

Automation And Solution Of 2d Truss Element Problem Using Python And Machine Learning

A PROJECT REPORT

Submitted by

| Sr. | Name of student | Enrolment No. |
|------------|----------------------------|----------------------|
| 1 | Makwana Rahul Premjibhai | 190200119037 |
| 2 | Pandya Manan Bharatbhai | 190200119046 |
| 3 | Pitroda Kishan Hasmukhbhai | 190200119055 |

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

In

Mechanical Engineering Department

Government Engineering College – Rajkot



Gujarat Technological University, Ahmedabad
[May, 2023]



**Government Engineering College
Rajkot**

CERTIFICATE

This is to certify that the project report submitted along with the project entitled **Automation And Solution Of 2d Truss Element Problem Using Python And Machine Learning** has been carried out by **Makwana Rahul Premjibhai** under my guidance in partial fulfillment for the degree of Bachelor of Engineering in Mechanical Engineering, 8th Semester of Gujarat Technological University, Ahmadabad during the academic year 2022-23.

Prof. (Dr.) H. I. Joshi
Internal Guide

Prof. (Dr.) B.B.KUCHHADIYA
Head of the Department



**Government Engineering College
Rajkot**

CERTIFICATE

This is to certify that the project report submitted along with the project entitled **Automation And Solution Of 2d Truss Element Problem Using Python And Machine Learning** has been carried out by **Pandya Manan Bharatbhai** under my guidance in partial fulfillment for the degree of Bachelor of Engineering in Mechanical Engineering, 8th Semester of Gujarat Technological University, Ahmadabad during the academic year 2022-23.

Prof. (Dr.) H. I. Joshi
Internal Guide

Prof. (Dr.) B.B.KUCHHADIYA
Head of the Department



**Government Engineering College
Rajkot**

CERTIFICATE

This is to certify that the project report submitted along with the project entitled **Automation And Solution Of 2d Truss Element Problem Using Python And Machine Learning** has been carried out by **Pitroda Kishan Hasmukhbhai** under my guidance in partial fulfillment for the degree of Bachelor of Engineering in Mechanical Engineering, 8th Semester of Gujarat Technological University, Ahmadabad during the academic year 2022-23.

Prof. (Dr.) H. I. Joshi
Internal Guide

Prof. (Dr.) B.B.KUCHHADIYA
Head of the Department



**Government Engineering College
Rajkot**

DECLARATION

We hereby declare that the Project report submitted along with the Project entitled **Automation And Solution Of 2d Truss Element Problem Using Python And Machine Learning** submitted in partial fulfillment for the degree of Bachelor of Engineering in Mechanical Engineering to Gujarat Technological University, Ahmedabad, is a bonafide record of original project work carried out by us at Government Engineering College, Rajkot under the supervision of Prof. (Dr.) H. I. Joshi and that no part of this report has been directly copied from any students' reports or taken from any other source, without providing due reference.

| Sr. | Name of student | Sign of Student |
|-----|----------------------------|-----------------|
| 1 | Makwana Rahul Premjibhai | |
| 2 | Pandya Manan Bharatbhai | |
| 3 | Pitroda Kishan Hasmukhbhai | |

ACKNOWLEDGEMENT

The success and outcome of our project required a lot of guidance and assistance from many people. We are extremely privileged to have got this all along the completion of our project.

We respect and thank all our colleagues for providing us an opportunity to do the project work and giving us all support and guidance wherever required, which made us capable to complete this project.

We owe our deep gratitude to our project guides, Prof. (Dr.) H. I. Joshi whose Encouragement, Supervision, Support, keen interest, and Mentorship from the preliminary stage to the concluding stage enabled us to develop an understanding of the subject.

We heartily thank our Head of Department Prof. (Dr.) B.B.KUCHHADIYA for his guidance and suggestions during this project work.

We are thankful to all faculties of our college “Government Engineering College, Rajkot” for helping us in every way possible.

Lastly, we offer our regards and thanks to all of those who supported us in any manner during the completion of our dissertation.

-Makwana Rahul Premjibhai
Pandya Manan Bharatbhai
Pitroda Kishan Hasmukhbhai

ABSTRACT

With widespread use of machine learning and capabilities to use Finite element method in different areas of engineering, Generalization of these methods are necessary so everyone can use such complex methods as tools without needing to learn the process. This report gives a feasible solution to the said problem. This report explains the construction of the python program for image processing using gaussian blur, canny edge detection, Hough probabilistic transform and machine learning unsupervised K-clustering method. Also, it demonstrates how python can be used to solve 2d truss problems of any given geometry.

Keywords: Python, Machine learning, Image processing, K-clustering, 2d truss, stress

LIST OF FIGURES

| | |
|--|----|
| Fig 2.4.1. Revolution of Automation..... | 7 |
| Fig 4.2.1. Import libraries..... | 17 |
| Fig 4.2.2. Inputs..... | 18 |
| Fig 4.2.3. Different lists..... | 18 |
| Fig 4.2.4. Calculation..... | 19 |
| Fig 4.3.1. Element stiffness matrix..... | 18 |
| Fig 4.3.2. Logic of global stiffness matrix..... | 20 |
| Fig 4.3.2. Logic for Boundary condition..... | 21 |
| Fig 4.4.1.1. Logic for supports..... | 22 |
| Fig 4.4.2.1. Logic for supports..... | 23 |
| Fig 4.5.1.1. Matrix reduction..... | 23 |
| Fig 4.5.2.1. Logic for supports..... | 24 |
| Fig 4.5.3.1. New coordinates..... | 25 |
| Fig 4.5.3.2. New lengths..... | 25 |
| Fig 4.5.4.1. Strain in element..... | 26 |
| Fig 4.5.5.1. Stress in elements..... | 26 |
| Fig 4.5.6.1. Element force..... | 27 |
| Fig 5.1. Loading image..... | 29 |
| Fig 5.2. Canny edge detection..... | 29 |
| Fig 5.3. Hough transform..... | 31 |
| Fig 5.4.1. Flattening..... | 32 |
| Fig 5.4.2. Storing..... | 33 |
| Fig 5.4.3. Color array..... | 33 |

| | |
|---|----|
| Fig 5.4.4. Segmenting..... | 34 |
| Fig 5.5.1. Function call read black pixel..... | 35 |
| Fig 5.5.2. Function call extract first ransac line..... | 35 |
| Fig 5.5.3. Function call generate plottable..... | 39 |
| Fig 5.5.4. Function call superimpose all linerers..... | 39 |
| Fig 5.5.5. Function call extract multiple line and saves..... | 37 |
| Fig 6.1.1. sum..... | 38 |
| Fig 6.1.2. Nodes and elements..... | 39 |
| Fig 6.1.3. coordinates..... | 39 |
| Fig 6.1.4. Modulus of elasticity and cross section area..... | 39 |
| Fig 6.1.5. Node connectivity..... | 39 |
| Fig 6.1.6. Global stiffness matrix..... | 40 |
| Fig 6.1.6. Loads..... | 40 |
| Fig 6.1.7. supports..... | 41 |
| Fig 6.1.8. Displacement matrix..... | 41 |
| Fig 6.1.9. Strain matrix..... | 41 |
| Fig 6.1.10. Stress matrix..... | 42 |
| Fig 6.1.12. Force in element..... | 43 |
| Fig 6.2.1. Main input image..... | 43 |
| Fig 6.2.2. Gaussian blur..... | 44 |
| Fig 6.2.3. Canny edge detection..... | 44 |
| Fig 6.2.4. Probabilistic Hough..... | 45 |
| Fig 6.2.5. Dominant colors..... | 45 |
| Fig 6.2.6. K-cluster..... | 46 |

| | |
|---|----|
| Fig 7.1. Enter node and element..... | 47 |
| Fig 7.2. Coordinates..... | 47 |
| Fig 7.3. other view for coordinates..... | 48 |
| Fig 7.4. Element connectivity..... | 48 |
| Fig 7.5. Other view for element connectivity..... | 49 |
| Fig 7.6. Support specification..... | 49 |
| Fig 7.7. Enter supports..... | 50 |
| Fig 7.8. Final results..... | 51 |

TABLE OF CONTENTS

| | |
|--|-----|
| ACKNOWLEDGEMENT..... | i |
| ABSTRACT..... | ii |
| LIST OF FIGURES..... | iii |
| TABLE OF CONTENTS..... | vi |
| Ch 1. INTRODUCTION..... | 1 |
| 1.1. GOAL AND CONTRIBUTION..... | 1 |
| 1.2. KEY CONTRIBUTIONS | 1 |
| Ch 2. LITERATURE REVIEW..... | 3 |
| 2.1. PAPER-1..... | 3 |
| 2.2. PAPER-2..... | 3 |
| 2.3. BACKGROUND..... | 4 |
| 2.3.1. Python..... | 4 |
| 2.3.2 Machine learning..... | 5 |
| 2.3.3 Machine learning Approaches..... | 5 |
| 2.3.4. Numerical Method..... | 6 |
| 2.4 Automation..... | 6 |
| CH 3. APPROACH METHODOLOGY..... | 8 |
| 3.1. APPROACH..... | 8 |
| 3.2. LIBRARIES..... | 11 |
| CH 4. PYTHON PROGRAMMING FOR TRUSS SOLUTION..... | 16 |
| 4.1. APPLICATON | 16 |
| 4.2. GLOBAL STIFFNESS MATRIX..... | 17 |

| | |
|---|----|
| 4.2.1. Importing The Libraries..... | 17 |
| 4.2.2. Getting The Inputs Form The Users..... | 18 |
| 4.2.3. Calculation of Element Stiffness Matrix..... | 19 |
| 4.2.4. Global Stiffness Matrix Assembling..... | 20 |
| 4.3 BOUNDARY CONDITIONS AND LOADING..... | 21 |
| 4.4. CREATING VARIABLE LISTS..... | 22 |
| 4.4.1. Defining Support Specifications..... | 22 |
| 4.4.2. Loading Conditions..... | 23 |
| 4.5. SOLVING THE MATRICES..... | 23 |
| 4.5.1. Matrix Reduction..... | 23 |
| 4.5.2. Solving The Matrices..... | 24 |
| 4.5.3. New Coordinates | 25 |
| 4.5.4. Strain In Elements..... | 26 |
| 4.5.5. Stress In Elements..... | 26 |
| 4.5.6. Element Forces..... | 27 |
| CH 5. IMAGE PROCESSING..... | 28 |
| 5.1. PREPROCESSING THE IMAGE..... | 28 |
| 5.2. DETECTING EDGES..... | 29 |
| 5.3. HOUGH TRANSFORM..... | 30 |
| 5.4. SEGMENTING TRUSS..... | 31 |
| 5.4.1. Flattening Each Channel Of Image..... | 32 |
| 5.4.2. Storing Info In Color Array..... | 32 |
| 5.4.3. Creating New Clustered Image..... | 33 |
| 5.5. TRUSS IDENTIFICATION..... | 34 |

| | |
|---|----|
| CH 6. OUTPUT OF CODES..... | 38 |
| 6.1. OUTPUT OF 2D TRUSS ANALYSIS..... | 38 |
| 6.1.1. Entering Total Number Of Nodes And Elements..... | 38 |
| 6.1.2. Entering Node Coordinates..... | 39 |
| 6.1.3. Entering Cross Section Area and Modulus Of Elasticity..... | 39 |
| 6.1.4. Entering Start And End Nodes Of Elements..... | 39 |
| 6.1.5. Global Stiffness Matrix..... | 40 |
| 6.1.6. Entering Boundary Conditions..... | 40 |
| 6.1.7. Answers..... | 41 |
| 6.2. IMAGE PROCESSING OUTPUT..... | 42 |
| 6.2.1. This Image Is Given As Input..... | 42 |
| 6.2.2. Gaussian Blur..... | 43 |
| 6.2.3. Canny Edge Detection..... | 44 |
| 6.2.3. Probabilistic Hough Transform..... | 45 |
| 6.2.5. Dominant Colors..... | 45 |
| 6.2.6. K Cluster Image..... | 45 |
| CH 7. WEBSITE..... | 47 |
| CH 8. FUTURE DEVELOPMENTS..... | 52 |
| Ch 9. CONCLUSION..... | 53 |
| CH 10. REFERENCE..... | 54 |

1. INTRODUCTION

Finite Element Analysis (FEA) is a crucial engineering tool that enables the simulation and analysis of complex systems across various industries. However, FEA sum calculations present significant challenges due to their complicated nature and dependence on specialized software like ANSYS, which is both time-consuming and requires a high level of expertise. As a result, engineers and researchers are often faced with the daunting task of overcoming these obstacles in order to produce efficient and accurate results.

