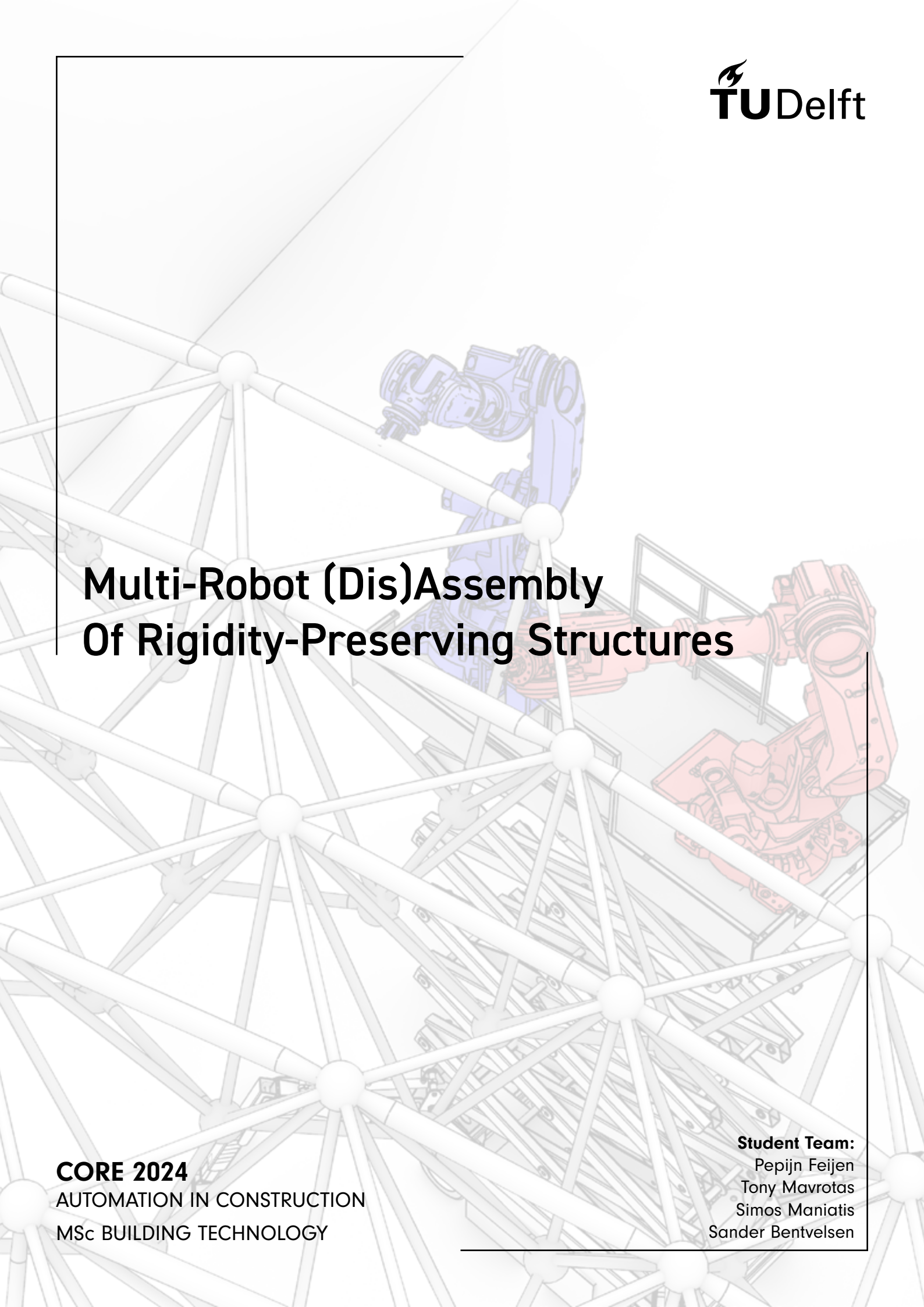
TU Delft

Multi-Robot (Dis)Assembly Of Rigidity-Preserving Structures



CORE 2024
AUTOMATION IN CONSTRUCTION
MSc BUILDING TECHNOLOGY

Student Team:
Pepijn Feijen
Tony Mavrotas
Simos Maniatis
Sander Bentvelsen

Student Team: Pepijn Feijen / Tony Mavrotas 6047807 / Sander Bentvelsen 4851579 / Simos Maniatis 5916046

I would like to express my gratitude to the teaching team of CORE 2024 for the support and guidance provided throughout this journey. Your insights and encouragement have been instrumental in navigating the complexities of this project. From conceptual exploration to technical problem-solving, your expertise has not only enriched our understanding but also inspired us to push the boundaries of innovation in robotic construction.

For future students interested in utilizing the UR5 and COMAU robots in tandem, we welcome them to reach out to us for any support or insights. Our contact emails are listed below for any inquiries or further collaboration opportunities.

At the end of this document, you will find a QR code that links to our GitHub repository, providing full documentation and access to a video showcasing our project. We hope this resource serves as a useful reference for upcoming students and future advancements.

Thank you once again for your dedication to fostering a collaborative and inspiring learning environment.

Tony Mavrotas : a.mavrotas@student.tudelft.nl
Sander Bentvelsen: s.a.bentvelsen@student.tudelft.nl
Pepijn Feijen: p.feijen@student.tudelft.nl
Simos Maniatis:S.maniatis@student.tudelft.nl

GITHUB

VIDEO

Contents

A1. Abstract

This project explores a multi-robot (dis)assembly system for building and dismantling rigidity-preserving space frames, aiming to transform construction through automation. Traditional methods often face high labor costs, long timelines, material waste, and environmental issues, especially with extensive scaffolding. Our system uses collaborative robots to assemble structural components in a sequence that maintains rigidity, minimizing the need for scaffolding and promoting efficiency and sustainability.

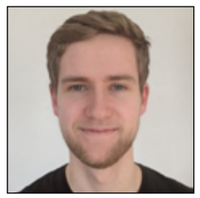
Each robot executes specific tasks using precision-engineered, modular components designed for easy assembly and disassembly. The node connections are optimized for structural efficiency and flexibility, featuring adjustable parts like hexagonal sleeves and pins for precise alignment. A computational model guides the assembly process to ensure stability by maintaining at least three connections per node and strategically placing components to prevent structural weaknesses.

Prototyping with scaled models validated the system's ability to maintain rigidity and the accuracy of robotic placement. The design emphasizes sustainability by maximizing material efficiency and reusability, aligning with circular economy principles. Reducing scaffolding lowers material costs and the carbon footprint, supporting adaptable and eco-friendly construction.

In summary, this multi-robot assembly framework advances construction automation by ensuring structural integrity through a carefully planned assembly sequence and modular components. It offers a scalable,efficient, and sustainable approach, setting the stage for adaptable architecture in modern urban development.

A2.Team Structure


Tony Mavrotas


Pepijn Feijen


Sander Bentvelsen


Simos Maniatis

From the very beginning, our team was passionate about transforming how construction is done. We wanted to tackle the huge labor costs, long building times, and the environmental issues that come with traditional methods. We dreamed of using robots to make building faster, smarter, and greener. This led us to tackle the possibility of developing a system where multiple robots work together to put together and take apart sturdy space frames without the need for bulky scaffolding. Our goal was to create a more efficient and sustainable way to build, one that could easily adapt and reuse materials, ultimately setting a new standard for modern construction.

