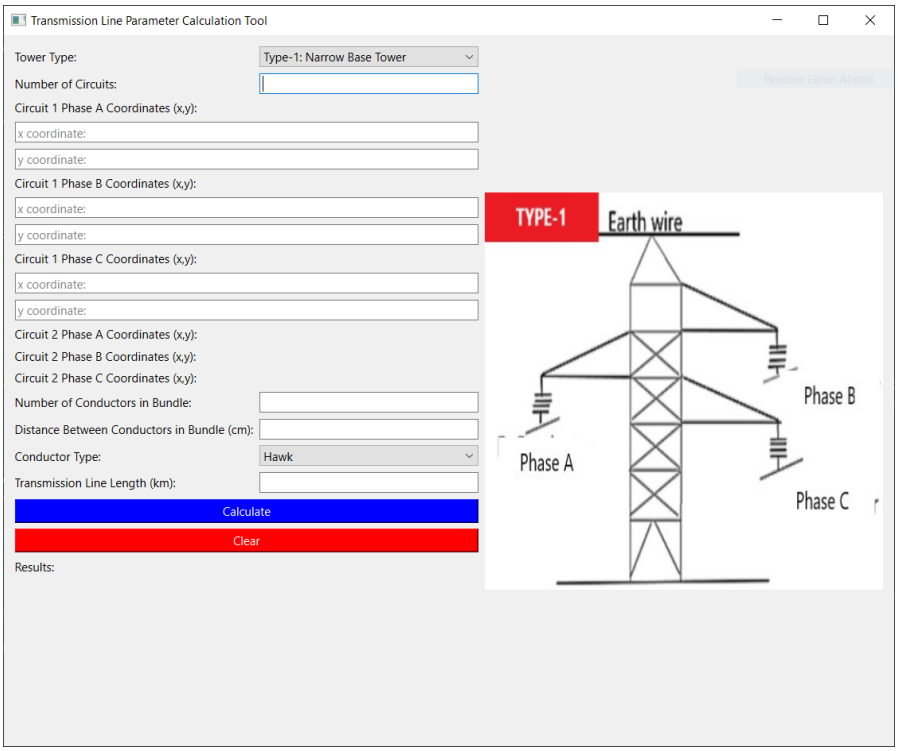

Figure 1: GUI of the program

3. Input Handling:

- The tool accepts multiple inputs, such as:
  i. Tower type,
  ii. Number of circuits,
  iii. Phase coordinates (x, y),
  iv. Number of conductors in the bundle,
  v. Distance between conductors,
  vi. Conductor type,
  vii. Transmission line length.
- Inputs are validated against predefined constraints for each tower type to ensure they fall within allowable ranges. This includes verifying the number of conductors, circuit configurations, and ensuring phase coordinates are within permissible limits.

4. Error Handling:

- The program may give an error message when errors are made while entering inputs according to the given parameters and restrictions. These are:
  i. The number of circuits cannot be different than 1 for Type-1 and Type-2 tower types and between 1 and 2 for Type-3. When a different input is entered, the program gives an error message. It can be seen in Figure 2.

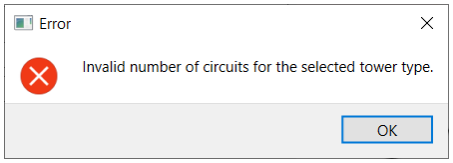

Figure 2: Error Message

  ii. Depending on the restrictions of the selected tower type, when the coordinates are entered incorrectly, the program gives an error message as to which coordinate was entered incorrectly. It can be seen in Figure 3.

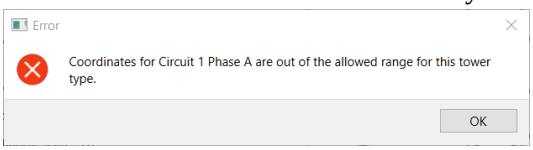

Figure 3: Error Message

  iii. Depending on the selected tower type, the number of conductors in the bundle is checked according to restrictions, and an error message is given. It can be seen in Figure 4.

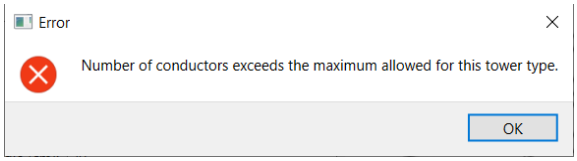

Figure 4: Error Message

  iv. If one input is missing, the program gives the error message seen in Figure 5.

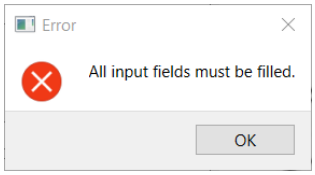

Figure 5: Error Message

5. Calculation of Distance Between Phases:

- An auxiliary function is written that calculates the distance between the entered A, B, and C phases according to their (x, y) coordinates as Dab, Dac, and Dbc according to the formula below, seen in Figure 6. Thanks to this function, GMD calculation was made.

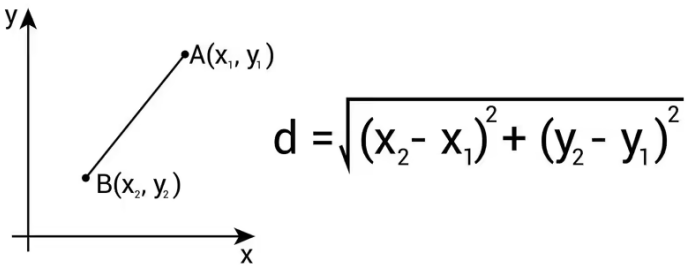

Figure 6: Formula of Distance Between Two Coordinates

6. **Geometric Mean Distance (GMD) Calculation:**
    - When the number of circuit is one, The GMD is calculated for the arrangement of phase conductors. This involves computing distances between phases and applying the geometric mean formula:

$$GMD = (D_{AB} \cdot D_{BC} \cdot D_{AC})^{\frac{1}{3}}$$

Where D_{AB}, D_{BC}, and D_{AC} are the distances between phase conductors A, B, and C, respectively.