This report investigates the potential of using Python and Machine Learning (ML) as a feasible solution to address the challenges associated with FEA sum calculations. By leveraging the adaptability and user-friendliness of Python, along with the predictive power of ML algorithms, we aim to streamline the FEA sum calculation process, reducing the time and computational effort required while maintaining accuracy.

Through the integration of Python's extensive library ecosystem and ML frameworks, we can develop tailored models to tackle specific FEA challenges, thereby automating and optimizing various aspects of the calculation process. This approach not only alleviates the burden of relying solely on ANSYS, but also democratizes FEA capabilities, making them accessible to a wider range of engineers and researchers.

In this report, we will explore the technical aspects of combining Python and ML for FEA sum calculations, as well as highlight real-world applications and case studies to demonstrate the effectiveness of this innovative approach. By embracing Python and ML-driven solutions, engineers can overcome the limitations of traditional FEA software, leading to more efficient and accurate engineering outcomes.

1.1. GOAL AND CONTRIBUTION

The primary goal of this project is to develop an easy-to-use Python tool that streamlines truss sum calculations in Finite Element Analysis (FEA), making truss analysis more accessible to engineers and researchers of varying expertise levels.

1.2. OUR PROJECT'S KEY CONTRIBUTIONS INCLUDE

Development of a flexible and efficient solution for truss sum calculations, designed to accommodate two distinct input methods. This versatility ensures that users can choose the method that best suits their needs and expertise.

Implementation of a direct input method, allowing users to provide coordinates and related data to describe the truss structure. This option caters to users who prefer a more traditional approach or have specific data readily available.

On the other hand, devising an innovative technique that enables users to take a photo of the truss system, input basic details such as dimensions and material properties, and compute the truss sum. This pioneering approach leverages image processing methods to extract essential truss structure information from the provided image, making the process more intuitive and user-friendly. Streamlining the FEA process by offering multiple input choices and automating truss information extraction. This optimization substantially reduces the time and effort required for truss sum calculations, increasing overall efficiency.

Broadening access to FEA capabilities through the integration of image processing techniques, enabling a more diverse group of engineers and researchers to carry out truss analyses without the need for specialized software or in-depth expertise. This democratization of FEA helps to foster greater innovation and collaboration within the field.

2. LITERATURE REVIEW

2.1. SEGMENTATION AND ANALYSIS OF A SKETCHED TRUSS FRAME USING MORPHOLOGICAL IMAGE PROCESSING TECHNIQUES

The field of structural analysis has witnessed significant advancements with the introduction of image processing techniques, particularly in the evaluation of hand-sketched or computer-generated truss frames. This literature review explores various scholarly sources that delve into the application of these techniques and the potential of automated structural analysis. The literature reveals a growing trend of automated structural analysis, emphasizing its potential in the field. This involves the application of image processing techniques to automate the complex and time-consuming process of structural analysis. This potential is particularly evident in the examination of hand-sketched or digitally generated truss frames. Future research directions revealed in the literature include the application of these techniques to analyze hand-sketched fixed 2D rigid frames. This proposition presents a unique opportunity to automate the analysis of these frames while maintaining the level of detail and accuracy required in structural engineering. Literature also proposes the potential for visualizing and overlaying the deformed shape of a frame, as well as the shear and moment forces for any member, using similar methodologies. This capability would significantly enhance the comprehensiveness of the structural analysis.

2.2. AN AUTOMATED DESIGN METHOD FOR PLANE TRUSSES BASED ON USER PREFERENCE INFORMATION

This paper presents a novel design method called STSA-P for plane trusses, which seeks to incorporate user preference into the structural design process. The proposed method utilizes a prediction model to quantify user preference information and transform it into an additional design objective. By allowing the coordination of multiple qualitative and quantitative objectives, the STSA-P method enables user preferences to guide the search process and explore various designs. The method's design is intended to reduce user fatigue and dependence by requiring only a short interaction and providing reusable preference information. The effectiveness of the proposed method is validated through a design example. The construction of the prediction model involves two steps: generating a valid

dataset and selecting an appropriate machine learning technique. The paper outlines a series of subjective and numerical experiments to validate the rationality of these steps and the accuracy of the prediction model. To coordinate user preference with other objectives of structural performance, the paper proposes an aggregate objective function based on the Modified Physical Programming (MPP) method. The cost function incorporates design constraints to effectively integrate the prediction model into the STSA method. While the STSA-P method currently considers only two structural performance objectives, i.e., structural weight and displacement, the paper suggests that further research could expand the method's functionality by incorporating additional design objectives, such as structural stiffness and natural frequency. Overall, the proposed STSA-P method provides a promising framework for incorporating user preference into the structural design process. The method's use of machine learning and the MPP method provides a robust approach to coordinating multiple objectives and constraints to generate designs that meet both user preferences and structural performance objectives. The paper's empirical evaluation of the method's effectiveness further demonstrates its potential for practical applications.

2.3. BACKGROUND

2.3.1. Python:

Python is a general-purpose, high-level programming language that was created by Guido van Rossum in the late 1980s and first released in 1991. The language was designed to be easy to learn, read, and write, and it has since grown to become one of the most popular programming languages in the world.

In the mechanical engineering field, Python is used extensively for engineering simulations, data analysis, and scientific computing. Python's simplicity, readability, and versatility make it an ideal tool for mechanical engineers who need to analyze large amounts of data or simulate complex systems. Python's ability to integrate with other software tools has also made it an important tool for engineers who need to create more efficient workflows and reduce errors.

Finite Element Analysis (FEA) is a numerical method for solving complex engineering problems using a computer. Python's popularity and flexibility have made it a popular choice for developing FEA software. Python has a large and active community of developers who have created open-source libraries such as NumPy, SciPy, and Matplotlib,

which provide a rich environment for numerical computing, data analysis, and visualization.

Python's development in FEA has revolutionized the way engineers design and analyze complex mechanical systems. Python has enabled engineers to create more efficient workflows, simulate more complex systems, and reduce the time and cost associated with traditional FEA methods. Python has also made FEA more accessible to a wider range of engineers, allowing them to solve problems that were previously out of reach.

In conclusion, Python's simplicity, versatility, and popularity have made it an essential tool for mechanical engineers in the analysis and simulation of complex mechanical systems. Python's development in FEA has enabled engineers to create more efficient workflows, simulate more complex systems, and solve problems that were previously out of reach.

2.3.2. Machine learning

Machine learning (ML), a subset of AI, evolved from the 18th-century Bayes' theorem, gaining significant momentum in the late 20th century with algorithms like perceptron and backpropagation, alongside the rise of the internet and advanced hardware.

In mechanical engineering, ML optimizes processes, predicts maintenance needs, and aids in efficient product design. For instance, ML algorithms, by learning from past data, can predict machinery failures, reducing downtime.

Finite Element Analysis (FEA) has also been revolutionized by ML. By learning from previous simulations and physical experiments, ML improves the speed and accuracy of FEA, thereby expediting the design process.

The integration of ML with image processing promises improved object detection, better image analysis, real-time processing, and more interactive AR and VR experiences. This has potential impacts across various sectors, including healthcare, autonomous vehicles, and manufacturing, and is likely to shape the future of technology development.

2.3.3. Machine learning Approaches

- **Supervised Learning** - In Supervised Machine learning, the data set is labelled, i.e., for each feature or independent variable, there is a matching target data set that we would use to train the model.

- **Unsupervised Learning** - Unlike Supervised Learning, the data set is not labelled in this situation. Thus, the clustering approach is used to categorise data based on its similarity among data points in the same group.
- **Reinforcement Learning** - A subset of Machine Learning in which the model is trained from each action made. The model is rewarded for right judgments and penalised for incorrect decisions, allowing it to learn patterns and make more accurate decisions on unknown data.
- **Semi-supervised learning** - learning algorithms that lie between unsupervised (no labelled training data) and supervised (fully labelled training data). It is employed when there is a lack of labelled data is difficult or expensive.

2.3.4. Numerical Method

The development of the finite element method (FEM) started in the 1950s with the work of Richard Courant, who is often considered the pioneer of FEM. The method was initially used for solving complex structural analysis problems in civil and aeronautical engineering. However, its use has since expanded to include a wide range of engineering and physics problems.

FEM was developed out of the need to solve complex elasticity and structural analysis problems in civil and aeronautical engineering. These problems were either unsolvable or too difficult to solve using traditional analytical methods, especially when dealing with complex geometries, non-homogeneous materials, or boundary conditions. (Finite element analysis (FEA) is a computational method used to predict how a product reacts to real-world forces, vibration, heat, fluid flow, and other physical effects. It is a critical part of many engineering fields, including mechanical, civil, aeronautical, and biomedical engineering.)

2.4. Automation

The concept of automation has been in existence for a long time, dating back to the industrial revolution in the 18th and 19th centuries, where machinery started to be used to enhance productivity and reduce manual labour. The term "automation" was first coined in the automobile industry in the 1940s. The Ford Motor Company set up a division called the Automation Division, which was responsible for the routine jobs on the assembly line.

- First Industrial Revolution: Transition from agrarian to industrial society, driven by mechanization, steam power, and railways.
- Second Industrial Revolution: Introduction of mass production techniques, interchangeable parts, assembly lines, and advancements in electricity, communication, and transportation image
- Third Industrial Revolution: The Digital Revolution, marked by the shift from analog to digital technology, including computers and the internet.
- Fourth Industrial Revolution: Industry 4.0, characterized by the fusion of physical, digital, and biological technologies, such as AI, IoT, and genetic engineering image.

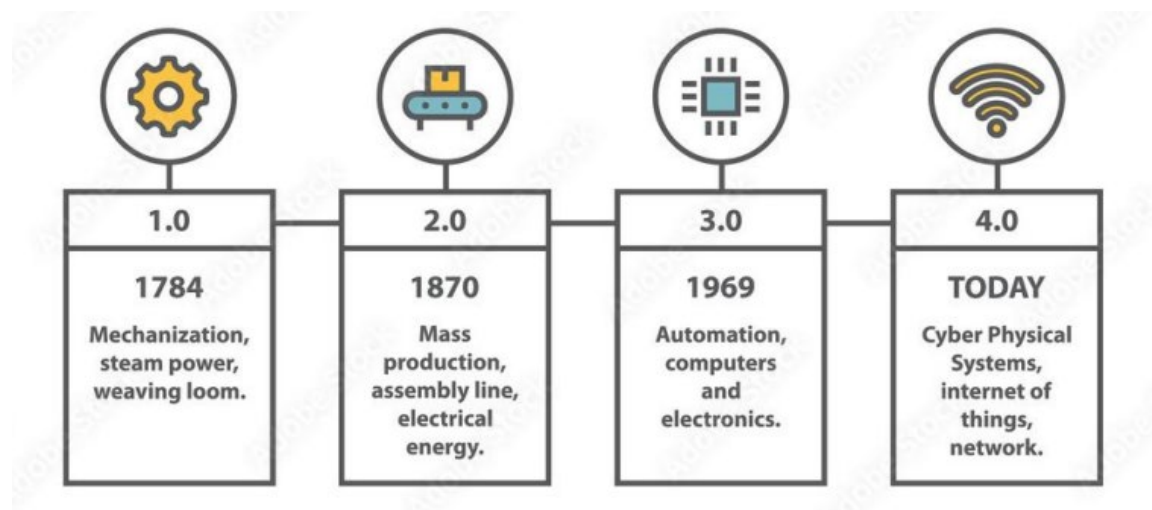


Fig 2.4.1 Revolution of Automation

As mentioned above, automation in this project means we develop one process in which users have not enter all kinds of basic data for calculation.