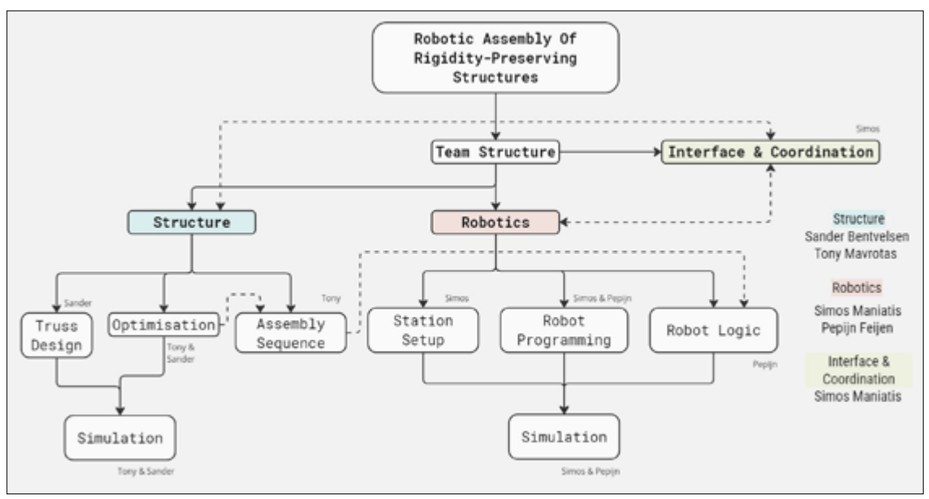


Figure 1: Team Structure Diagram (authors).


Figure 2: The construction of the roof of the Great Court. (https://www.britishmuseum.org/blog/everything-you-ever-wanted-know-about-great-court)


Figure 3: Robots working together making the process more efficient. (Bruun, Besler, Adriaenssens, & Parascho, 2022, p.1).

A3. Problem Analysis

Modern construction methods often grapple with **time-consuming** assembly processes, **excessive** material usage, and **high labor** demands, particularly when it comes to scaffolding and assembling large structures. These inefficiencies drive up costs and extend project timelines, hindering the overall productivity and scalability of the construction industry. Addressing these challenges is crucial, especially as urbanization accelerates and the demand for sustainable infrastructure grows. There is an urgent need to streamline construction practices to reduce material waste, lower labor expenses, and enhance safety and sustainability in building projects.

Challenges in Modern Construction

1.Time-Consuming Assembly Processes:
Traditional construction methods rely heavily on manual labor and sequential assembly steps, which inherently take longer to complete. For instance, erecting scaffolding is a labor-intensive process that requires careful planning and execution, often leading to delays in project timelines. These time-consuming processes not only delay project completion but also increase the likelihood of cost overruns and resource misallocation.

2. Excessive Material Usage:
The conventional approach to construction often results in significant material waste. Inefficient use of resources, such as excess steel, concrete, and other building materials, not only inflates project costs but also has detrimental environmental impacts. This waste is exacerbated by the need for redundant scaffolding and support structures, which consume additional materials without contributing to the final building's functionality.

3.High Labor Demands:
Construction projects typically require a large workforce to handle various tasks, from manual assembly to operating heavy machinery. This high demand for labor not only inflates project costs due to wages and benefits but also makes projects vulnerable to labor shortages and fluctuations in workforce availability. Moreover, reliance on manual labor increases the risk of human error, which can compromise the quality and safety of the constructed structures.

4. Safety Concerns:
Traditional scaffolding and manual assembly processes pose significant safety risks to workers. Falls, equipment malfunctions, and other accidents are common hazards in construction sites, leading to injuries, fatalities, and increased insurance costs. Ensuring worker safety is paramount, yet current methods often fall short in mitigating these risks effectively.

Figure 4: Construstion workers on-site during dangerous tasks. (https://www.flyability.com/blog/scaffolding)

Figure 5: Excessive material use for modern construction applications.

The Need for Transformation
As urbanization accelerates, cities are expanding rapidly, necessitating the construction of more buildings and infrastructure at a pace that traditional methods cannot sustain.

Integration of Robotic Technology
This project seeks to transform construction assembly by integrating robotic technology, specifically through the use of co-working robots. These robots are designed to work collaboratively, taking over tasks traditionally reliant on extensive scaffolding. By eliminating scaffolding, the project aims to cut down both the time and materials required for assembly while also reducing the labor intensity associated with conventional methods.

1.Reduction of Time and Materials:
Robotic systems can operate continuously without the need for breaks, significantly accelerating the assembly process. Moreover, robots can optimize material usage by precisely handling components, thereby minimizing waste. The elimination of scaffolding not only reduces the physical materials required but also streamlines the workflow, as robots can directly access construction areas without the need for intermediary support structures.

2. Lowering Labor Expenses:
By automating repetitive and labor-intensive tasks, the reliance on a large workforce diminishes. This shift not only reduces labor costs but also reallocates human workers to more skilled and supervisory roles, enhancing overall productivity. Furthermore, robotic systems can mitigate the impact of labor shortages, ensuring that projects remain on schedule despite fluctuations in workforce availability.

3. Enhancing Precision and Reliability:
Robotic assembly offers unparalleled precision, ensuring that components are placed accurately and consistently. This precision reduces the likelihood of structural defects and enhances the overall quality of the constructed buildings. Additionally, robots can perform tasks with a high degree of reliability, minimizing errors that can lead to costly rework and delays.

4. Improving Safety and Sustainability:
By taking over hazardous tasks, such as working at heights or handling heavy materials, robots significantly improve on-site safety. This reduction in human exposure to dangerous environments decreases the risk of accidents and injuries. Moreover, the optimized use of materials and the reduction of waste contribute to more sustainable construction practices, aligning with global environmental goals.

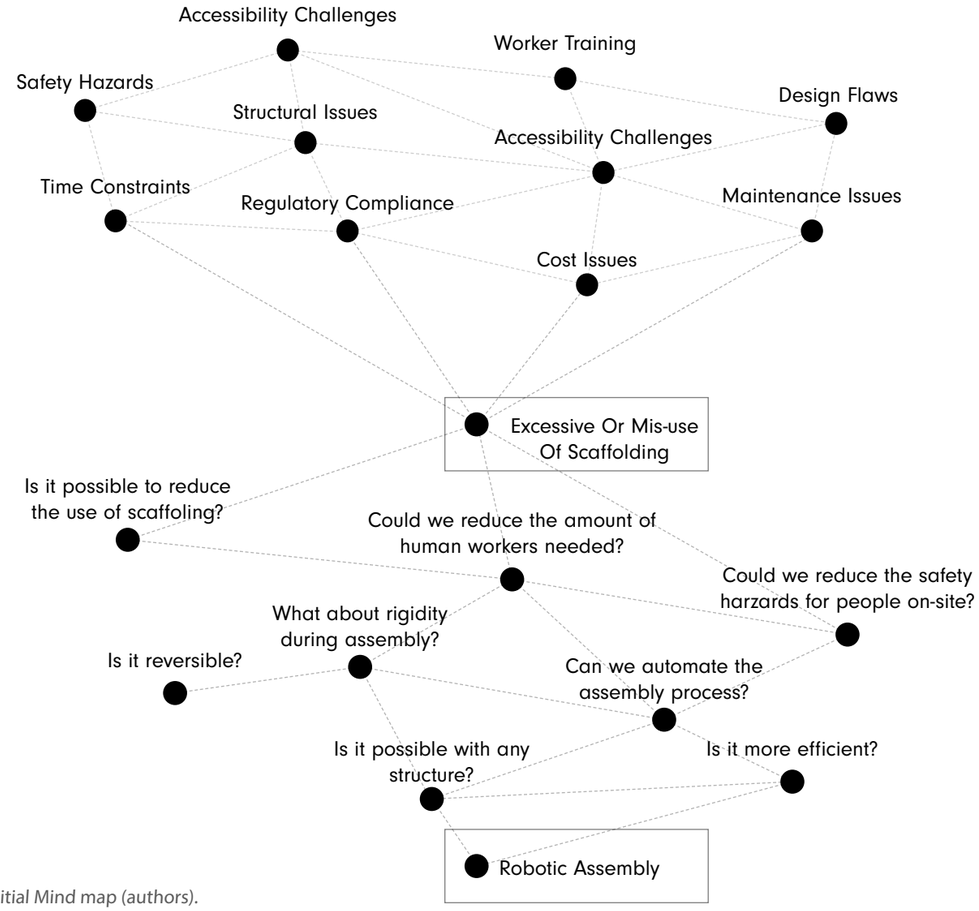


Figure 6: Initial Mind map (authors).