    - When the number of circuits is two for tower type-3, typical arrangement of the conductors can be seen in Figure 7 [1].

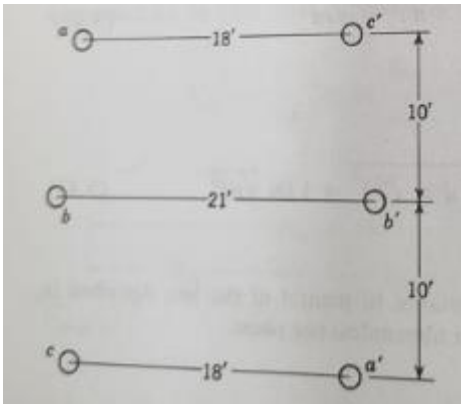

Figure 7: Typical Arrangement of the Conductors

    - GMD between Phase a and Phase b:

$$\text{GMD}_{ab} = \sqrt[4]{D_{ab} \cdot D_{ab'} \cdot D_{a'b} \cdot D_{a'b'}}$$

- D_{ab}: Distance between phase a and b.
- $D_{ab'}$: Distance between phase a and b'.
- $D_{a'b}$: Distance between phase a' and b.
- $D_{a'b'}$: Distance between phase a' and b'.

Overall GMD for Double Circuit: $\text{GMD}_{\text{double-circuit}} = \sqrt[3]{\text{GMD}_{ab} \cdot \text{GMD}_{ac} \cdot \text{GMD}_{bc}}$

- GMD_{ab}: GMD between phases a and b.
- GMD_{ac}: GMD between phases a and c.
- GMD_{bc}: GMD between phases b and c.

This provides a concise calculation method for the GMD in a double-circuit transmission line system.

7. **Geometric Mean Radius (GMR) Calculation**:
    - When the number of circuits is one, the GMR for the bundled conductors is calculated based on the number of conductors in the bundle and the distance between them:

$$\text{GMR} = \left(\text{GMR}_{\text{conductor}} \cdot d^{(n-1)}\right)^{\frac{1}{n}}$$

where $\text{GMR}_{\text{conductor}}$ is the given GMR of a single conductor, d is the distance between conductors in the bundle, and n is the number of conductors.

    - When the number of circuits is two for tower type-3, the GMR is calculated separately for each phase pair (a-a', b-b', c-c'):

$$\text{GMR}_{aa'} = \sqrt{\text{GMR}_{\text{bundle}} \cdot d_{aa'}}$$
$$\text{GMR}_{bb'} = \sqrt{\text{GMR}_{\text{bundle}} \cdot d_{bb'}}$$
$$\text{GMR}_{cc'} = \sqrt{\text{GMR}_{\text{bundle}} \cdot d_{cc'}}$$

- $\text{GMR}_{aa'}$: Distance between phase a and a'.
- $\text{GMR}_{bb'}$: Distance between phase b and b'.
- $\text{GMR}_{cc'}$: Distance between phase c and c'.

The overall GMR for the double circuit is then calculated as the geometric mean of the GMR values for each phase pair:

$$\text{GMR}_{\text{double-circuit}} = (\text{GMR}_{aa'} \cdot \text{GMR}_{bb'} \cdot \text{GMR}_{cc'})^{\frac{1}{3}}$$

8. **Equivalent Resistance Calculation:**

    - The Req for the bundled conductors is calculated based on the number of conductors in the bundle and the distance between them:

$$\text{Req} = \left(r_{eq} \cdot d^{(n-1)}\right)^{\frac{1}{n}}$$

where r_{eq} is the equivalent radius of a single conductor, d is the distance between conductors in the bundle, and n is the number of conductors.

    - When the number of circuits is two, the equivalent radius for each phase pair (a-a', b-b', c-c') is calculated as follows:

$$\text{R}_{eq_{aa'}} = \sqrt{R_{eq} \cdot d_{aa'}}$$
$$\text{R}_{eq_{bb'}} = \sqrt{R_{eq} \cdot d_{bb'}}$$
$$\text{R}_{eq_{cc'}} = \sqrt{R_{eq} \cdot d_{cc'}}$$

The overall R_{eq} for the double circuit is: $R_{eq_{\text{double-circuit}}} = \left(R_{eq_{aa'}} \cdot R_{eq_{bb'}} \cdot R_{eq_{cc'}}\right)^{\frac{1}{3}}$

- $d_{aa'}$: Distance between phase a and a'.
- $d_{bb'}$: Distance between phase b and b'.
- $d_{cc'}$: Distance between phase c and c'.

9. **Line Resistance Calculation**:
   - The line resistance is calculated using the AC resistance of the conductor and the length of the transmission line:

$$R = R_{AC} \cdot \text{length}$$

where R_{AC} is the AC resistance per kilometer of the conductor which is given in Table1, and length is the length of the transmission line in kilometers.

10. **Inductance Calculation**:
    - The inductance per unit length is calculated using the GMD and GMR:

$$L = \frac{\mu_0}{2\pi} \ln\left(\frac{\text{GMD}}{\text{GMR}}\right) \text{ H/m}$$

The total inductance for the transmission line is then:

$$L_{\text{total}} = L \cdot \text{length} \times 10^6$$

where L_{total} is in milli Henries.