3. APPROACH METHODOLOGY

3.1. APPROACH

Finite Element Analysis (FEA) is a numerical method used for predicting how a real-world object will react to forces, vibration, heat, and other physical effects. It subdivides a large problem into smaller, simpler, parts that are called finite elements. The simple equations that model these finite elements are then assembled into a larger system of equations that models the entire problem.

In terms of truss problems, these are a common application of FEA, where the truss is broken down into finite elements (usually the individual truss members) and the forces and stresses are calculated for each.

we focus on FEA truss problems, which involve the analysis of truss structures subjected to various loads and constraints. Several methods can be employed to solve these problems, including.

1. **Gaussian Method**
2. **Elimination Method**
3. **Python**
4. **CAD Software**
5. **Machine Learning (ML)**
6. **MATLAB**

1. **Gaussian Method:** Also known as Gaussian elimination, this is a method for solving linear systems of equations. It involves three types of elementary row operations:
 - Swap the position of two rows.
 - Multiply a row by a non-zero scalar.
 - Add or subtract one row to another.

The goal is to transform the matrix to row-echelon form (or even reduced row-echelon form), which makes the system of equations easier to solve. Gaussian

elimination is fundamental to many methods in numerical linear algebra, such as LU and QR factorizations, and is used to solve linear least squares problems as well.

2. **Elimination Method:** Similar to the Gaussian method, The elimination method is a technique employed to solve systems of linear equations, bearing similarities to the Gaussian method. This method is characterized by its simplicity, as it involves manipulating the equations to eliminate one variable at a time through addition or subtraction. Although it may not be as efficient or universally applicable as Gaussian elimination, the elimination method is easier to comprehend and utilize, particularly for smaller systems of equations.

Advantages:

- Applicable to any size of linear system
- Can be easily implemented in various programming languages

Disadvantages:

- Computationally expensive for large systems
- Prone to numerical errors due to round-off and truncation

3. **Python:** Python is a general-purpose programming language that is widely used in scientific computing. When conducting Finite Element Analysis (FEA) for truss problems, one would typically begin by defining the truss's nodes, elements, applied forces, and constraints. Utilizing libraries such as NumPy and SciPy allows for the creation of stiffness matrices and the resolution of equation systems. Moreover, the matplotlib library can be employed to visualize both the truss and the resulting data. Specialized FEA libraries, such as pyFEM, are also available for use in Python.

Advantages:

- Open-source and free to use
- Extensive libraries for scientific computing
- Easy to learn and implement

Disadvantages:

- Slower than other programming languages like C++ or Fortran
- May require additional libraries for advanced FEA capabilities

4. **CAD Software:** Computer-Aided Design (CAD) software is a crucial tool in the creation of accurate drawings and technical illustrations, both in two and three

dimensions. One of the key features of CAD software is its ability to perform Finite Element Analysis (FEA), which allows users to analyze and solve complex engineering problems related to structures, such as trusses. Some popular CAD software packages with FEA capabilities include AutoCAD, SolidWorks, and ANSYS.

By utilizing these software packages, engineers can construct a truss model, define the material properties, and apply the necessary loads and constraints. The built-in FEA tool then assists in solving the problem by providing valuable data and visualizations of the displacements, stresses, and strains within the truss. This information is essential for ensuring the structural integrity and safety of the truss in real-world applications.

Advantages:

- Intuitive graphical user interface
- Advanced tools for modelling and analysis
- Integration with other engineering software

Disadvantages:

- Expensive licensing fees
- Steeper learning curve compared to programming languages
- Limited customization and flexibility

5. **Machine Learning (ML):** Machine Learning (ML) methods can be used in conjunction with FEA to enhance the design and analysis process. For instance, ML can be used to create predictive models for the performance of a truss structure under various loading conditions. This can aid in the design process by providing insights into how changes in the design may affect the performance of the structure.

In addition, ML can be used to optimize the design of a truss. By training a model on a range of potential designs and their performance, an optimal design can be identified. This can save time and resources by reducing the need for extensive manual design iteration and testing.

Advantages:

- Fast prediction of truss behaviour
- Can handle complex, non-linear relationships
- Continuously improves with more data

Disadvantages:

- Requires large datasets for training
- Limited interpretability of the model
- May not be suitable for all FEA truss problems

6. **MATLAB:** MATLAB is a high-level programming language and interactive environment that is widely used in engineering and scientific computing. It has built-in functions for matrix operations, solving systems of equations, and visualizing data, making it a powerful tool for FEA.

In addition to its core capabilities, MATLAB also has toolboxes specifically designed for FEA. These toolboxes provide additional functions for tasks like mesh generation, application of boundary conditions, and solution of the system of equations. They also often include features for advanced analysis types, like non-linear and dynamic analysis.

Advantages:

- Comprehensive libraries for FEA and numerical computing
- Easy to learn and implement
- Excellent visualization capabilities

Disadvantages:

- Expensive licensing fees
- Slower than other programming languages like C++ or Fortran
- May require additional toolboxes for advanced FEA capabilities

3.2. LIBRARIES

for our purpose we have chosen to use python as our main coding language to solve the 2d truss problem using gaussian elimination method and then using machine learning for image processing to extract geometry from an image.

Here are the libraries used in the project.

1. NumPy:

NumPy (short for Numerical Python) is a Python library that is used for scientific computing, data analysis, and numerical computations. It provides efficient array and matrix operations that can handle large amounts of data and is widely used in the scientific and engineering communities.

NumPy is built around a powerful N-dimensional array object, which allows for fast and efficient computations on arrays of any dimensionality. The library provides a large number of mathematical functions, including basic operations like addition, subtraction, multiplication, and division, as well as more advanced functions like Fourier transforms, linear algebra operations, and random number generation.

NumPy is also designed to be highly compatible with other scientific computing libraries in Python, such as Pandas (for data analysis) and Matplotlib (for data visualization). Overall, NumPy is an essential tool for anyone working with scientific or numerical data in Python, and is widely used in fields such as physics, engineering, finance, and more.

2. Math:

The math library in Python provides a set of functions that allow you to perform mathematical operations on numeric data. Some of the functions in the math library include:

- i. Trigonometric functions: The math library provides functions to calculate the sine, cosine, and tangent of an angle in radians.
- ii. Exponential and logarithmic functions: The math library provides functions to calculate the exponential and logarithmic values of a number.
- iii. Basic arithmetic functions: The math library provides functions to perform basic arithmetic operations like addition, subtraction, multiplication, and division.
- iv. Statistical functions: The math library provides functions to calculate statistical properties of a set of data like mean, median, and standard deviation.
- v. Special functions: The math library also includes a number of special functions like the gamma function and the error function.

3. OpenCV:

OpenCV (Open Source Computer Vision Library) is a popular open-source computer vision and machine learning software library written in C++. However, it also provides interfaces for Python and other programming languages. OpenCV is widely used for real-time computer vision applications, such as facial recognition, object detection, tracking, and image processing.

OpenCV offers a vast collection of image processing and computer vision algorithms, including:

Image processing functions: such as filtering, transformations, and feature extraction.

Feature detection and description: such as edge detection, corner detection, and keypoint detection.

Object detection: such as Haar cascades, HOG (Histogram of Oriented Gradients), and deep learning-based object detection.

Tracking: such as optical flow, object tracking, and multi-object tracking.

Machine learning: OpenCV offers several machine learning algorithms for classification, clustering, and regression, among others.

Camera calibration: OpenCV provides functions for camera calibration and 3D reconstruction.

4. OS:

The OS (Operating System) module in Python provides a way to interact with the underlying operating system that Python is running on. It provides a simple and consistent interface to access various operating system functions such as file operations, process management, environment variables, etc.

Here are some of the key functionalities provided by the OS module in Python:

The OS module provides a number of functions for file and directory operations, such as creating and deleting files and directories, changing file permissions, and more the OS module allows you to manage processes running on the system, such as starting new processes, killing existing ones, and getting information about running processes. The OS module allows you to access and modify environment variables on the system, which are useful for configuring various system settings. The OS module provides various other functions, such as retrieving information about the current working directory, checking whether a file or directory exists, and more.

5. scikit-learn:

Scikit-learn, also known as sklearn, is a popular machine learning library in Python that provides a wide range of tools for data analysis and modelling. It is built on top of NumPy, SciPy, and matplotlib, and is designed to work well with other libraries in the scientific Python ecosystem.

The sklearn library provides various machine learning algorithms, including classification, regression, clustering, and dimensionality reduction. These algorithms can be used for tasks such as predicting outcomes, identifying patterns, and discovering relationships in data. Sklearn also provides tools for model selection, pre-processing, and evaluation, making it a powerful tool for data science and machine learning tasks.

Sklearn is easy to use and has a simple, consistent API, making it a popular choice for both beginners and experienced machine learning practitioners. It also has excellent documentation and a vibrant community of users, which provides support and resources for those looking to learn more about machine learning in Python.

6. Matplotlib:

Matplotlib is a popular plotting library for Python that provides a wide range of visualization tools. It allows users to create a variety of plots such as line, bar, scatter, histogram, and pie charts, as well as 3D plots and heatmaps. Matplotlib is built on top of NumPy and provides a high-level interface for creating and manipulating plots.

Matplotlib provides a number of different APIs for creating plots. The most commonly used is the pyplot API, which provides a simple interface for creating and customizing plots. This API is similar to the plotting functionality of MATLAB, making it familiar to users who have experience with that software.

Matplotlib also provides a low-level object-oriented API that allows for more complex and customized plots. This API gives users fine-grained control over the appearance and behaviour of plots, allowing them to create highly specialized visualizations.

In addition to creating static plots, Matplotlib also provides tools for creating interactive plots that can be embedded in web pages or applications. This is achieved through the use of the matplotlib widget toolkit, which provides a set of graphical user interface (GUI) elements that can be used to create interactive plots.

7. Scikit-image:

Scikit-image (skimage) is an open-source image processing library for Python that is built on top of the scientific Python ecosystem. It provides a comprehensive set of functions for processing and analyzing images, including algorithms for segmentation, feature extraction, and filtering.

The library contains a wide range of functions for image processing tasks, including color and brightness adjustments, image filtering, edge detection, morphology, and image segmentation. It also provides tools for feature detection and extraction, including the HOG (Histogram of Oriented Gradients) feature descriptor, which is widely used for object detection and recognition.

Skimage also includes functions for image restoration, such as deconvolution, and for image registration and transformation. It also provides tools for working with image sequences, including video processing and tracking.

Skimage has a user-friendly API that makes it easy to use and integrate with other Python libraries, such as NumPy, SciPy, and Matplotlib. It is widely used in research and industry for a variety of applications, including computer vision, remote sensing, and medical imaging.

4. PYTHON PROGRAMMING FOR TRUSS SOLUTION

Python is used by millions of engineers and scientist around the world to analyze and design the systems and products that are transforming our world.

Python is a high-level, interpreted programming language that was first released in 1991. It is an open-source language that can be used for a wide range of applications, including web development, scientific computing, data analysis, artificial intelligence, and more.

Python's popularity has grown steadily over the years, and it has become one of the most widely used programming languages in the world. It is known for its simple syntax, which makes it easy to read and write, as well as its extensive library of modules, which provide access to a wide range of functionality.

4.1. APPLICATION IN ENGINEERING ANALYSIS AND MACHINE LEARNING

Python is used in various engineering fields for data analysis, modeling, and simulation. Engineers use Python libraries like NumPy, SciPy, and Pandas to perform mathematical computations, data manipulation, and data visualization. Python can also be used to develop engineering software for various applications like structural analysis, fluid dynamics, and control systems. Also Python is one of the most popular languages for machine learning and artificial intelligence (AI). Python libraries like TensorFlow, Keras, and PyTorch are commonly used for building machine learning models, including neural networks, deep learning, and natural language processing. Python is also used for data preprocessing, feature engineering, and model evaluation.