Research Questions

The primary research questions guiding this project include:

a) Can scaffolding be entirely eliminated through robotic assembly?

b) Is it possible to assemble large structures, such as space frames, with robots without compromising structural rigidity?

c) Additionally, can the assembly process be made reversible to ensure a reliable disassembly mechanism?

Addressing Key Challenges
These research questions address the key challenges of implementing robotic systems in construction by focusing on three critical aspects:

1. Feasibility:
Determining whether robotic assembly can practically replace traditional methods involves assessing the technological capabilities of current robotics, the adaptability to various construction environments, and the integration with existing construction workflows.

2. Structural Integrity:
Ensuring that robotic assembly does not compromise the strength and durability of large structures is essential. This involves rigorous testing and validation of robotic techniques to meet or exceed industry standards for building safety and performance.

3. Adaptability:
The ability to reverse the assembly process introduces flexibility in construction projects, allowing for modifications and updates without significant waste or disruption. This adaptability is vital for creating sustainable and resilient infrastructure that can evolve with changing needs.

Conclusion
The integration of co-working robotic technology in construction holds the promise of revolutionizing the industry by addressing its most pressing inefficiencies. By reducing the reliance on scaffolding, minimizing material waste, lowering labor costs, and enhancing safety and precision, robotic assembly can significantly improve the productivity and sustainability of construction projects. The research questions outlined will guide the exploration of these technologies' feasibility, structural integrity, and adaptability, paving the way for a more efficient and scalable construction industry poised to meet the demands of a rapidly urbanizing world.

Figure 7: 2 robots working with a human operator to assemble a timber structure (https://link.springer.com/article/10.1007/s41693-024-00137-7).

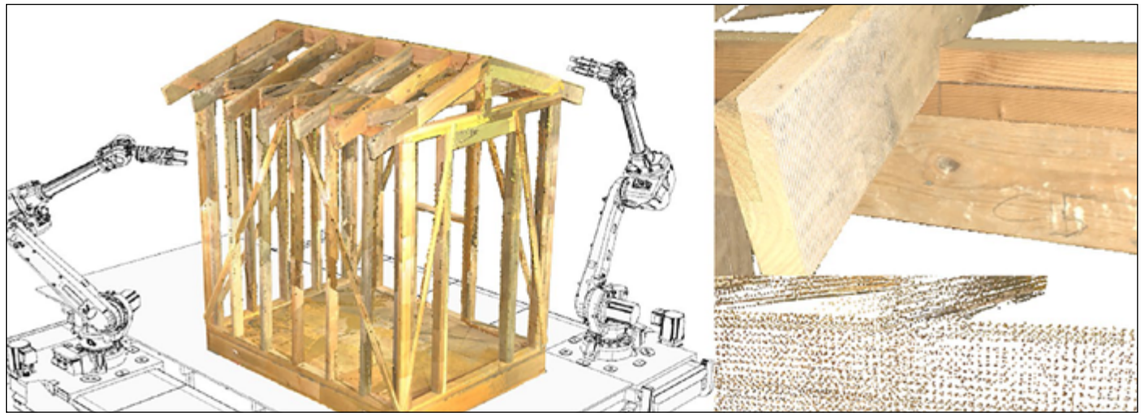


Figure 8: Robots working together making the process more efficient (https://link.springer.com/article/10.1007/s41693-024-00137-7).

A4. Case Studies
The Zero-Waste Project

The ZeroWaste project demonstrates the use of cooperative robotic systems for scaffold-free disassembly and reassembly of timber structures, incorporating circular economy principles such as material reuse. This project employed robotic arms equipped with 3D cameras to capture as-built geometric data, which guided the precise operations of the robots. Although the current project does not utilize cameras or sensors, this gap is addressed by leveraging computational modeling and predictive simulations to anticipate geometric and structural conditions. By using tools such as Grasshopper, Karamba, and Python, these predictive capabilities allow for structural analysis and path-planning without reliance on real-time sensor feedback.

The ZeroWaste project emphasized the importance of topological sequencing and the development of support hierarchy graphs to ensure scaffold-free robotic assembly. This method guarantees structural stability at every stage, which aligns directly with the goal of the current project to eliminate scaffolding from construction processes. Instead of cameras, computational simulations are used to plan precise robotic sequences that ensure stability during the assembly of large structures like space frames.

In addition, the ZeroWaste project focuses on material reuse by dismantling and reassembling timber components into new configurations, reflecting a commitment to circular construction practices. This aligns with the project's goal to reduce material waste and improve sustainability. Although the current project may not focus on timber specifically, the principles of reuse and modularity are directly applicable, supporting the aim of cutting down on material consumption and time-intensive processes.

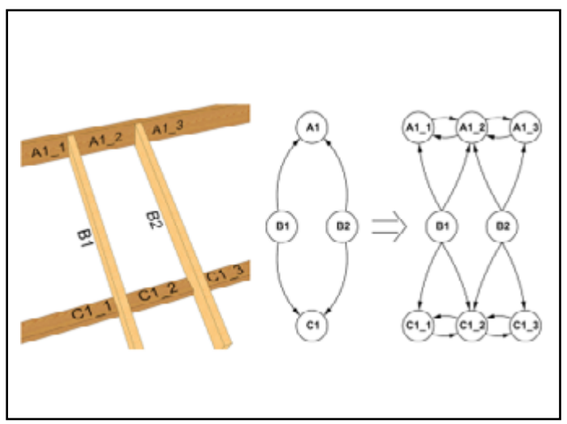


Figure 9: Element structural hierarchy (https://link.springer.com/article/10.1007/s41693-024-00137-7).

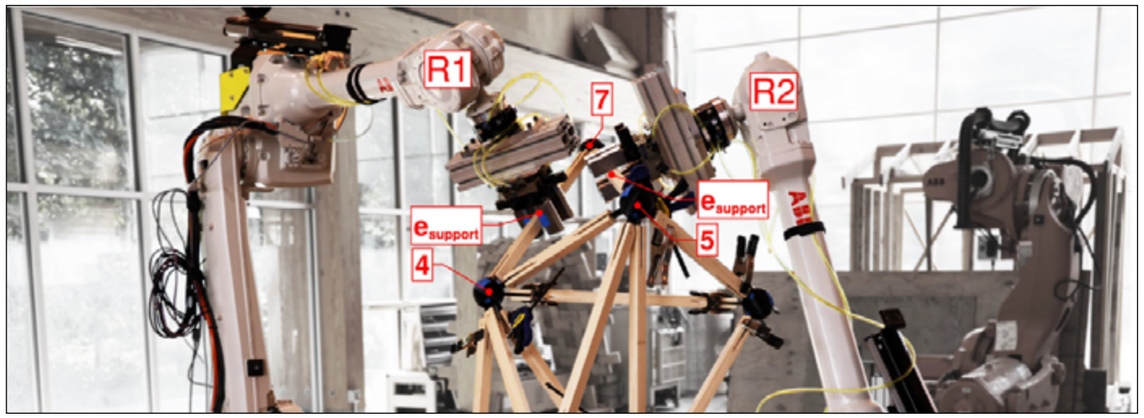


Figure 10: Both robots supporting the structure following a coworking task allocation process according to a rigidity graph. (Bruun et al., 2022, p. 18).