11. **Capacitance Calculation**:
    - The capacitance per unit length is calculated using the GMD and the equivalent radius (R_{eq}):

$$C = \frac{2\pi\epsilon_0\epsilon_r}{\ln\left(\frac{\text{GMD}}{\text{Req}}\right)}$$

    - The total capacitance for the transmission line is then:

$$C_{\text{total}} = C \cdot \text{length} \times 10^9$$

where C_{total} is in microfarads.

12. **Line Capacity Calculation**:
    - The capacity of the transmission line is calculated based on the voltage and current capacity of the conductor:

$$\text{Line Capacity} = V \cdot I \cdot \sqrt{3} \cdot 10^{-3}$$

where V is the voltage level of the tower type in volts, and I is the current capacity of the conductor in amperes. Capacity in Mega Volt Amperes.

Assumptions

To simplify the calculations and ensure the tool's usability, several assumptions were made during the development process:

1. Ideal Conductor Placement:
    - It is assumed that the conductors are placed precisely according to the input coordinates without any physical deviations or sagging.
2. Uniform Conductor Types:
    - The tool assumes uniform conductor types for each phase, as specified by the user. Different types of conductors within the same phase or bundle are not considered.
3. Perfect Transposition:
    - The tool assumes perfect transposition of the transmission lines, ensuring no unbalance in the line parameters due to phase arrangement or geometric deviations.
4. Negligible Earth Effects:
    - For the sake of simplicity, the effect of the earth on the capacitance calculations is considered negligible. This assumption allows for more straightforward mathematical models without complex ground interactions.
5. Constant Environmental Conditions:
    - The calculations assume constant environmental conditions, such as temperature and humidity, which can affect conductor resistance and capacitance in real-world scenarios.
6. Specific Voltage Levels:
    - The tool uses predefined voltage levels for each tower type as provided in the project specifications. Any variations in voltage levels are not considered in the calculations.
7. Fixed Conductor Configuration:
    - The conductor configuration within the bundle is assumed to be a regular polygon, simplifying the calculation of distances between conductors.

By following this methodology and sticking to these assumptions, the Transmission Line Parameter Calculation Tool effectively provides accurate and reliable results for the design and analysis of transmission lines. This approach ensures that the tool is both practical for educational purposes and useful for real-world engineering applications.

Test Results

Some example calculations for different tower types, different conductor types, different conductor numbers and different number of circuits can be seen in Figures 8,9,10,11 below.