Python has several libraries and frameworks that are commonly used for image processing tasks, making it a popular choice for image processing applications. Here are some of the key ways Python is used in image processing:

OpenCV: OpenCV is an open-source computer vision library that is widely used for image and video processing. It has a Python API that allows developers to easily integrate it with their Python code.

Scikit-Image: Scikit-Image is a Python library that is used for image processing, computer vision, and machine learning. It provides a range of image processing algorithms and

functions that can be used for tasks like image filtering, segmentation, and feature extraction.

PIL/Pillow: Python Imaging Library (PIL) is an open-source library that adds support for opening, manipulating, and saving many different image file formats. Pillow is a fork of PIL that adds support for Python 3.

TensorFlow: TensorFlow is a popular machine learning framework that can be used for a range of applications, including image processing. It provides a range of pre-trained models that can be used for tasks like object detection and image classification.

Keras: Keras is a high-level neural networks API that is built on top of TensorFlow. It provides a range of pre-trained models that can be used for image classification, object detection, and other image processing tasks.

Overall, Python's flexibility, ease of use, and extensive library support make it a popular choice for image processing tasks, especially in research and development settings.

4.2 CODE LOGIC FOR SOLVING 2D TRUSS PROBLEMS VIA PYTHON: CREATING GLOBAL STIFFNESS MATRIX

4.2.1 Importing The Libraries

first of all MATH library and NUMPY library which are used for mathematical and array operations in python respectively are imported.

```
import math
import numpy
```

Fig 4.2.1 Import libraries

4.2.2 Getting The Inputs Form The Users.

```
tn = int(input('Enter the total number of nodes : ')) #total nodes
te = int(input('Enter the total number of Elements : ')) #total elements
xco = [] #x co ordinate of nodes
yco = [] #y co ordinate of nodes
for i in range(tn):
    x = float(input('Enter the x co-ordinate of node '+str(i+1)+' in mm : '))
    y = float(input('Enter the y co-ordinate of node '+str(i+1)+' in mm : '))
    xco.append(x)
    yco.append(y)
```

Fig 4.2.2 Inputs

“tn” represents total number of nodes and “te” represents total number of elements. By using a for loop bounded by the total number of nodes. X and Y coordinates of each nodes are taken from user, which are then stored in their respective list

After that the value of cross section area and modulus of elasticity is taken

```
snofel = [] #start node of elements
enofel = [] #end node of elements
lenofel = [] #length of the element
elcon = [] #constant of the element
cosofel = [] #cos of element
sinofel = [] #sin of element
```

Fig 4.2.3 Different lists

These are the lists will be required for storing mode data taken from user and results of some calculations like length of the element and the element constant AE/L

In the figure below we can see that by using a for loop bounded by “te” the start and end nodes of an element are taken from the user

By them list of coordinates is sliced and only the coordinates of the start and end points are taken and stored in the coordinates x1,y1,x2,y2 for further calculation

In list (a-1) component is taken because in python index starts from 0 so to call the nth member of the list, n-1 position on the list needs to be called .

By using the formula of length from the coordinates we can calculate the length of the elements which is then stored in the “lenofel” list

Then the element constant and the sin and cos value for further calculation is calculated

```
for i in range(te):
    a = int(input('Enter the Start node of element '+str(i+1)+' : '))
    b = int(input('Enter the End node of element '+str(i+1)+' : '))
    x1 = float(xco[a-1])
    y1 = float(yco[a-1])
    x2 = float(xco[b-1])
    y2 = float(yco[b-1])
    l = math.sqrt((x2-x1)**2+(y2-y1)**2)
    con = A*E/l
    cos = (x2-x1)/l
    sin = (y2-y1)/l

    snofel.append(a)
    enofel.append(b)
    lenofel.append(l)
    elcon.append(con)
    cosofel.append(cos)
    sinofel.append(sin)
```

Fig 4.2.4 Calculation

4.2.3 Calculation of Element Stiffness Matrix

```
elstmat = [] #element stiffness matrix

for i in range(te):
    cc = float(cosofel[i])**2
    ss = float(sinofel[i])**2
    cs = float(cosofel[i])*float(sinofel[i])

    mat = elcon[i]*numpy.array([[cc, cs, -cc, -cs],
                                [cs, ss, -cs, -ss],
                                [-cc, -cs, cc, cs],
                                [-cs, -ss, cs, ss]])

    elstmat.append(mat)
##print(elstmat)
```

Fig 4.3.1 Element stiffness matrix

As shown in image first an empty list for element stiffness matrix is created to store them. We know that total number of EST(element stiffness matrix) is equal to the total number of elements, hence using a for loop with range “te”

where “cc” is the square of cos of element and “ss” is square of the sin of element calculated previously and “cs” is the multiplication of the cos and sin

Then numpy library is used to create the array which represents the element stiffness matrix in gaussian method for solving 2d truss problem and element constant is taken from the “elcon” list made before and is multiplied by the matrix, after that it is saved on a variable called “mat”

Mat is then appended in the empty elstmat list and loop continues, after the completion of the loop element stiffness matrixes are stored in a list which will be used to create global stiffness-matrix.

4.2.4 Global Stiffness Matrix Assembling

```

gstmatmap = []
for i in range(te):
    m = snofel[i]*2
    n = enofel[i]*2
    add = [m-1, m, n-1, n]

    gmat = numpy.zeros((tn*2, tn*2))
    elmat = elstmat[i]
    for j in range(4):
        for k in range(4):
            a = add[j]-1
            b = add[k]-1
            gmat[a,b] = elmat[j,k]
    gstmatmap.append(gmat)
## print(numpy.around(gmat, 3))

GSM = numpy.zeros((tn*2, tn*2))
for mat in gstmatmap:
    GSM = GSM+mat

print('\nGlobal Stiffness Matrix of the Truss\n')
print(numpy.around(GSM, 3))

```

Global stiffness matrix mapping, gstmatmap will be the square matrix of tn*
do this for each elements
taking the start node of element(i) and multiply by 2
taking the end node of element(i) and multiply by 2
Address of columns and rows of gstmatmap for element(i)
if startnode is 1 and end node is 2 then add=[1,2,3,4]
if startnode is 1 and end node is 3 then add=[1,2,5,6]
global stiffness matrix loaded with zeros for element(i)
taking the element stiffness matrix of element(i)
addressing row of GST matrix for element(i)
addressing column of GST matrix for element(i)
updating the values in GST matrix with EST matrix of element(i)
storing the resultant matrix in gstmatmap list
creating an empty GSM matrix
adding all the matrix in the gstmatmap list
this will result in assembled stiffness matrix of the truss structure

Fig 4.3.2 Logic of global stiffness matrix

To get the global stiffness matrix, first mapping matrixes are created where all values are 0 and only values which are none zero places are occupied by the values of individual element stiffness matrix values then storing these matrixes and adding them will provide global stiffness matrix. For that an empty list called “gstmatmap” created then for “te” a for loop is created. “m” and “n” are constants which has the value of start and end of elements multiplied by 2 so for example. An element is started at node 2 and ends at node 4 so. $m=2*2=4$ and $n=4*2=8$ then a row matrix is generated and stored in variable “add” so here add will be [1,2,7,8] which are the row and column positions where the values of element stiffness matrix needs to be added in global stiffness matrix, for that zero matrix with $2*total\ nodes$ is created because all nodes have 2 degree of freedoms, which is why the global stiffness matrix will be and square $2*tn \times 2*tn$ matrix.

Now the individual value of EST are stored into “elmat” variable and for loop by range 4 is ran with another similar nested loop because each EST has 4 rows and 4 columns now “a” and “b” variable are the row and column numbers, then those values of EST is stored into the zero matrix one by one till this nested loop is over. This mapped zero matrix is appended into “gstmatmap”. By adding all these matrixes global stiffness matrix is created.

4.3 CODE LOGIC FOR SOLVING 2D TRUSS PROBLEMS VIA PYTHON: BOUNDARY CONDITIONS AND LOADING

```
displist = []
forcelist = []
for i in range(tn):
    a = str('u')+str(i+1)
    displist.append(a)
    b = str('v')+str(i+1)
    displist.append(b)
    c = str('fx')+str(i+1)
    forcelist.append(c)
    d = str('fy')+str(i+1)
    forcelist.append(d)
```

Fig 4.3.2 Logic for Boundary condition

4.4 CREATING VARIABLE LISTS TO

apply boundary conditions first a displacement list is created where the displacements labels (u1,v1,u2,v2.....) are stored and a force list is used where the force labels are stored (fx1,fy1,fx2,fy2.....) which will be associated with values further to do the calculation of stress and strain and to fix the boundaries by restricting the displacement values

4.4.1 Defining Support Specifications

```
dispmat = numpy.ones((tn*2,1))
tsupn = int(input('Enter the total number of nodes having supports : ')) #total number of supported nodes
supcondition = ['P = pinned',
                'H = Horizontal restrained (vertical is free to move)',
                'V = Vertical restrained (Horizontal is free to move)']

for i in range(tsupn):
    supn = int(input('\nEnter the node number of support : ')) #supported node
    for a in supcondition:
        print(a)
    condition = str(input('\nEnter the condition of the support : '))
    if condition in ['P', 'p']:
        dispmat[supn*2-2, 0] = 0
        dispmat[supn*2-1, 0] = 0
    elif condition in ['H', 'h']:
        dispmat[supn*2-2, 0] = 0
    elif condition in ['V', 'v']:
        dispmat[supn*2-1, 0] = 0
    else:
        print('Please enter valid entries')
```

Fig 4.4.1.1 Logic for supports

As shown in figure displacement matrix “dispmat” is created which is filled with 1, each node has 2 degrees of freedom so it will be a column matrix with dimension $tn*2 \times 1$. Then total number of supports are taken from the user and support conditions are taken for each individual nodes to restrict degree of freedoms in bounded nodes.

For that a for loop ranged by total number of supports is used then the number of the node which is supported is taken and then from the dictionary the “pinned”, “horizontal restrained”, “vertical restrained” are taken from users from the keywords “P”, “H”, “V” then from nested Else IF argument which condition is defined is chosen.

If the chosen node number is pinned then that nodes degree of freedom is restricted by giving U and V value of that node in “dispmat” as 0, for example if node 4 is pinned then values U4 and V4 will be 0. To get the positions in displacement matrix we can use similar

logic from global stiffness matrix calculation, similarly for horizontal restrained only U is 0 and for vertically restrained only V is 0 for chosen node.

4.4.2 Loading Conditions

```
print('\n_____Loading_____ \n')
forcemat = numpy.zeros((tn*2,1))
tlon = int(input('Enter the total number of loaded nodes : ')) #total number of loaded nodes

for i in range(tlon):
    lon = int(input('\nEnter the node number of Loading : ')) #Loaded node
    fx = float(input('Enter the Horizontal load at this node in N : '))
    fy = float(input('Enter the Vertical load at this node in N : '))
    forcemat[lon*2-2, 0] = fx
    forcemat[lon*2-1, 0] = fy
```

Fig 4.4.2.1 Logic for supports.

Similar logic as defining boundary condition can be applies here, instead of giving the zero values in the displacement matrix to restraint the node, to apply loading the loading values will be stored in the chosen node locations in the force matrices.

4.5 CODE LOGIC FOR SOLVING 2D TRUSS PROBLEMS VIA PYTHON: SOLVING THE MATRICES

4.5.1 Matrix Reduction

```
###_____Matrix Reduction_____###

rcdlist = []
for i in range(tn*2):
    if dispmat[i,0] == 0:
        rcdlist.append(i)

rrgsm = numpy.delete(GSM, rcdlist, 0) #row reduction
crgsm = numpy.delete(rrgsm, rcdlist, 1) #column reduction
rgsm = crgsm #reduced global stiffness matrix
rforcemat = numpy.delete(forcemat, rcdlist, 0) #reduced force mat
rdispmat = numpy.delete(dispmat, rcdlist, 0) #reduced disp mat
```

Fig 4.5.1.1 Matrix reduction

Now before solving the matrix, they are reduced for more efficient calculations so, firstly an empty list called “rcdlist” which represents rows and columns which needs to be removed to get reduced force, displacement and global stiffness matrix

The degrees of freedom which are restrained are the ones which are removed from the matrix so to get the positions of those nodes a for loop is used by range of $tn*2$. In that loop each value of displacement matrix is seen if the value is zero then that position is appended into “rcdlist” after that using `numpy.delete` rows from global stiffness matrix is deleted and new matrix is stored in “rrgsm” variable then from that columns are deleted and stored in “crgsm” variable which is then stored in “rsgm” which represents reduced global stiffness matrix. In similar way rows from force matrix and displacement matrix are removed.