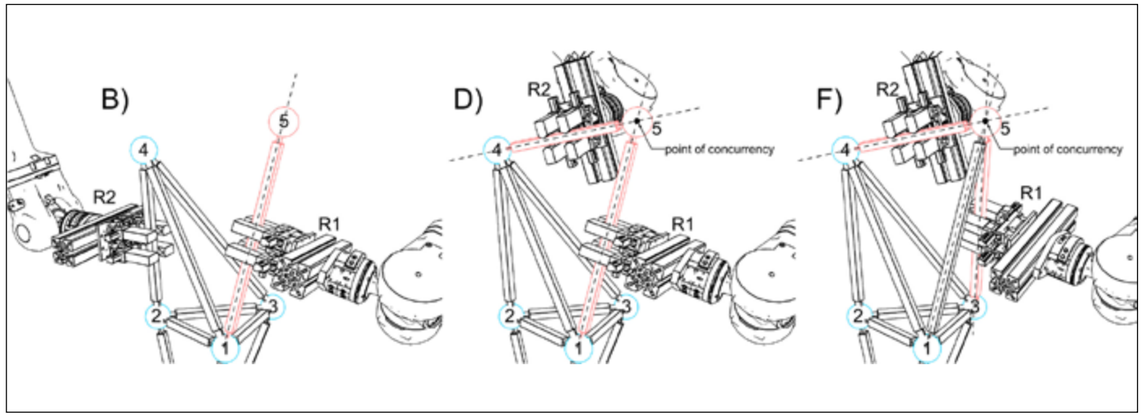


Figure 11: Points of concurrency (Bruun et al., 2022).

A4. Case Studies
Scaffold Free Robotic Assembly

The research on structural rigidity theory applied to space frames provides a computationally-driven framework for scaffold-free robotic assembly, particularly focusing on the rigidity theory of space frames. This work uses graph theory and a rigidity-preserving design process, ensuring that structures remain stable throughout robotic assembly without external supports. Unlike the ZeroWaste project, which relied on cameras for real-time data, this study shows that stability can be ensured purely through computational techniques—a critical approach for the current project, which does not use real-time feedback systems.

The study employs Henneberg assembly steps to guarantee that space frames retain their rigidity as they are assembled or disassembled. The use of rigid graphs and tetrahedral cells ensures that the structure remains stable during every stage of assembly. This methodology is particularly relevant to the project's goal of eliminating scaffolding while maintaining structural rigidity during robotic construction. By using computational simulations to determine rigidity, the need for real-time sensors is removed, making it possible to pre-plan assembly sequences based on computational predictions.

The research also introduces graph-based algorithms to generate disassembly sequences, ensuring that the process is reversible and maintains stability at every stage. This ties into the project's focus on creating a modular, reversible assembly system that maintains structural performance throughout the process. The mathematical modeling in this study serves as a foundation for developing computational simulations that can ensure structural stability in the absence of sensor feedback.

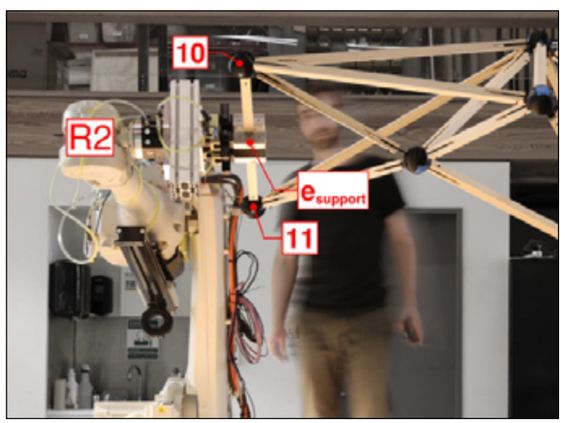


Figure 12: Human-Robot Collaboration (Bruun et al., 2022).

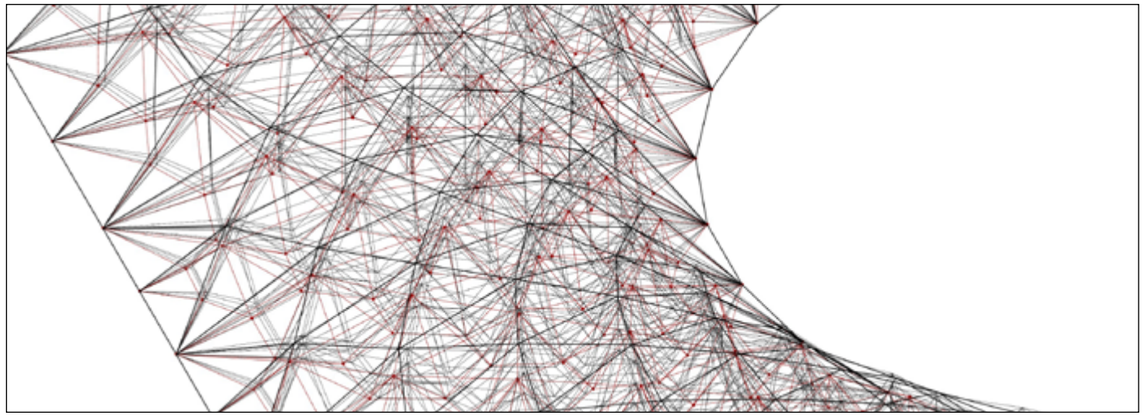


Figure 13: Analysing and reconfiguring the design of an existing structure. (authors)

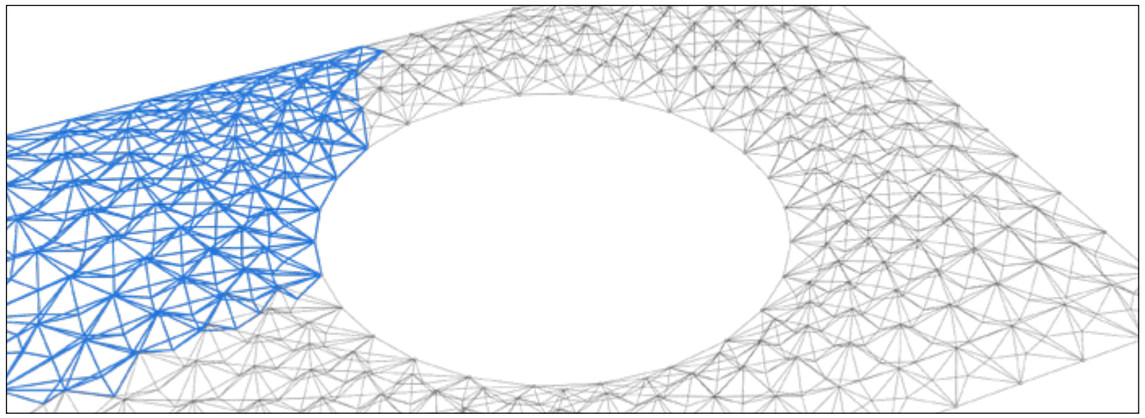


Figure 14: Creating the assembly sequence algorithm, providing us the order of assembly. (authors)

A5. Research Methods & Obejctives

To address these questions, the project outlines several research and design objectives. These include developing a co-working robotic assembly system capable of handling complex construction tasks, ensuring the rigidity and stability of the assembled structures, and creating a reversible assembly process for easy disassembly and maintenance. The project also emphasizes optimizing the geometry for better structural performance and efficient task allocation among the robots. A crucial objective is to evaluate the performance of the proposed robotic methods against traditional construction techniques to confirm their effectiveness and practicality.