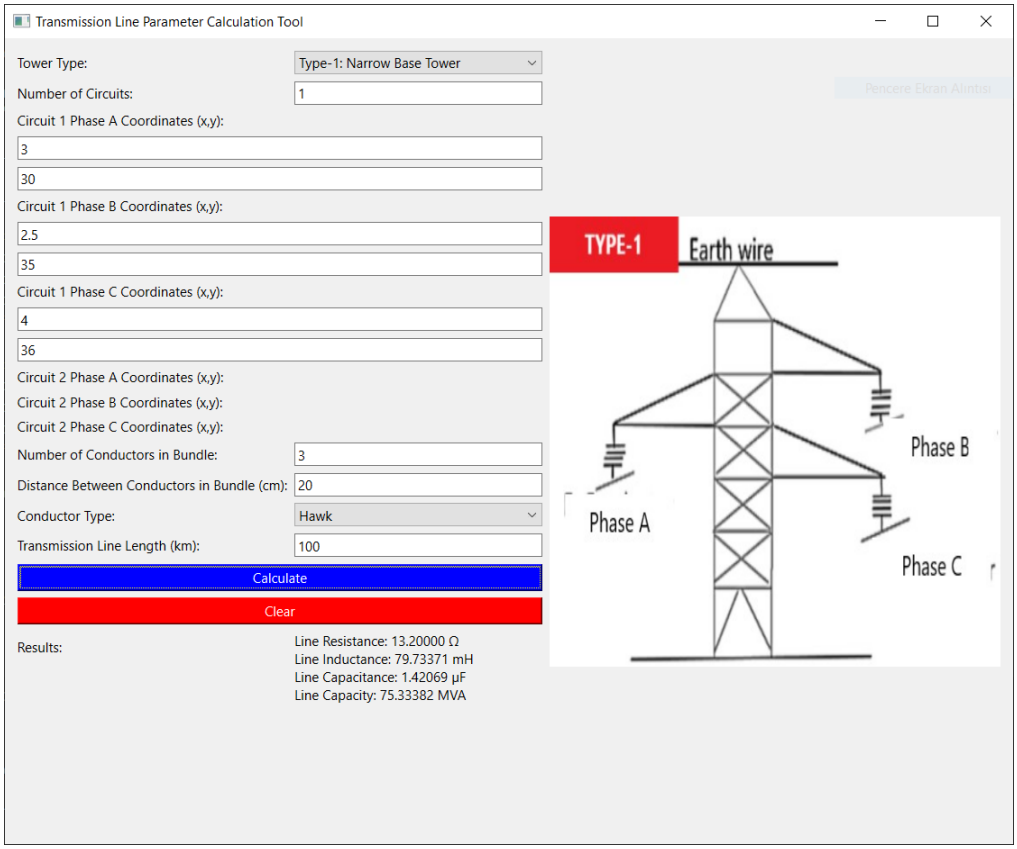

Figure 8: Example Test Result

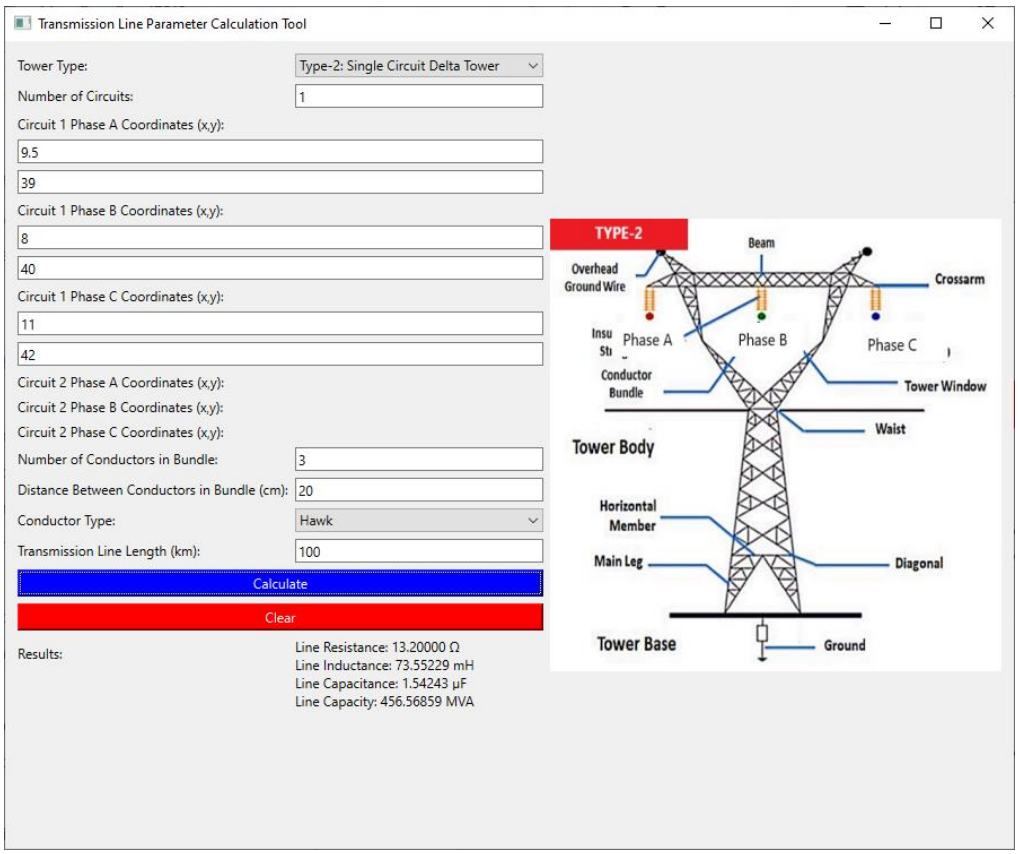

Figure 9: Example Test Result

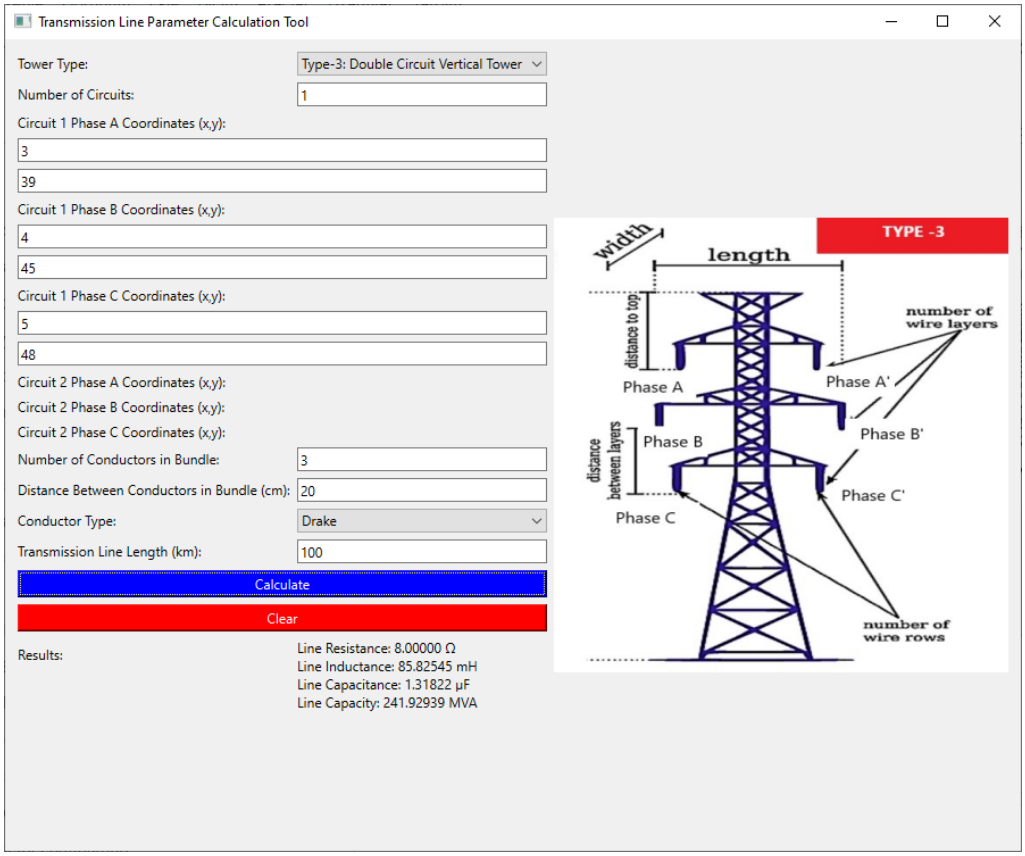

Figure 10: Example Test Result

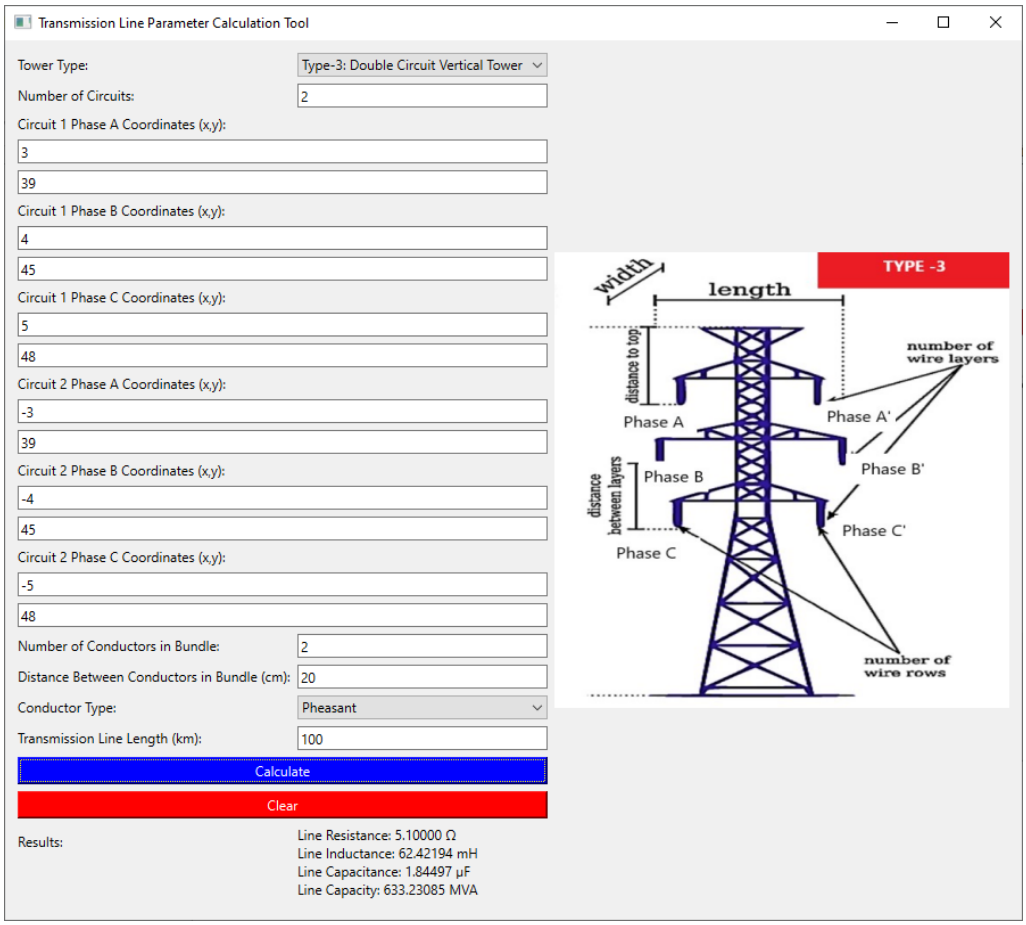

Figure 11: Example Test Result

Conclusion

The Transmission Line Parameter Calculation Tool developed in this project provides a comprehensive solution for accurately determining essential parameters of transmission lines based on various tower types and conductor configurations. By leveraging Python and the PySide6 framework, we created an intuitive and efficient application that simplifies the complex and time consuming process of transmission line design.

Our tool ensures that users can input diverse specifications and receive accurate results, enhancing the reliability and performance of transmission networks. The successful implementation of this tool demonstrates the practical application of theoretical concepts in electrical engineering and highlights the importance of accurate parameter calculation in the design and maintenance of power systems.

Through this project, we have gained valuable insights into the intricacies of transmission line design, including the impact of geometric configurations and electrical properties on line performance. This experience not only strengthens our understanding of power systems but also equips us with practical skills in software development and GUI design. The Transmission Line Parameter Calculation Tool stands as a testament to the integration of theoretical knowledge and practical application, providing a useful resource for professionals in the field. In the appendix section there are

References

[1] J. J. Grainger and W. D. Stevenson, Power System Analysis, McGraw-Hill, 1994 (Reserve, TK3001 .G73 1994) or J. J. Grainger, W. D. Stevenson, G.W. Chang, Power System Analysis, McGraw-Hill, 2016.

[2] Lecture Notes.