4.5.2 Solving The Matrices

```
dispresult = numpy.matmul(numpy.linalg.inv(rsgm), rforcemat)
rin = 0
for i in range(tn*2):
    if dispmat[i,0] == 1:
        dispmat[i,0] = dispresult[rin,0]
        rin = rin+1
##print(dispmat)

forceresult = numpy.matmul(GSM, dispmat)
##print(forceresult)

print('\n\nGlobal Stiffness Matrix of the Truss\n')
print(GSM)
print('\n\nDisplacement matrix of nodes\n')
print(dispmat)
print('\n\nForce matrix of nodes\n')
print(forceresult)
```

Fig 4.5.2.1 Logic for supports

The matrices are solved by $KQ=F$, where k is reduced global stiffness matrix, Q is reduced displacement matrix and F is reduced force matrix.

So to find the value of displacement matrix we will need to make Q the subject so $[Q]=[K]^{-1} * [F]$ so by `numpy.mathmul` function results are taken and stored in “dispresult” variable.

To get the final displacement values we need to change the reduced matrix solution values into the normal displacement values so as shown in figure in “dispmat” when the position

for that iteration has the value one it is replaced with the same position value from “dispresult” matrix.

Then foceresult is calculated by $F=KQ$,

4.5.3 New Coordinates Of Nodes And New Length Of Members

```
## _____ new co ordinates of nodes _____ ####

newxco = []
newyco = []
count = 0
for i in range(tn):
    k = xco[i]+dispmat[count,0]
    newxco.append(k)
    count = count+1
    l = yco[i]+dispmat[count,0]
    newyco.append(l)
    count = count+1
```

Fig 4.5.3.1 New coordinates

New coordinates are found by adding the displacement results into their respective coordinates then those are saved in new lists called “newxco” for x coordinates and “newyco” for new y coordinates.

```
### _____ new length of memebers _____ ####

newlenofel = []
for i in range(te):
    a, b = snofel[i], enofel[i]
    x1 = float(newxco[a-1])
    y1 = float(newyco[a-1])
    x2 = float(newxco[b-1])
    y2 = float(newyco[b-1])
    l = math.sqrt((x2-x1)**2+(y2-y1)**2)
    newlenofel.append(l)
```

Fig 4.5.3.2 New lenghts

In similar way how previously the length of elements was calculated new lengths are calculated by same logic but with new coordinates.

4.5.4 Strain In Elements

```
###_____strain in elements_____###

numpy.set_printoptions(3, suppress=False)

elstrain = numpy.zeros((te,1))
for i in range(te):
    | elstrain[i,0] = (newlenofel[i]-lenofel[i])/(lenofel[i])
print('\n***Positive is Tensile\nNegative is Compressive***\n')

print('\n\nStrain in the elements')
print(elstrain)
numpy.set_printoptions(3, suppress=True)
```

Fig 4.5.4.1 Strain in element

Strain is calculated by the formula l/L where l is change in length of element and L is original length of element and the ratio gives the element strain which is then saved into an array making an element strain matrix for easy deduction.

4.5.5 Stress In Elements

```
###_____stress in elements_____###

elstress = numpy.zeros((te,1))
for i in range(te):
    | elstress[i,0] = E * elstrain[i,0]

print('\n\nStress in the elements')
print(elstress)
```

Fig 4.5.5.1 Stress in elements

Stress is calculated by hooks law $S=E*e$ where S is stress, E is modulus of elasticity and e is element strain . A for loop is used to take the values from element strain matrix and then after calculation stored into element stress matrix.

4.5.6 Element Forces

```
### _____ Member forces _____ #####  
  
eforce = numpy.zeros((te,1))  
for i in range(te):  
    eforce[i,0] = A * elstress[i,0]  
  
print('\n\nForce in the element')  
print(eforce)
```

Fig 4.5.6.1 Element force

Element force matrix is made in similar way shown previously in 4.3.7. for element force matrix $S=F/A$ is used where F is force in element and A is cross section area of element and S is stress, so to find force $F=A*S$.

5. IMAGE PROCESSING

Image processing is a field of study that involves the manipulation of digital images using computer algorithms. It involves techniques for improving image quality, enhancing image features, and extracting information from images.

Image processing is used in a variety of applications, including medical imaging, satellite imaging, digital photography, surveillance, robotics, and more. Common techniques used in image processing include image filtering, segmentation, edge detection, object recognition, and pattern recognition.

Here are the steps that will be required to process an image and extract the geometry from it.

- i. Preprocessing the image
- ii. Detecting edges
- iii. Hough Transform
- iv. Segmenting truss
- v. Truss Identification
- vi. Refining the truss
- vii. Visualization

5.1 PREPROCESSING THE IMAGE

The first step is to preprocess the image to improve its quality and remove any noise that might affect the accuracy of the truss detection. This can be achieved by applying image filters, such as a median or Gaussian filter, to smooth out the image and remove any unwanted artifacts.

Here gaussian blur by open CV library is used to blur the image, so that the jittery lines and noise can be removed.

```
# read the input image
img_path = "E:\WhatsApp Image 2023-04-20 at 12.18.26.jpg"
if not os.path.exists(img_path):
    print(f"Image file not found: {img_path}")
    exit()

img = cv2.imread(img_path)

if img is None:
    print(f"Failed to load image: {img_path}")
    exit()

gaussian_img=cv2.GaussianBlur(img,(3,3),0,borderType=cv2.BORDER_CONSTANT)
```

Fig 5.1 Loading image

5.2 DETECTING EDGES

To detect the edge contours canny edge detection is used provided in Open CV documentation. To use canny edge detection the blurred image is converted into a grayscale image and then canny edge function is used [1,3].

```
# convert the input image to grayscale image
gray = cv2.cvtColor(gaussian_img,cv2.COLOR_BGR2GRAY)

sigma= 0.3
median=np.median(gray)
# median=np.median(img)
lower=int(max(0,(1.0-sigma)*median))
upper=int(min(255,(1.0+sigma)*median))
edges=cv2.Canny(gray,lower,upper)
minLineLength = 10
maxLineGap = 5
```

Fig 5.2 Canny edge detection

5.3 HOUGH TRANSFORM

The Hough Transform is a feature extraction technique used in image processing and computer vision to detect simple shapes, such as lines, circles, and ellipses in an image. It was originally developed to detect lines in an image, but later extended to detect circles and other shapes.

The basic idea behind the Hough Transform is to convert the image from the x-y coordinate system to the Hough space, which is a parameter space that represents the parameters of the shapes to be detected. In the case of lines, the Hough space is represented by two parameters, the slope and the y-intercept of the line. Each point in the x-y plane corresponds to a curve in the Hough space. The intersection of these curves represents the parameters of the line that passes through the corresponding point in the x-y plane.

The Hough Transform algorithm involves the following steps:

Edge detection: Detect edges in the image using an edge detection algorithm, such as the Canny edge detector.

Parameter space: Define the parameter space based on the type of shape to be detected. For lines, the parameter space is a two-dimensional space, with one dimension representing the slope and the other representing the y-intercept.

Voting: For each edge point in the image, vote for all the possible curves in the parameter space that pass through that point.

Accumulation: Accumulate the votes in the parameter space to find the most probable curves.

Thresholding: Apply a threshold to the accumulated votes to select the most probable curves.

Extraction: Extract the lines or other shapes corresponding to the selected curves in the parameter space.

The Hough Transform is a powerful technique for detecting simple shapes in an image, but it has some limitations. It is sensitive to noise and may detect spurious shapes, and it is computationally expensive, especially for large images. However, it can be combined with other techniques, such as edge linking and filtering, to improve its performance.

```
# apply probabilistic Hough transform
lines = cv2.HoughLinesP(edges,1,np.pi/180,minLineLength,maxLineGap)
for line in lines:
    for x1,y1,x2,y2 in line:
        cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)
cv2.imshow("blur",gaussian_img)
cv2.imshow('houghlines.jpg',img)
cv2.imshow('edges', edges)
cv2.waitKey(0)
```

Fig 5.3 Hough transform

With Hough Transform only the borders and some edges can be detected not all the straight lines, which is why an alteration of Hough Transform called probabilistic Hough transform is used here to detect all the line shapes. It is an improvement over the standard Hough transform, as it is much faster and more efficient [4].

The basic idea behind the Hough probabilistic transform is to first detect edge pixels in the image using an edge detection algorithm such as Canny edge detector. Then, instead of considering all possible lines that pass through the image, the algorithm randomly selects a subset of edge pixels and tries to fit a line to them using the least-squares method. The result is a set of line segments that approximate the edges in the image.

5.4 SEGMENTING TRUSS:

Once the lines are detected image needs to be segmented into different colour spaces to do that there are many clustering algorithms available like hierarchical clustering and k-means clustering. Here k means clustering is used [2].

K means clustering:

K-means clustering is a popular unsupervised machine learning algorithm used to group similar data points together. The algorithm works by partitioning a dataset into k clusters, where k is a pre-specified number of clusters. The goal is to minimize the sum of squared distances between each data point and its assigned cluster center, also known as the "centroid".

The algorithm works by first randomly selecting k data points to serve as the initial centroids. Each data point is then assigned to the nearest centroid based on the Euclidean distance between the point and the centroid. After all the data points have been assigned to

clusters, the centroids are updated to be the mean of all the data points in their respective clusters. This process of assigning data points to clusters and updating centroids is repeated iteratively until convergence is reached, which occurs when the assignments of the data points to clusters no longer change.

K-means clustering can be used for a variety of applications, such as image segmentation, customer segmentation, and anomaly detection. However, the algorithm has some limitations. It requires the number of clusters k to be specified in advance, and it is sensitive to the initial placement of the centroids, which can result in different clustering. Additionally, it assumes that the data points are normally distributed and have equal variances, which may not always be true in real-world scenarios.

Here sci-kit learn library is used to do k mean clustering. Matplotlib is used to show the result in graph points with different colour values.

5.4.1 Flattening Each Channel Of Image

```
# Flatten Each channel of the Image
all_pixels = img.reshape((-1,3))
print(all_pixels.shape)

dominant_colors = 10

km = KMeans(n_clusters=dominant_colors)
km.fit(all_pixels)
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10,
       random_state=None, tol=0.0001, verbose=0)
centers = km.cluster_centers_
print(centers) # In RGB Format
```

Fig 5.4.1 Flattening

First, the image is flattened so that each pixel is represented as a row of three values (red, green, and blue values). The “KMeans” class from the “sklearn.cluster” module is then used to cluster these pixels into a given number of clusters, specified by the “n_clusters” parameter. In this case, the number of clusters is set to “dominant_colors”. The centroids of these clusters are then extracted and stored in the “centers” variable. This array contains the RGB values of the dominant colors in the image.