Our strategic approach starts with the assembly of space frames using two collaborative robots (cobots). We utilize advanced computational tools like Grasshopper, Karamba, Python, and RoboDK to streamline the process. First, we redesign structural elements within specific geometric and topological constraints to ensure they work with robotic assembly. Grasshopper, integrated with Rhino, allows us to create and optimize space frame designs parametrically. Karamba, a structural analysis plugin for Grasshopper, helps us assess and improve the structural integrity of these designs. and efficiently.

Python scripting automates and customizes workflows, ensuring seamless integration between design and analysis. RoboDK, a robot simulation and programming software, is crucial for developing and testing the cobots, ensuring they coordinate tasks accurately.

The primary benefit we aim to achieve is removing scaffolding from space frame construction, which is typically wasteful and time-consuming. To answer these questions, our research and design objectives include developing a robotic assembly system capable of handling complex tasks, ensuring the assembled structures are rigid and stable.

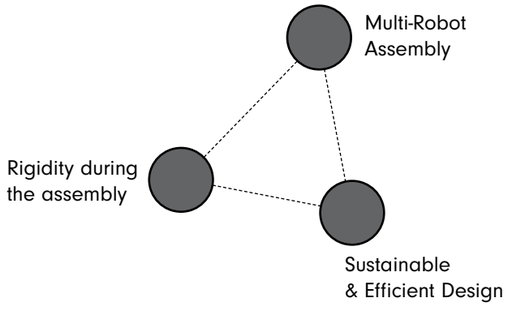


Figure 15: Project Goals (authors).

A6. Timeline

In the initial weeks of the course, our focus was primarily directed towards research and coding workshops aimed at enhancing our skills and familiarizing ourselves with the current year's topics. This year's emphasis was on automation in construction, particularly the integration of robotics. The scope of automation in construction encompasses various domains, including manufacturing, human-robot interaction, and the reduction of safety hazards. The breadth of these fields provided numerous opportunities for exploration, learning, and experimentation.

Team Formation and Research Focus

Our team coalesced around a shared objective: to transform the construction of large-scale structures in modern construction practices. We identified a significant issue in the industry—the excessive use of materials for scaffolding. Scaffolding not only consumes substantial resources but is also time-consuming and poses multiple challenges during construction. Upon reviewing existing research, we discovered projects addressing the robotic assembly of structures without the need for scaffolding. This aligned with our ambition to innovate and improve construction methodologies.

Research Directions

Our research concentrated on ensuring the structural rigidity during both the assembly and disassembly phases to prevent collapses. We delved into the Henneberg steps and Type 1 movements, investigating how to apply specific topological constraints within this framework. To ground our research in a practical context, we selected the roof of the Great Court by Foster and Partners—a renowned space frame structure—as our case study. Our objective was to reconfigure and optimize its design to demonstrate the feasibility of constructing such a structure using a multi-robot setup, thereby reducing the reliance on scaffolding and other labor-intensive tasks.

Project Development and Challenges

Our team approached the project from multiple angles:

1. Structural Redesign and Optimization: Adapted the space frame design to accommodate robotic assembly methods.
2. Algorithm Development: Created an assembly sequence algorithm to determine the order of element assembly, ensuring structural rigidity throughout the process.
3.Multi-Robot Setu: Established a system involving multiple robots to achieve proof of concept for the proposed assembly method.

Throughout the quarter, each aspect of the project presented significant challenges. Extensive research and guidance from structural physics professors were essential to address these issues. A major practical challenge was the coordination of two different robots— the UR5 cobot and the COMAU industrial robot. These robots differed vastly in their programming languages and operational protocols, complicating simultaneous usage.

Resource Constraints and Solutions

Initially, our supervisor recommended seeking external assistance due to the faculty's limited access to only one type of cobot. Our visits to BOUWLAB in Haarlem and SAM XL revealed potential collaborations; however, logistical constraints such as inadequate equipment and the necessity for additional sponsorships hindered progress. Consequently, we opted to utilize the existing equipment within our faculty.

Progress and Technical Developments

By the midterm, the assembly sequence algorithm was nearly complete, albeit with some residual bugs. On the structural front, we advanced to designing reversible nodes and optimizing them for our application. The decision to employ both the UR5 and COMAU robots necessitated the development of a robust task handling and distribution system. We leveraged Grasshopper and the RoboDK API to program and export instructions tailored to each robot.

However, practical issues arose due to the COMAU robot's limited recent usage among students and faculty, leading to unforeseen technical difficulties. Additionally, we had to install a more advanced gripper on the UR5, necessitating the relocation of the existing gripper to the COMAU robot.

Proof of Concept and Future Implications

With support from the Lama Lab and our instructors, we successfully achieved a proof of concept. The two robots were able to execute several steps in our simplified prototype design, demonstrating the feasibility of our approach. We anticipate that our work will serve as a valuable reference for future students and faculty members, mitigating the challenges we encountered and fostering continued innovation in robotic construction assembly.

Conclusion

Our project underscores the potential of robotics in revolutionizing construction practices by reducing material usage, enhancing safety, and streamlining assembly processes. Despite encountering significant challenges, our successful proof of concept highlights the viability of scaffolding-free robotic assembly. We hope our findings will inspire and assist future research endeavors in this transformative field.

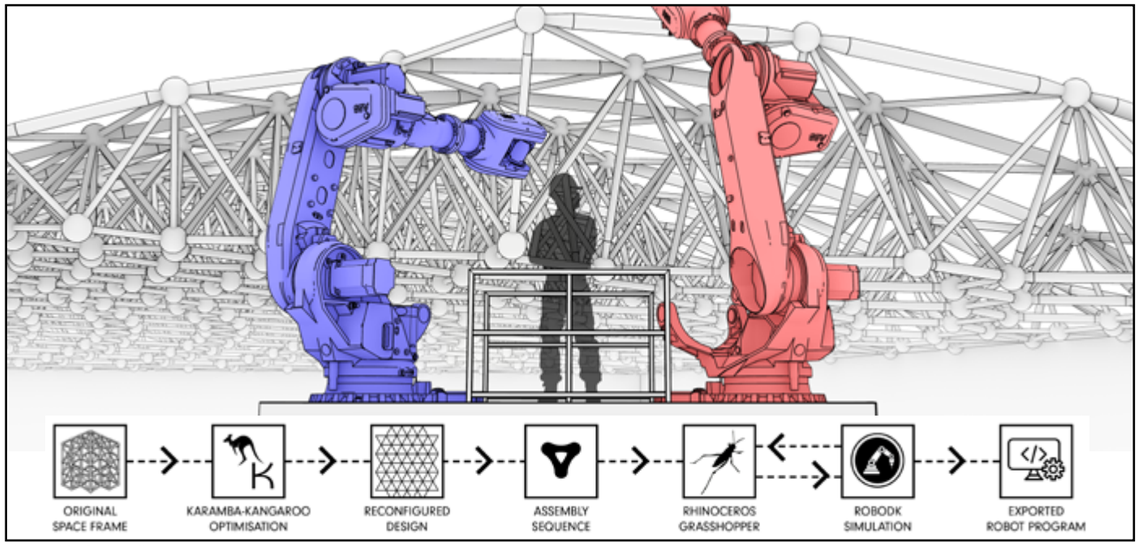


Figure 16: Project Structure (authors).