Appendix

############################## OUR PROJECT CODE ##############################

```
import sys
import os
from PySide6.QtGui import QPixmap
from PySide6.QtWidgets import QApplication, QWidget, QPushButton, QVBoxLayout,
QHBoxLayout, QComboBox, QLabel, QLineEdit, QFormLayout, QMessageBox
import math

# Conductor and tower type details with additional constraints
conductors = {
    'Hawk': {'Diameter': 21.793, 'GMR': 8.809, 'AC_resistance': 0.132, 'Current Capacity': 659},
    'Drake': {'Diameter': 28.143, 'GMR': 11.369, 'AC_resistance': 0.080, 'Current Capacity': 907},
    'Cardinal': {'Diameter': 30.378, 'GMR': 12.253, 'AC_resistance': 0.067, 'Current Capacity': 996},
    'Rail': {'Diameter': 29.591, 'GMR': 11.765, 'AC_resistance': 0.068, 'Current Capacity': 993},
    'Pheasant': {'Diameter': 35.103, 'GMR': 14.204, 'AC_resistance': 0.051, 'Current Capacity': 1187}
}

tower_types = {
    'Type-1: Narrow Base Tower': {'max_height': 39, 'min_height': 23, 'max_horizontal': 4,
'min_horizontal': 2.2, 'voltage': 66, 'max_conductors': 3},
    'Type-2: Single Circuit Delta Tower': {'max_height': 43, 'min_height': 38.25, 'max_horizontal':
11.5, 'min_horizontal': 9.4, 'max_horizontal_center': 8.9, 'voltage': 400, 'max_conductors': 4},
    'Type-3: Double Circuit Vertical Tower': {'max_height': 48.8, 'min_height': 36, 'max_horizontal':
5.35, 'min_horizontal': 1.8, 'voltage': 154, 'max_conductors': 3}
}

def check_constraints(tower_type, num_conductors, coordinates, num_circuits):
    if num_conductors > tower_types[tower_type]['max_conductors']:
        return "Number of conductors exceeds the maximum allowed for this tower type."

    if num_conductors <= 0:
        return "Number of conductors cannot be less than 1."

    if (tower_type == 'Type-3: Double Circuit Vertical Tower' and num_circuits not in [1, 2]) or
(tower_type in ['Type-1: Narrow Base Tower', 'Type-2: Single Circuit Delta Tower'] and num_circuits
!= 1):
        return "Invalid number of circuits for the selected tower type."

    tower_specs = tower_types[tower_type]
    for circuit in coordinates:
        for phase, (x, y) in coordinates[circuit].items():
            if tower_type == 'Type-1: Narrow Base Tower':
```

```
            if not ((tower_specs['min_horizontal'] <= abs(x) <= tower_specs['max_horizontal']) and (x
>= tower_specs['min_horizontal'] or x <= -tower_specs['min_horizontal']) and
tower_specs['min_height'] <= y <= tower_specs['max_height']):
                return f"Coordinates for Circuit {circuit} Phase {phase} are out of the allowed range for
this tower type."
        elif tower_type == 'Type-2: Single Circuit Delta Tower':
            if phase == 'B':
                if not (-8.9 <= x <= 8.9 and tower_specs['min_height'] <= y <=
tower_specs['max_height']):
                    return f"Coordinates for Circuit {circuit} Phase {phase} are out of the allowed range
for this tower type."
            else:
                if not ((9.4 <= abs(x) <= 11.5) and tower_specs['min_height'] <= y <=
tower_specs['max_height']):
                    return f"Coordinates for Circuit {circuit} Phase {phase} are out of the allowed range
for this tower type."
        elif tower_type == 'Type-3: Double Circuit Vertical Tower':
            if not ((tower_specs['min_horizontal'] <= abs(x) <= tower_specs['max_horizontal']) and (x
>= tower_specs['min_horizontal'] or x <= -tower_specs['min_horizontal']) and
tower_specs['min_height'] <= y <= tower_specs['max_height']):
                return f"Coordinates for Circuit {circuit} Phase {phase} are out of the allowed range for
this tower type."
    return None  # No errors

class TransmissionLineCalc(QWidget):
    def __init__(self):
        super().__init__()
        self.init_ui()

    def init_ui(self):
        self.setWindowTitle('Transmission Line Parameter Calculation Tool')

        main_layout = QHBoxLayout()  # Main layout to hold form and image
        form_layout = QFormLayout()  # Form layout for inputs and buttons

        self.tower_type_combo = QComboBox()
        self.tower_type_combo.addItems(list(tower_types.keys()))
        self.tower_type_combo.currentIndexChanged.connect(self.update_phase_inputs)
        self.tower_type_combo.currentIndexChanged.connect(self.update_image)
        form_layout.addRow('Tower Type:', self.tower_type_combo)

        self.num_circuits_input = QLineEdit()
        self.num_circuits_input.textChanged.connect(self.update_phase_inputs)
        form_layout.addRow('Number of Circuits:', self.num_circuits_input)

        self.phase_inputs = {}
        for circuit in range(1, 3):  # Support up to 2 circuits
            for phase in ['A', 'B', 'C']:
```

```
                coord_x = QLineEdit()
                coord_x.setPlaceholderText('x coordinate:')
                coord_y = QLineEdit()
                coord_y.setPlaceholderText('y coordinate:')
                self.phase_inputs[(circuit, phase)] = (coord_x, coord_y)
                phase_layout = QVBoxLayout()
                phase_layout.addWidget(QLabel(f'Circuit {circuit} Phase {phase} Coordinates (x,y):'))
                phase_layout.addWidget(coord_x)
                phase_layout.addWidget(coord_y)
                form_layout.addRow(phase_layout)

        self.num_conductors_input = QLineEdit()
        form_layout.addRow('Number of Conductors in Bundle:', self.num_conductors_input)

        self.distance_between_conductors_input = QLineEdit()