5.4.2 Storing Info In Color Array

First the centroid is converted into integer values from the RGB colour values to plot further using “matplotlib”

```
# Convert to Integer format
centers = np.array(centers,dtype='uint8')
print(centers)
i = 1

plt.figure(0,figsize=(8,2))
```

Fig 5.4.2 Storing

Next, the code plots each dominant color as a color swatch using Matplotlib and stores the RGB values of each dominant color in the colors array.

```
# Storing info in color array
colors = []

for each_col in centers:
    plt.subplot(1,dominant_colors,i)
    plt.axis("off")
    i+=1

    colors.append(each_col)

# Color Swatch
a = np.zeros((100,100,3),dtype='uint8')
a[:, :, :] = each_col
plt.imshow(a)

plt.show()
```

Fig 5.4.3 Color array

5.4.3 Creating New Clustered Image

A new image is created by assigning each pixel in the original image to its closest dominant color based on the cluster labels assigned by K-means. This is done by iterating over each pixel in the flattened image, and assigning it the RGB value of its closest dominant color from the colors array. The resulting image is then reshaped to its original shape and displayed using Matplotlib. After that it is saved at the desired location is “os” module.

```

# Segmenting our original image
new_img = np.zeros((img.shape[0]*img.shape[1],3),dtype='uint8')
print(new_img.shape)
colors
km.labels_
# Iterate over the image
for ix in range(new_img.shape[0]):
    new_img[ix] = colors[km.labels_[ix]]

new_img = new_img.reshape((original_shape))
plt.imshow(new_img)
plt.show()
if os.path.isdir("D:/ransac images"):
    breakpoint
else:
    os.mkdir("D:/ransac images")
cv2.imwrite('D:/ransac images/image-ransac.png',new_img)

```

Fig 5.4.4 Segmenting

5.5 TRUSS IDENTIFICATION

For truss identification RANSAC algorithm is used to convert the data points of clustered image into the elements then identify the nodes and elements using connectivity analysis

RANSAC uses many libraries such as, “numpy”, ”matplotlib”, and ”skimage”

This program performs line extraction from an image using RANSAC algorithm. The program reads an input image file and applies RANSAC algorithm to detect lines from black pixels in the image. It returns the inliers detected by the RANSAC algorithm along with the LineModelND that represents the detected line.

The program consists of several helper functions.

The “read_black_pixels” function reads the input image file and returns a numpy array with the shape (N,2) which represents the black pixels in the image.

```
def read_black_pixels(imagefilename:str):
    #returns a numpy array with shape (N,2) N points, x=[0], y=[1]
    #The coordinate system is Cartesian
    np_image=io.imread(imagefilename,as_gray=True)
    black_white_threshold=0
    if (np_image.dtype == 'float'):
        black_white_threshold=0.5
    elif (np_image.dtype == 'uint8'):
        black_white_threshold=128
    else:
        raise Exception("Invalid dtype %s " % (np_image.dtype))
    indices=np.where(np_image <= black_white_threshold)
    width=np_image.shape[1]
    height=np_image.shape[0]
    cartesian_y=height-indices[0]-1
    np_data_points=np.column_stack((indices[1],cartesian_y))
    return np_data_points, width,height
```

Fig 5.5.1 Function call read black pixel

The `extract_first_ransac_line` function accepts a numpy array with shape (N,2) N points, with coordinates x=[0],y=[1]. It applies the RANSAC algorithm to detect a line and returns the inliers of the just discovered RANSAC line, all data points with the inliers removed, and the model line.

```
def extract_first_ransac_line(data_points:[], max_distance:int):
    """
    Accepts a numpy array with shape N,2 N points, with coordinates x=[0],y=[1]
    Returns
    A numpy array with shape (N,2), these are the inliers of the just discovered ransac line
    All data points with the inliers removed
    The model line
    """

    model_robust, inliers = ransac(data_points, LineModelND, min_samples=min_samples,
                                   residual_threshold=max_distance, max_trials=1000)
    results_inliers=[]
    results_inliers_removed=[]
    for i in range(0,len(data_points)):
        if (inliers[i] == False):
            #Not an inlier
            results_inliers_removed.append(data_points[i])
            continue
        x=data_points[i][0]
        y=data_points[i][1]
        results_inliers.append((x,y))
    return np.array(results_inliers), np.array(results_inliers_removed),model_robust
```

Fig 5.5.2 Function call extract first ransac line

The `generate_plottable_points_along_line` function computes points along the specified line model. It takes the model, `xmin`, `xmax`, `ymin`, and `ymax` as input parameters and returns a numpy array of the shape `[[x1,y1],[x2,y2]]`.

```
def generate_plottable_points_along_line(model:LineModelND, xmin:int,xmax:int, ymin:int, ymax:int):
    """
    Computes points along the specified line model
    The visual range is
    between xmin and xmax along X axis
    and
    between ymin and ymax along Y axis
    return shape is [[x1,y1],[x2,y2]]
    """
    unit_vector=model.params[1]
    slope=abs(unit_vector[1]/unit_vector[0])
    x_values=None
    y_values=None
    if (slope > 1):
        y_values=np.arange(ymin, ymax,1)
        x_values=model.predict_x(y_values)
    else:
        x_values=np.arange(xmin, xmax,1)
        y_values=model.predict_y(x_values)

    np_data_points=np.column_stack((x_values,y_values))
    return np_data_points
```

Fig 5.5.3 Function call generate plottable

The `superimpose_all_inliers` function creates an RGB image array with dimension `heightXwidth` and draws the points with various colours. It takes the inliers of all RANSAC lines and the width and height of the image as input parameters and returns the RGB image array.

```
def superimpose_all_inliers(ransac_lines,width:float, height:float):
    #Create an RGB image array with dimension heightXwidth
    #Draw the points with various colours
    #return the array

    new_image=np.full([height,width,3],255,dtype='int')
    colors=[(0,255,0),(255,255,0),(0,0,255)]
    for line_index in range(0,len(ransac_lines)):
        color=colors[line_index % len(colors)]
        ransac_lineinfo=RansacLineInfo=ransac_lines[line_index]
        inliers=ransac_lineinfo.inliers
        y_min=inliers[:,1].min()
        y_max=inliers[:,1].max()
        x_min=inliers[:,0].min()
        x_max=inliers[:,0].max()
        plottable_points=generate_plottable_points_along_line(ransac_lineinfo.model, xmin=x_min,xmax=x_max, ymin=y_min,ymax=y_max)
        for point in plottable_points:
            x=int(round(point[0]))
            if (x >= width) or (x < 0):
                continue
            y=int(round(point[1]))
            if (y >= height) or (y < 0):
                continue
            new_y=height-y-1
            new_image[new_y][x][0]=color[0]
            new_image[new_y][x][1]=color[1]
            new_image[new_y][x][2]=color[2]
    return new_image
```

Fig 5.5.4 Function call superimpose all linerers

The `extract_multiple_lines_and_save` function accepts the input file name, the number of iterations, the maximum distance, and the minimum inliers allowed as input parameters. It uses the above helper functions to extract multiple lines from the input image and saves the output images. The function applies the RANSAC algorithm to detect multiple lines and iteratively applies RANSAC algorithm to detect lines until the minimum number of inliers is reached. The function superimposes all the inliers detected by the RANSAC algorithm on the input image and saves the output image.

```
def extract_multiple_lines_and_save(inputfilename:str, iterations:int, max_distance:int, min_inliers_allowed:int):
    """
    min_inliers_allowed - a line is selected only if it has more than this inliers. The search process is halted when this condition is met
    max_distance - This is the RANSAC threshold distance from a line for a point to be classified as inlier
    """
    print("-----")
    print("Processing: %s" % (inputfilename))
    folder_script=os.path.dirname(__file__)
    absolute_path=os.path.join(folder_script,"images/",inputfilename)

    results:List[RansacLineInfo]=[]
    all_black_points,width,height=read_black_pixels(absolute_path)
    print("Found %d pixels in the file %s" % (len(all_black_points),inputfilename))
    starting_points=all_black_points
    for index in range(0,iterations):
        if (len(starting_points) <= min_samples):
            print("No more points available. Terminating search for RANSAC")
            break
        inlier_points,inliers_removed_from_starting,model=extract_first_ransac_line(starting_points,max_distance=max_distance)
        if (len(inlier_points) < min_inliers_allowed):
            print("Not sufficeint inliers found %d , threshold=%d, therefore halting" % (len(inlier_points),min_inliers_allowed))
            break
        starting_points=inliers_removed_from_starting
        results.append(RansacLineInfo(inlier_points,model))
        print("Found %d RANSAC lines" % (len(results)))
    superimposed_image=superimpose_all_inliers(results,width,height)
    #Save the results
    io.imshow("D:\ransac_images",superimposed_image)
```

Fig 5.5.5 Function call extract multiple line and saves

6. OUTPUT OF CODES

6.1. OUTPUT OF 2D TRUSS ANALYSIS PYTHON CODE:

Solving this problem using the code [6].

A four bar truss is loaded as shown in Fig. P. 9.4.8(a). Assuming that for each element the cross-sectional area is 400 mm^2 and modulus of elasticity is 200 GPa , determine the deflection, reaction forces and stresses in each element.

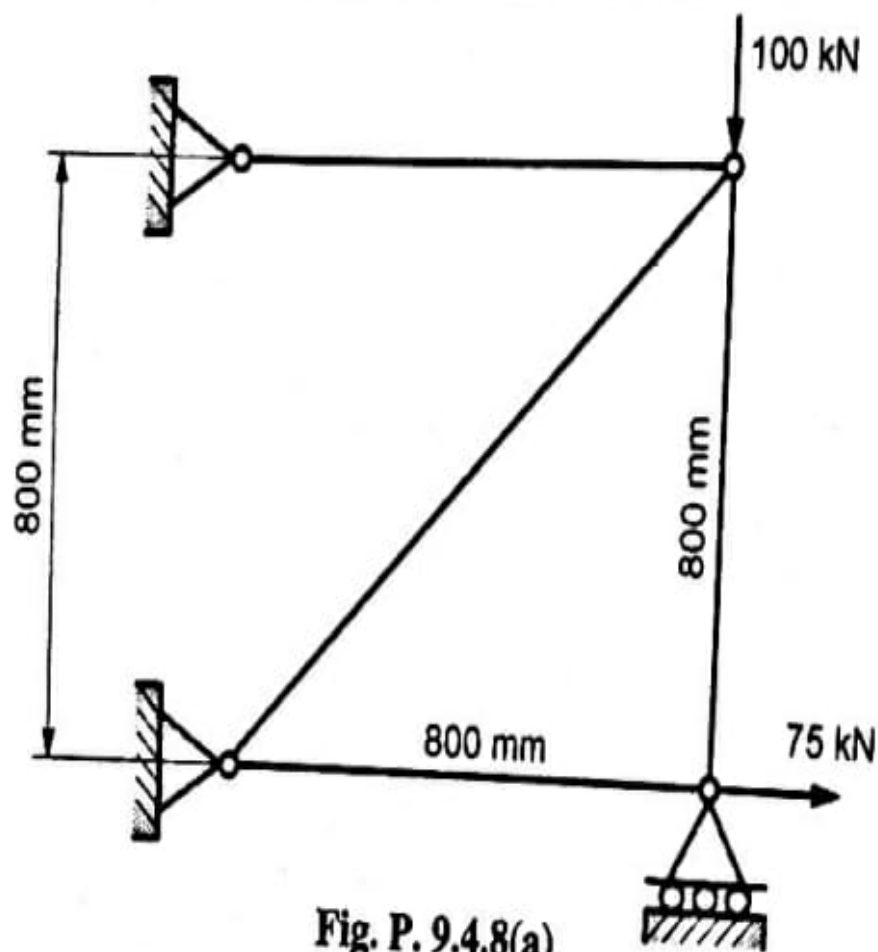


Fig 6.1.1 sum

6.1.1 Entering Total Number Of Nodes And Elements:

```
Enter the total number of nodes : 4  
Enter the total number of Elements : 4
```

Fig 6.1.2 Nodes and elements

6.1.2. Entering Node Coordinates:

```
Enter the x co-ordinate of node 1 in mm : 0  
Enter the y co-ordinate of node 1 in mm : 0  
Enter the x co-ordinate of node 2 in mm : 800  
Enter the y co-ordinate of node 2 in mm : 0  
Enter the x co-ordinate of node 3 in mm : 800  
Enter the y co-ordinate of node 3 in mm : 800  
Enter the x co-ordinate of node 4 in mm : 0  
Enter the y co-ordinate of node 4 in mm : 800
```

Fig 6.1.3 coordinates

6.1.3. Entering Cross Section Area and Modulus Of Elasticity:

```
Enter the Area of cross section in mm2: 400  
Enter the Modulus of Elasticity in N/mm2 : 200000
```