A7. Project Overview & Requirements

Below there is an overview of how data is handled and a small preview of what takes place with each step of our proposal.

1. Initial Design with Geometric Curves

Defining the Boundary and Foundational Shapes: The design begins by outlining the boundary.Within this boundary, a circle is centered and shifted 3 meters north to align with the existing context of the Great Court. Using Rhinoceros 3D and Grasshopper, foundational geometric curves are inputted to define the structure's shape.

Creating a Grid of Points: These foundational curves are divided into smaller segments to form discrete points, effectively creating a grid. This grid organizes the spatial layout and determines each connection point in the structure. The circular roof is segmented into twelve radial sections, introducing modularity and allowing different patterning rules to govern each segment's mesh.

Generating and Segmenting the Mesh for Fabrication:A mesh is generated within Grasshopper based on the grid, outlining the structure's connections and overall shape as a preliminary blueprint. To ensure practicality for construction, the mesh is divided into radial segments with specific patterning rules. Further subdivision accommodates material sizes, such as glass panes, ensuring proper fit and maintaining the structure's integrity.

2. Structural Optimization with Simulation Tools

Dynamic Relaxation using Kangaroo: The subdivided mesh undergoes optimization with Kangaroo, Grasshopper's physics engine. This dynamic relaxation process simulates physical forces to adjust the mesh, enhancing structural stability and minimizing deformation under loads. Initially, the mesh appears rigid and uniform, but as forces are applied, it relaxes into a form that balances load distribution, resulting in a more fluid roof design.

Creating a 3D Truss Model: From the optimized mesh and defined structural parameters, spatial constraints, and construction requirements, a three-dimensional truss model is developed. This model includes calculated beam lengths essential for structural analysis and fabrication. Truss elements are generated by connecting midpoints and corners of the mesh faces, ensuring each tetrahedral element is individually rigid for assembly integrity.

3. Structural Analysis with Karamba

Defining Load Cases and Material Properties: Various load cases are established to simulate real-world conditions, including self-weight, glass weight, wind loads, and maintenance loads. Detailed specifications for beams and supports, encompassing material properties and placements, are inputted into Karamba.

Analyzing Structural Performance: Karamba evaluates the structural performance under the defined load cases, assessing stress distribution, deformation, and overall stability. This analysis ensures the structure can withstand intended loads without failure.

Optimizing the Structure: Based on Karamba's analysis, the structural model is refined to meet performance criteria. Using algorithms like Karamba's Optimize Cross Sections, parameters such as allowable deflection and maximum utilization factors are adjusted to achieve an efficient distribution of truss beams. This optimization reduces the overall weight while maintaining structural integrity, suggesting a more efficient load-bearing solution compared to traditional methods.

4.Support System Development

Calculating Loads: The self-weight of the structure is determined by Karamba, with node weights overestimated by 20-30%. Glass weight is set at 315 tons, and maintenance load at 1 kN/m², excluding non-governing snow loads.

Running Simulations and Optimizations: Initial truss models with identical cross-sections are optimized using Karamba to adjust allowable deflection and utilization factors.

6. Assembly Validation and Support Placement

Validating Rigidity During Assembly: The truss must maintain rigidity during assembly. The Karamba model is deconstructed and reconstructed for each assembly step, running new simulations to calculate deformation and axial forces. Temporary supports are introduced where utilization factors and deformations exceed limits to prevent structural collapse.

7. Final Refinements and Optimization

Exploring Geometric Configurations: The input parameters for the compression domes are adjusted to explore various geometric configurations. Multiple compression domes are simulated, and their corresponding trusses are generated. These trusses undergo the full range of Karamba optimization steps, and their weights are recorded. The most efficient dome shape is selected based on these simulations.

8.Planning Robotic Assembly

Defining Robotic Parameters: Two robotic arms, Robot A and Robot B, are designated for assembly tasks. Their specifications—IDs, base positions, reach limits, and reference frames aligned with the structure's coordinates—are defined.

Calculating Kinematics: A kinematic algorithm(RoboDK api) within Grasshopper calculates the required movements for each robot. It solves for joint positions and trajectories to reach target points while adhering to mechanical constraints.

Conducting Reachability Analysis: The algorithm outputs include robot movement plans and reachability analyses, confirming that the robots can perform their assigned tasks without issues.

9. Task Distribution and Simulation

Optimizing Task Allocation: A task distribution algorithm divides the assembly tasks between Robots A and B, optimizing for reachability, efficiency, and structural stability during assembly.

Simulating Robot Programs with RoboDK: The robot programs are integrated into Grasshopper using the RoboDK API. Simulations are run to verify paths, sequences, and actions, checking for potential issues like collisions or unreachable areas.

Adjusting Based on Simulation Feedback: Any identified issues are corrected by adjusting the robot programs within Grasshopper.

10. Final Assembly Execution

Executing the Robot Programs: The verified programs are uploaded to Robots A and B. They execute the assembly sequence, constructing the structure step by step.

Maintaining Structural Stability: Temporary supports, such as crane supports, are used as needed to maintain stability throughout the construction process until key connections are secured.

Completing the Structure: The assembly results in a completed physical truss structure, built efficiently and accurately through robotic assembly.

11. Data Management and Integration

Seamless Data Transfer: Throughout the workflow, data is carefully managed and transferred between stages using Grasshopper as the primary interface or an external Python application parsing data into grasshopper using Hops.

Integration of Software Tools: The project leverages multiple software tools:

Rhinoceros 3D: For initial modeling and geometry input.

Grasshopper: Serves as the central platform for visual programming and integration.

Kangaroo: For physics-based mesh optimization.

Karamba: For structural analysis and optimization.

RoboDK & RoboDK API: For simulating and verifying robotic paths and programs.

Ensuring Workflow Efficiency: Each software tool processes specific aspects of the project, but all are interconnected through Grasshopper, ensuring a smooth and efficient workflow.

Key Takeaways:

Advanced Software Utilization: Leveraging tools like Rhinoceros 3D, Grasshopper, Kangaroo, Karamba, and RoboDK allows for sophisticated design and analysis.

Robotic Precision: Programming Robots A and B with detailed movement plans ensures accurate and efficient assembly.

Optimization and Efficiency: Using algorithms for material cutting and task distribution reduces waste and optimizes the construction process.

Integrated Workflow: Seamless data transfer and software integration are crucial for project success, highlighting the importance of a unified computational environment.