        form_layout.addRow('Distance Between Conductors in Bundle (cm):',
self.distance_between_conductors_input)

        self.conductor_type_combo = QComboBox()
        self.conductor_type_combo.addItems(list(conductors.keys()))
        form_layout.addRow('Conductor Type:', self.conductor_type_combo)

        self.line_length_input = QLineEdit()
        form_layout.addRow('Transmission Line Length (km):', self.line_length_input)

        self.calculate_button = QPushButton('Calculate')
        self.calculate_button.setStyleSheet("background-color: blue; color: white;")
        self.calculate_button.clicked.connect(self.perform_calculation)
        form_layout.addRow(self.calculate_button)

        self.clear_button = QPushButton('Clear')
        self.clear_button.setStyleSheet("background-color: red; color: white;")
        self.clear_button.clicked.connect(self.clear_inputs)
        form_layout.addRow(self.clear_button)

        self.results_label = QLabel('')
        form_layout.addRow('Results:', self.results_label)

        main_layout.addLayout(form_layout)  # Add form layout to main layout

        # Add image to the right side
        self.image_label = QLabel()
        main_layout.addWidget(self.image_label)  # Add image label to main layout

        self.setLayout(main_layout)  # Set the main layout

        # Initial image
        self.update_image()
```

```
    def update_image(self):
        tower_type = self.tower_type_combo.currentText()
        current_directory = os.path.dirname(os.path.abspath(__file__))

        if tower_type == 'Type-1: Narrow Base Tower':
            image_path = os.path.join(current_directory, 'type1_image.jpg')
        elif tower_type == 'Type-2: Single Circuit Delta Tower':
            image_path = os.path.join(current_directory, 'type2_image.jpg')
        elif tower_type == 'Type-3: Double Circuit Vertical Tower':
            image_path = os.path.join(current_directory, 'type3_image.jpg')
        else:
            image_path = ''

        if image_path and os.path.exists(image_path):
            pixmap = QPixmap(image_path)
            self.image_label.setPixmap(pixmap)
            self.image_label.setScaledContents(True)
            self.image_label.setFixedSize(400, 400)  # Adjust size as needed
        else:
            self.image_label.setText('Image not found.')

    def update_phase_inputs(self):
        tower_type = self.tower_type_combo.currentText()
        num_circuits = int(self.num_circuits_input.text()) if self.num_circuits_input.text().isdigit() else 1

        for circuit in range(1, 3):
            for phase in ['A', 'B', 'C']:
                visible = (tower_type == 'Type-3: Double Circuit Vertical Tower' and num_circuits == 2) or
circuit == 1
                self.phase_inputs[(circuit, phase)][0].setVisible(visible)
                self.phase_inputs[(circuit, phase)][1].setVisible(visible)

    def distance(self, x1, x2, y1, y2):
        return math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

    def GMD_calculator(self, coordinates):
        ax, ay = coordinates[1]['A']
        bx, by = coordinates[1]['B']
        cx, cy = coordinates[1]['C']
        a2b = self.distance(ax, bx, ay, by)
        a2c = self.distance(ax, cx, ay, cy)
        b2c = self.distance(bx, cx, by, cy)
        return (a2b * a2c * b2c) ** (1 / 3)

    def GMR_calculator(self, GMR_conductor, num_conductors, distance_between_conductors):
        if num_conductors == 1:
            return GMR_conductor
```

```
        elif num_conductors == 2 :
            return ((GMR_conductor * distance_between_conductors) ** (1 / 2))
        elif num_conductors == 3 :
            return (((GMR_conductor * distance_between_conductors ** 2)) ** (1 / 3))
        elif num_conductors == 4:
            return ((GMR_conductor * (distance_between_conductors ** 2) *
(distance_between_conductors * math.sqrt(2))) ** (1 / 4))

    def Req_calculator(self, r_eq, num_conductors, distance_between_conductors):
        if num_conductors == 1:
            return r_eq
        elif num_conductors == 2 :
            return ((r_eq * distance_between_conductors) ** (1 / 2))
        elif num_conductors == 3 :
            return ((r_eq * (distance_between_conductors ** 2)) ** (1 / 3))
        elif num_conductors == 4:
            return ((r_eq * (distance_between_conductors ** 2) * (distance_between_conductors *
math.sqrt(2))) ** (1 / 4))

    def two_GMR(self, GMR, a1x, b1x, c1x, a1y, b1y, c1y, a2x, b2x, c2x, a2y, b2y, c2y):
        GMR1 = (self.distance(a1x, a2x, a1y, a2y) * GMR) ** (1/2)
        GMR2 = (self.distance(b1x, b2x, b1y, b2y) * GMR) ** (1/2)
        GMR3 = (self.distance(c1x, c2x, c1y, c2y) * GMR) ** (1/2)
        return (GMR1 * GMR2 * GMR3) ** (1/3)

    def two_Req(self, Req, a1x, b1x, c1x, a1y, b1y, c1y, a2x, b2x, c2x, a2y, b2y, c2y):
        Req1 = (self.distance(a1x, a2x, a1y, a2y) * Req) ** (1/2)
        Req2 = (self.distance(b1x, b2x, b1y, b2y) * Req) ** (1/2)
        Req3 = (self.distance(c1x, c2x, c1y, c2y) * Req) ** (1/2)
        return (Req1 * Req2 * Req3) ** (1/3)

    def two_GMD(self, a1x, b1x, c1x, a1y, b1y, c1y, a2x, b2x, c2x, a2y, b2y, c2y):
        a1b1 = self.distance(a1x, b1x, a1y, b1y)
        a1b2 = self.distance(a1x, b2x, a1y, b2y)
        a2b1 = self.distance(a2x, b1x, a2y, b1y)
        a2b2 = self.distance(a2x, b2x, a2y, b2y)
        a_b = (a1b1 * a1b2 * a2b1 * a2b2) ** (1/4)