Fig 6.1.4 Modulus of elasticity and cross section area

6.1.4. Entering Start And End Nodes Of Elements:

```
Enter the Start node of element 1 : 1  
Enter the End node of element 1 : 2  
Enter the Start node of element 2 : 2  
Enter the End node of element 2 : 3  
Enter the Start node of element 3 : 1  
Enter the End node of element 3 : 3  
Enter the Start node of element 4 : 4  
Enter the End node of element 4 : 3
```

Fig 6.1.5 Node connectivity

6.1.5. Global Stiffness Matrix:

```
Global Stiffness Matrix of the Truss

[[ 135355.339  35355.339 -100000.    0.   -35355.339 -35355.339
    0.         0.         ]
 [  35355.339  35355.339    0.    0.   -35355.339 -35355.339
    0.         0.         ]
 [-100000.    0.   100000.    0.    0.    0.
    0.         0.         ]
 [    0.         0.    0.  100000.    0.  -100000.
    0.         0.         ]
 [ -35355.339 -35355.339    0.    0.  135355.339  35355.339
 -100000.    0.         ]
 [ -35355.339 -35355.339    0.  -100000.  35355.339 135355.339
    0.         0.         ]
 [    0.         0.    0.    0.  -100000.    0.
 100000.    0.         ]
 [    0.         0.    0.    0.    0.    0.
    0.         0.         ]]
```

Fig 6.1.6 Global stiffness matrix

6.1.6. Entering Boundary Conditions:

Loads

```
-----Loading-----

Enter the total number of loaded nodes : 2

Enter the node number of Loading : 2
Enter the Horizontal load at this node in N : 75000
Enter the Vertical load at this node in N : 0

Enter the node number of Loading : 3
Enter the Horizontal load at this node in N : 0
Enter the Vertical load at this node in N : -100000
```

Fig 6.1.6 Loads

Supports:

```

-----Support Specifications-----
Enter the total number of nodes having supports : 3

Enter the node number of support : 1
P = pinned
H = Horizontal restrained (vertical is free to move)
V = Vertical restrained (Horizontal is free to move)

Enter the condition of the support : p

Enter the node number of support : 2
P = pinned
H = Horizontal restrained (vertical is free to move)
V = Vertical restrained (Horizontal is free to move)

Enter the condition of the support : v

Enter the node number of support : 4
P = pinned
H = Horizontal restrained (vertical is free to move)
V = Vertical restrained (Horizontal is free to move)

Enter the condition of the support : p

```

Fig 6.1.7 supports

6.1.7. Answers:

```

Displacement matrix of nodes

[[ 0.    ]
 [ 0.    ]
 [ 0.75  ]
 [ 0.    ]
 [ 0.207 ]
 [-0.793]
 [ 0.    ]
 [ 0.    ]]

```

Fig 6.1.8 Displacement matrix

```
Strain in the elements
[[ 0.001]
 [-0.001]
 [-0.    ]
 [ 0.    ]]
```

Fig 6.1.9 Strain matrix

```
Stress in the elements
[[ 187.5  ]
 [-198.177]
 [-73.184]
 [ 51.875]]
```

Fig 6.1.10 Stress matrix

```
Force matrix of nodes
```

```
[[ -54289.322]
 [ 20710.678]
 [ 75000.    ]
 [ 79289.322]
 [ -0.      ]
 [-100000.   ]
 [-20710.678]
 [ 0.       ]]
```

```
***Positive is Tensile
Negative is Compressive***
```

Fig 6.1.11 Force matrix

```
Force in the element  
[[ 75000. ]  
 [-79270.883]  
 [-29273.691]  
 [ 20749.96 ]]
```

Fig 6.1.12 Force in element

6.2. IMAGE PROCESSING OUTPUT:

6.2.1. This Image Is Given As Input:



Fig 6.2.1 Main input image

6.2.2 Gaussian Blur:



Fig 6.2.2 Gaussian blur

6.2.3 Canny Edge Detection:

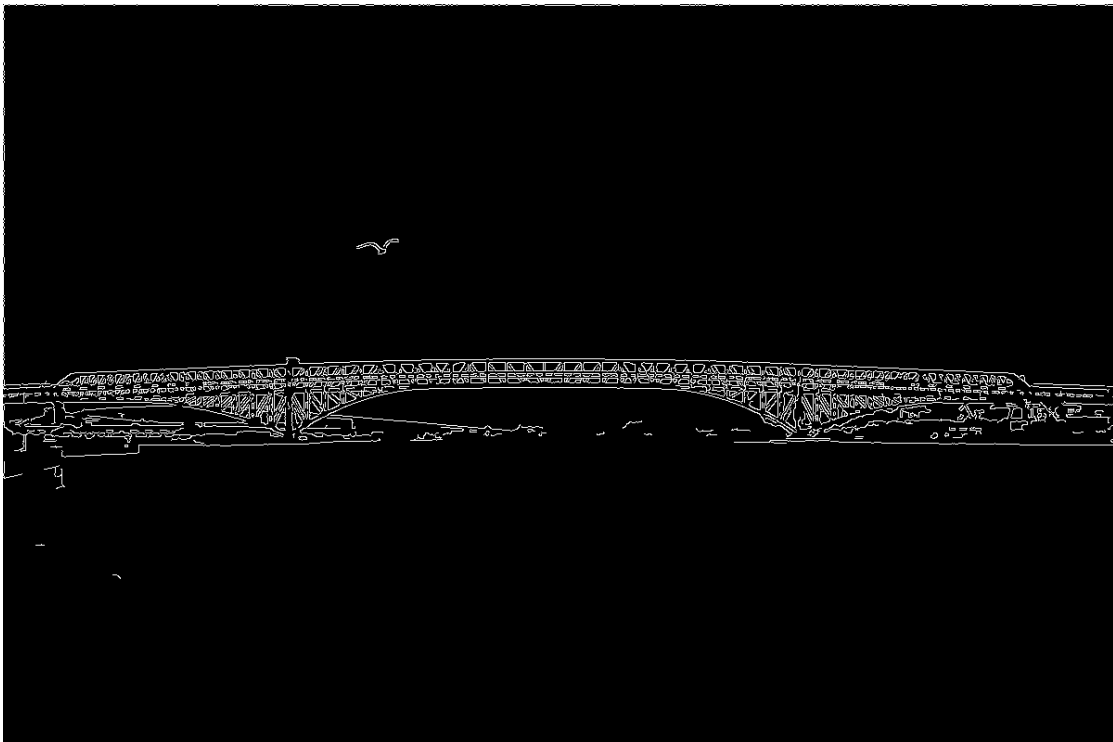


Fig 6.2.3 Canny edge detection

6.2.3. Probabilistic Hough Transform:



Fig 6.2.4 Probabilistic Hough

6.2.5 Dominant Colors:



Fig 6.2.5 Dominant colors

6.2.6 K Cluster Image:

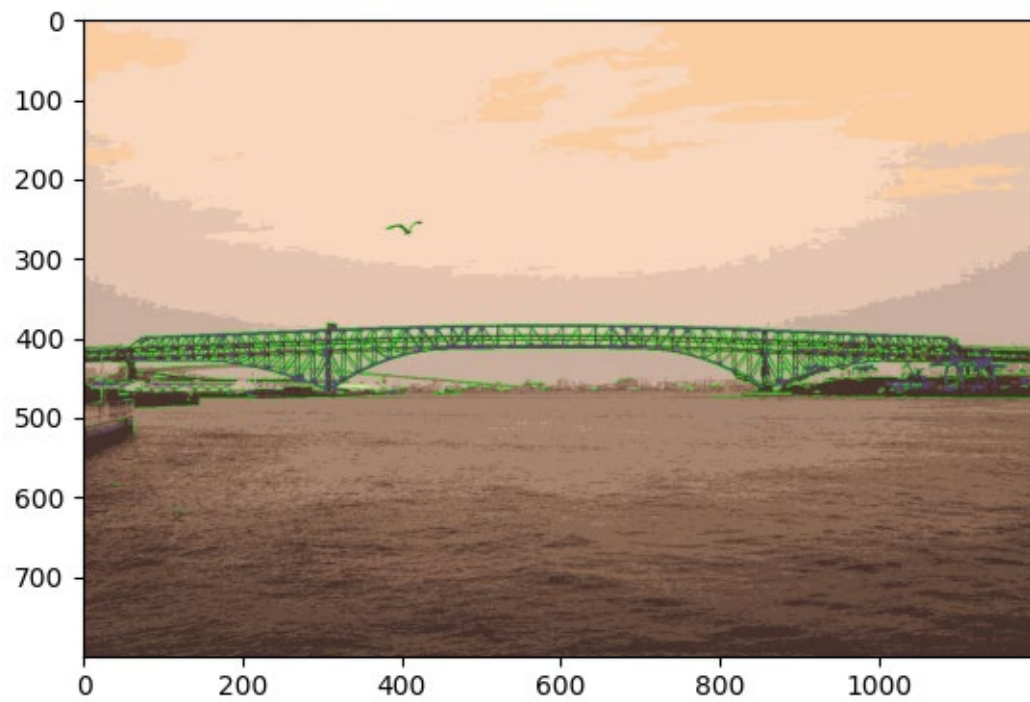


Fig 6.2.6 K-cluster

7. WEBSITE

We have created Python primary code. However, all users need support or a special debugger for running the code; that is why, we created an exemplary user interface for consumers who can easily enter the data of truss and get answers. We assume some restrictions, such as one cross-section area for the entire structure.

First and foremost, the user has to enter the node number and element number. The number should be logically correct then only the user goes to the next step [7].

Step 1

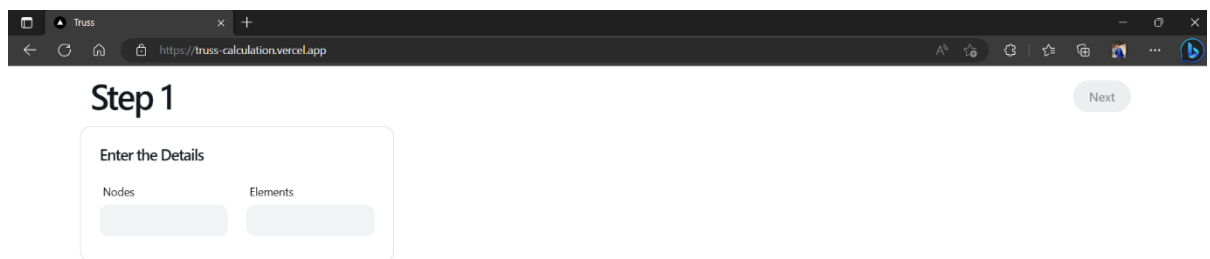


Fig 7.1 Enter node and element.