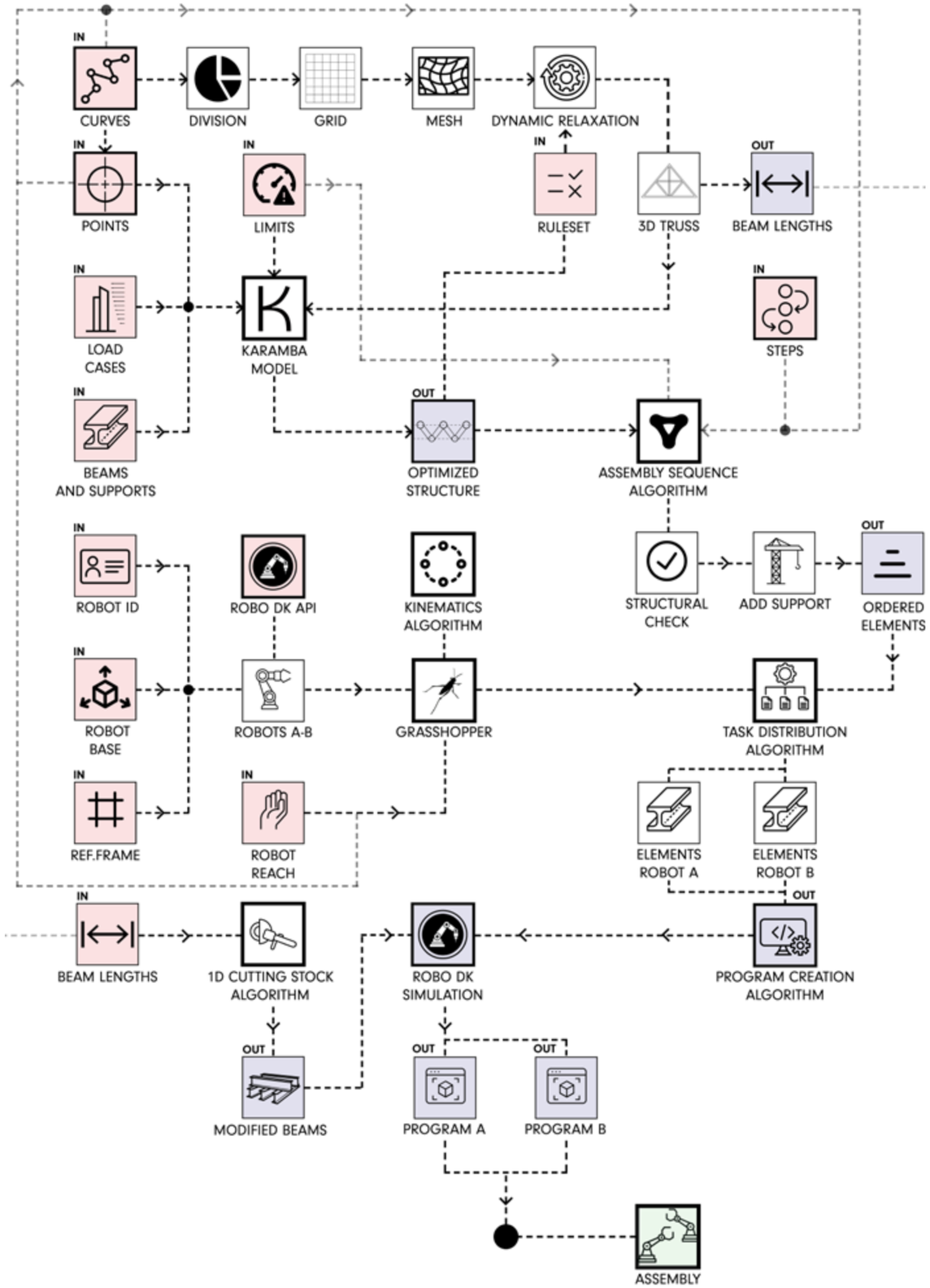
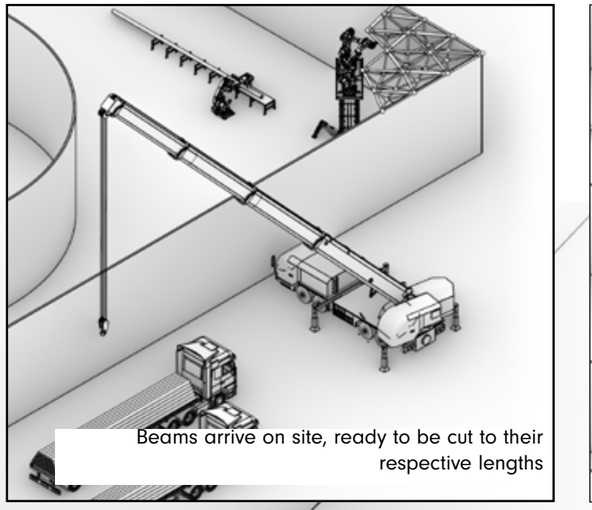
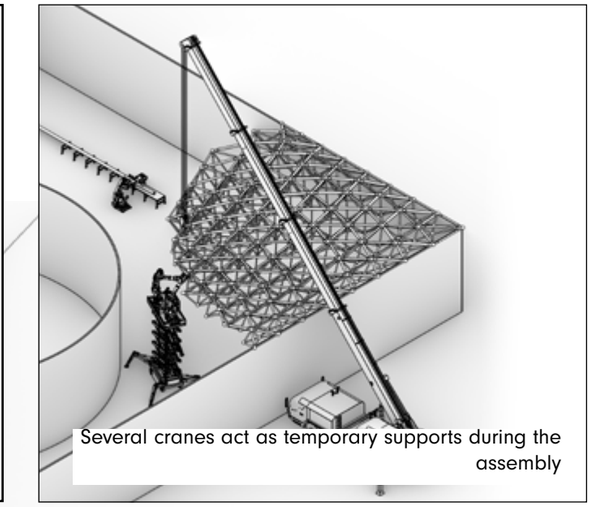


Figure 17: System Overview Diagram (authors).



Beams arrive on site, ready to be cut to their respective lengths

Figure 18: Onsite Assembly A (authors).



Several cranes act as temporary supports during the assembly

Figure 18: Onsite Assembly B(authors).