        a1c1 = self.distance(a1x, c1x, a1y, c1y)
        a1c2 = self.distance(a1x, c2x, a1y, c2y)
        a2c1 = self.distance(a2x, c1x, a2y, c1y)
        a2c2 = self.distance(a2x, c2x, a2y, c2y)
        a_c = (a1c1 * a1c2 * a2c1 * a2c2) ** (1/4)

        b1c1 = self.distance(b1x, c1x, b1y, c1y)
        b1c2 = self.distance(b1x, c2x, b1y, c2y)
        b2c1 = self.distance(b2x, c1x, b2y, c1y)
        b2c2 = self.distance(b2x, c2x, b2y, c2y)
```

```
        b_c = (b1c1 * b1c2 * b2c1 * b2c2) ** (1/4)
        return (a_b * a_c * b_c) ** (1/3)

    def perform_calculation(self):
        # Check if all necessary inputs are provided
        if not self.num_circuits_input.text() or not self.num_conductors_input.text() or not
self.distance_between_conductors_input.text() or not self.line_length_input.text():
            QMessageBox.critical(self, "Error", "All input fields must be filled.")
            return

        tower_type = self.tower_type_combo.currentText()
        num_conductors = int(self.num_conductors_input.text())
        num_circuits = int(self.num_circuits_input.text())

        coordinates = {}
        for circuit in range(1, num_circuits + 1):
            coordinates[circuit] = {}
            for phase in ['A', 'B', 'C']:
                x = float(self.phase_inputs[(circuit, phase)][0].text())
                y = float(self.phase_inputs[(circuit, phase)][1].text())
                coordinates[circuit][phase] = (x, y)

        all_coords = [coordinates[circuit][phase] for circuit in coordinates for phase in
coordinates[circuit]]
        if len(all_coords) != len(set(all_coords)):
            QMessageBox.critical(self, "Error", "Two or more phases have the same coordinates.")
            return

        error_message = check_constraints(tower_type, num_conductors, coordinates, num_circuits)
        if error_message:
            QMessageBox.critical(self, "Error", error_message)
            return

        distance_between_conductors = float(self.distance_between_conductors_input.text()) / 100 #cm
conversion
        conductor_type = self.conductor_type_combo.currentText()
        line_length = float(self.line_length_input.text())

        conductor_info = conductors[conductor_type]
        R = conductor_info['AC_resistance'] * line_length

        GMR_conductor = conductor_info['GMR'] / 1000  # Convert GMR to meters

        if tower_type == 'Type-3: Double Circuit Vertical Tower' and num_circuits == 2:
            a1x, a1y = coordinates[1]['A']
            b1x, b1y = coordinates[1]['B']
            c1x, c1y = coordinates[1]['C']
            a2x, a2y = coordinates[2]['A']
```

```
        b2x, b2y = coordinates[2]['B']
        c2x, c2y = coordinates[2]['C']

        GMR = self.two_GMR(GMR_conductor, a1x, b1x, c1x, a1y, b1y, c1y, a2x, b2x, c2x, a2y,
b2y, c2y)
        Req = self.two_Req(conductor_info['Diameter'] / 2000, a1x, b1x, c1x, a1y, b1y, c1y, a2x, b2x,
c2x, a2y, b2y, c2y)
        GMD = self.two_GMD(a1x, b1x, c1x, a1y, b1y, c1y, a2x, b2x, c2x, a2y, b2y, c2y)

        L = (2 * 10 ** -7 * math.log(GMD / GMR)) * line_length * 10**6
        epsilon_0 = 8.854 * 10 ** -12
        epsilon_r = 1
        C = (2 * math.pi * epsilon_0 * epsilon_r / math.log(GMD / Req)) * line_length * 10**9
        voltage = tower_types[tower_type]['voltage']
        current_capacity = conductors[conductor_type]['Current Capacity']
        MVA = 2 * voltage * current_capacity * math.sqrt(3) * 10 ** -3

        results = [
            f'Line Resistance: {R:.5f} Ω',
            f'Line Inductance: {L:.5f} mH',
            f'Line Capacitance: {C:.5f} μF',
            f'Line Capacity: {MVA:.5f} MVA',
          #  f'GMD: {GMD:.5f}',
          #  f'GMR: {GMR:.5f}',
          #  f'Req: {Req:.5f}'
        ]

    else:
        GMR = self.GMR_calculator(GMR_conductor, num_conductors,
distance_between_conductors)
        Req = self.Req_calculator(conductor_info['Diameter'] / 2000, num_conductors,
distance_between_conductors)
        GMD = self.GMD_calculator(coordinates)

        L = (2 * 10 ** -7 * math.log(GMD / GMR)) * line_length * 10**6
        epsilon_0 = 8.854 * 10 ** -12
        epsilon_r = 1
        C = (2 * math.pi * epsilon_0 * epsilon_r / math.log(GMD / Req)) * line_length * 10**9
        voltage = tower_types[tower_type]['voltage']
        current_capacity = conductors[conductor_type]['Current Capacity']
        MVA = voltage * current_capacity * math.sqrt(3) * 10 ** -3

        results = [
            f'Line Resistance: {R:.5f} Ω',
            f'Line Inductance: {L:.5f} mH',
            f'Line Capacitance: {C:.5f} μF',
            f'Line Capacity: {MVA:.5f} MVA',
          #  f'GMD: {GMD:.5f}',
```

```
            #  f'GMR: {GMR:.5f}',
            #  f'Req: {Req:.5f}'
           ]
        self.results_label.setText("\n".join(results))

    def clear_inputs(self):
        self.num_circuits_input.clear()
        for circuit in range(1, 3):
            for phase in ['A', 'B', 'C']:
                self.phase_inputs[(circuit, phase)][0].clear()
                self.phase_inputs[(circuit, phase)][1].clear()
        self.num_conductors_input.clear()
        self.distance_between_conductors_input.clear()
        self.line_length_input.clear()
        self.results_label.clear()

app = QApplication(sys.argv)
window = TransmissionLineCalc()
window.show()
sys.exit(app.exec())

################################ OUR PROJECT CODE ################################
```