Step 2

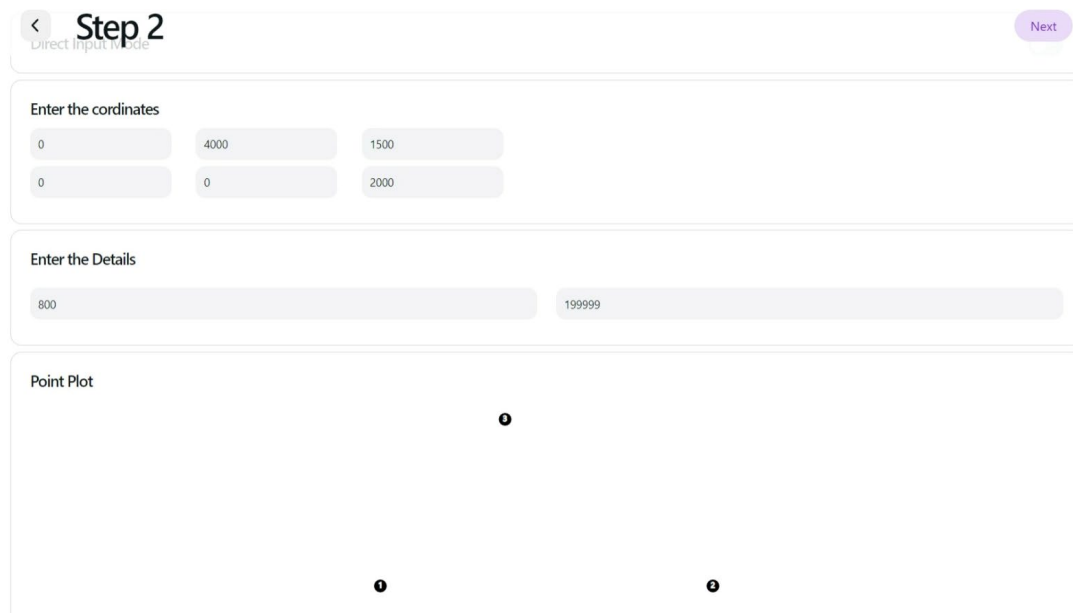
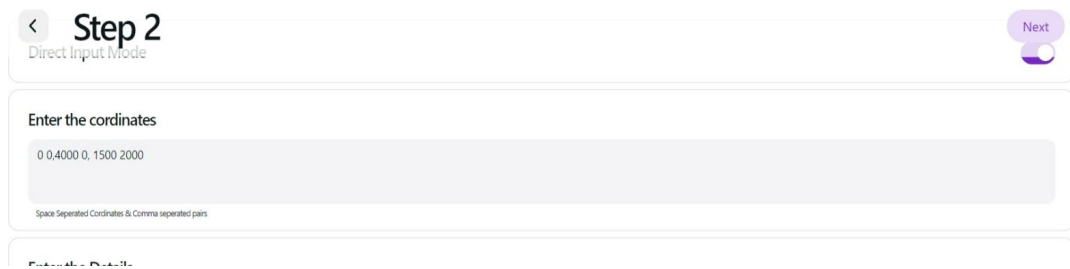


Fig 7.2 Coordinates



< **Step 2** Next

Direct Input Mode

Enter the coordinates

0 0,4000 0, 1500 2000

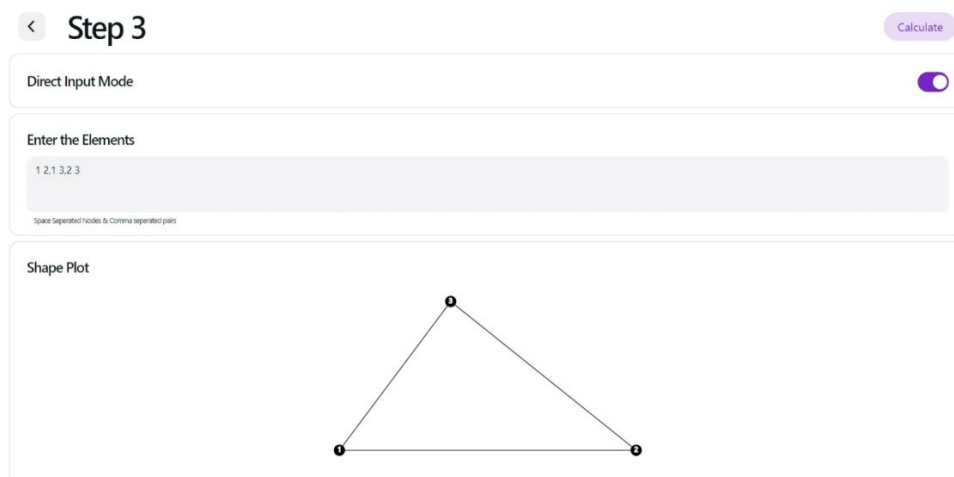
Space Separated Coordinates & Comma separated pairs

Fig 7.3 other view for coordinates

To initiate the truss analysis, the user must provide all coordinates of nodes, which can be seen point plot, as well as the area, which must enter in mm^2 , and the modulus of elasticity, which must enter in N/mm^2 . Users have two views for entering the coordinate system.

Step-3

After the second step, users have to provide element connectivity starting point of starting element and ending point. In this step, the user can enter either data in tabular form or with space and a comma. Users can see the results at the point plot when they provide connectivity. Hence, they can verify and edit too.



< **Step 3** Calculate

Direct Input Mode

Enter the Elements

1 2,1 3,2 3

Space Separated Nodes & Comma separated pairs

Shape Plot

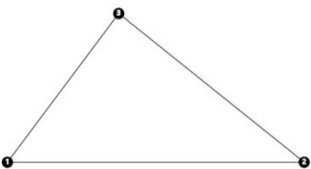


Fig 7.4 Element connectivity

< **Step 3** Calculate

Direct Input Mode ☐

Enter the Elements

| | | |
|---|---|---|
| 1 | 1 | 3 |
| 2 | 3 | 3 |

Fig 7.5 Other view for element connectivity

Step 4

If users want only the global stiffness matrix, they can see it in this step; however, if users require further calculation, they must provide support details, meaning nodes are either fixed, vertically restrained, horizontally restrained, or any force is applied.

< **Step 4** Next

Support Specifications

Number of nodes having supports: Number of nodes having loads:

Global Stiffness Matrix

| | | | | | |
|--------|---------------------|---------------------|---------------------|---------------------|---------------------|
| 63040 | 30720 | -40000 | 0 | -23040 | -30720 |
| 30720 | -40960.000000000001 | 0 | 0 | -30720 | -40960.000000000001 |
| -40000 | 0 | 70472.9291489963 | -24378.343319197047 | -30472.929148996314 | 24378.343319197047 |
| 0 | 0 | -24378.343319197047 | 19502.674655357638 | 24378.343319197047 | -19502.674655357638 |
| -23040 | -30720 | -30472.929148996314 | 24378.343319197047 | 53512.929148996314 | 6341.656680802953 |
| -30720 | -40960.000000000001 | 24378.343319197047 | -19502.674655357638 | 6341.656680802953 | 60462.674655357645 |

Shape Plot

Fig 7.6 Support specification

Step 5

which nodes are fixed and on which node having support enter in the next step.

Step 5

[Next](#)

Enter Supported Nodes and Select Its Type

1

2

P ▾

V ▾

Enter Loaded Nodes and Its Force

1

 F_H 8000 F_V -12000

Shape Plot

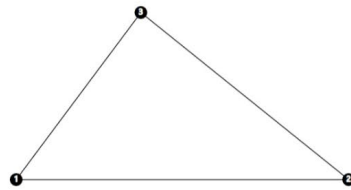


Fig 7.7 Enter supports

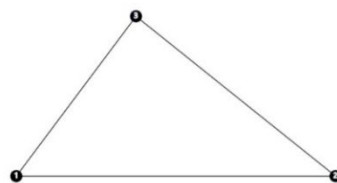
Step 6

Results

Global Stiffness Matrix of the Truss

| | | | | | |
|--------|--------------------|---------------------|---------------------|---------------------|---------------------|
| 63040 | 30720 | -40000 | 0 | -23040 | -30720 |
| 30720 | 40960.00000000001 | 0 | 0 | -30720 | -40960.00000000001 |
| -40000 | 0 | 70472.9291489963 | -24378.343319197047 | -30472.929148996314 | 24378.343319197047 |
| 0 | 0 | -24378.343319197047 | 19502.674655357638 | 24378.343319197047 | -19502.674655357638 |
| -23040 | -30720 | -30472.929148996314 | 24378.343319197047 | 53512.929148996314 | 6341.65668002953 |
| -30720 | -40960.00000000001 | 24378.343319197047 | -19502.674655357638 | 6341.65668002953 | 60462.674655357645 |

Shape Plot



| | |
|------------------------------|--|
| Displacement matrix of nodes | |
| 0 | |
| 0 | |
| 0.265625 | |
| 0 | |
| 0.3412098434321629 | |
| -0.34135660132412284 | |

| | |
|-------------------------|--|
| Force matrix of nodes | |
| -8000.000000000004 | |
| 3499.9999999999964 | |
| -1.8189894035458565e-12 | |
| 8500 | |
| 8000 | |
| -11999.999999999995 | |

| | |
|--------------------------|--|
| Strain in the Elements | |
| 0.00006640625 | |
| -0.000027325487462258025 | |
| -0.00008503914680668805 | |

| | |
|------------------------|--|
| Stress in the elements | |
| 13.281250000000002 | |
| -5.465097492451605 | |
| -17.00782936133761 | |

| | |
|-----------------------|--|
| Force in the elements | |
| 10625.000000000002 | |
| -4372.077993961284 | |
| -13606.263489070088 | |

Fig 7.8 Final results

Clients can find out all results Global stiffness matrix , Displacement matrix , force matrix , strain in the element and stress in the element.

In future, the same kind of user interface will also be available for Image processing. This is why users can use this platform on any device without any computational software or a smart device.

The link of the website is here : <https://www.gecr.engineer/>

8. FUTURE DEVELOPMENTS

This research's ultimate goal is to create a software that can directly extract geometry from an actual real-life structure using machine learning and image processing and then to use that 2d geometry to smoothly do finite element analysis, moreover this can be implemented in various fields such as prototype creation by comparing the results of the previous analysis and using the trained machine learning model to generate an idea geometry for any given value of load.

Furthermore, by perspective scaling and some image restrictions and machine learning model can be trained and created which can give a 3D model of the object from the image then this model can be used to perform using other FEA software or laymen can use the inbuilt basic analysis program to give the boundary conditions to get the required results.

Current model is capable of detecting the elements with high accuracy but with some optimization to remove background more accurate geometry can be extracted.

With further development all the parts of FEA can be surrogated by machine learning in similar fashion

9. CONCLUSION

The project was carried out with the goal of designing and building a system capable of extracting geometry from an image using machine learning and then do the FEA analysis of that geometry with given boundary conditions.

The designed system has demonstrated to be capable of detecting geometry contours with high accuracy, hence easily isolating the geometry from rest of the image, owing to Hough probabilistic transform.

After experimenting with many values of gaussian blur matrix extra detected objects can be optimized and more precise geometry can be generated with absence of unnecessary objects.

k-clustering, unsupervised machine learning model can effectively classify the color segments and then generate data points which is further used in RASAC algorithm to create the computer generated geometry of the object.

Once a program successfully generates the geometry of the object it can be used in the code created to perform the finite element analysis.

All the libraries used are open-source libraries. And for future development hand generated dataset needs to be created, labeled and used in training of the machine learning model to more accurately detect the geometry and relative length of the components.

With our current research the FEA of the geometry can be done, however computer-generated geometry from image geometry needs further coding. Right now, the detected geometry can be seen on the input image accurately.

10. REFERENCE

- [1]. M. Kamari and O. Güneş, "Segmentation and Analysis of a Sketched Truss Frame Using Morphological Image Processing Techniques," pp. 3-4, Mar. 2023. Available: https://www.researchgate.net/publication/305398093_Segmentation_and_Analysis_of_a_Sketched_Truss_Frame_Using_Morphological_Image_Processing_Techniques . [Accessed: May 7, 2023].
- [2]. M. Kamari and O. Güneş, "Segmentation and Analysis of a Sketched Truss Frame Using Morphological Image Processing Techniques," pp. 8, Mar. 2023. Available: https://www.researchgate.net/publication/305398093_Segmentation_and_Analysis_of_a_Sketched_Truss_Frame_Using_Morphological_Image_Processing_Techniques . [Accessed: May 7, 2023].
- [3]. OpenCV. "Canny Edge Detection," OpenCV Documentation, [Online]. Available: https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html. [Accessed: May 7, 2023].
- [4]. OpenCV. "Hough Line Transform," OpenCV Documentation, [Online]. Available: https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html. [Accessed: May 7, 2023].
- [5]. "sklearn.cluster.KMeans," scikit-learn Documentation, [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>.
- [6]. R. B. Patil, "Computer-aided design", pp. 9-23, 2018. Available: <https://techknowledgebooks.com/product/computer-aided-design/>. [Accessed: May 7, 2023].
- [7]. Dr. Vijaykumar S Jatti and Prof. Mandar Sapre, "Finite Element Methods", pp. 3-8 ,2021. Available: <https://www.techneobooks.in/product/sppu/semester-7/finite-element-methods-> . [Accessed: May 7, 2023].