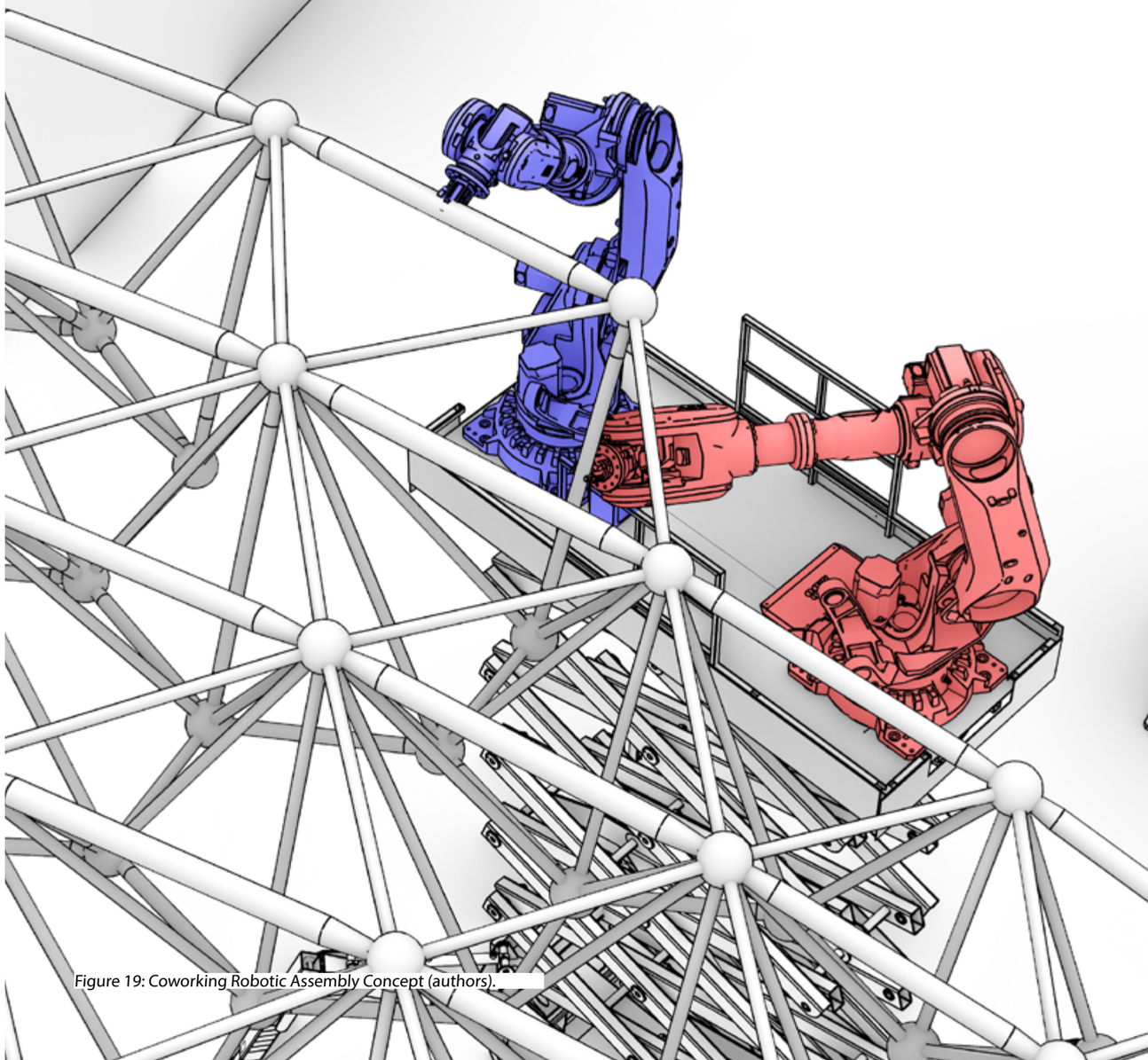


Figure 19: Coworking Robotic Assembly Concept (authors).

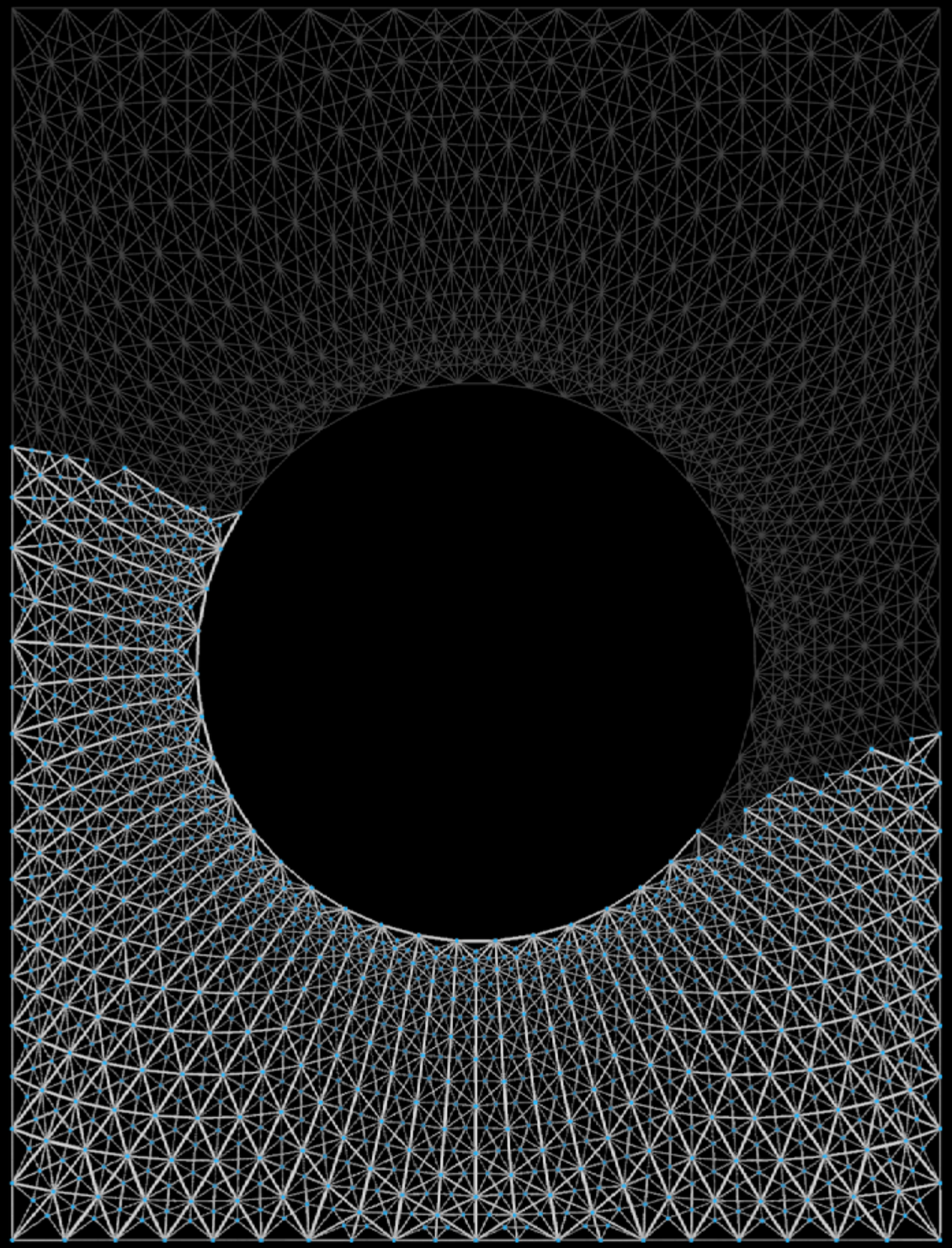
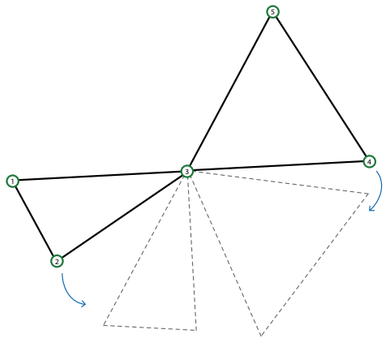
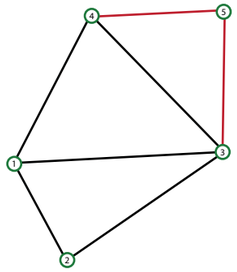


Figure 20: The assembly sequence algorithm iterating through the structure assembly (authors).

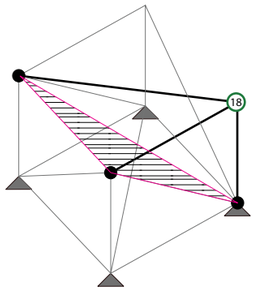
B.ASSEMBLY SEQUENCE



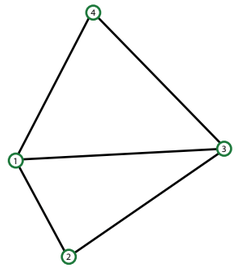
A Non-Rigid Graph in 2D



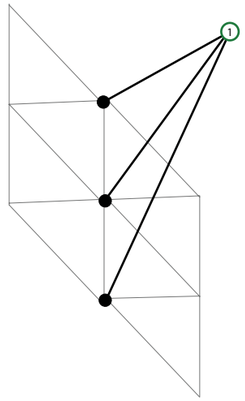
A Henneberg Type 1a step



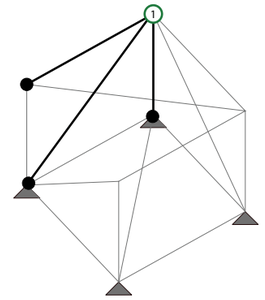
Support points are well positioned, and added node is rigid



A Rigid Graph in 2D



Support points are co-linear (and co-planar)



Support points are only co-planar

Figure 21: Checking for rigidity (authors).

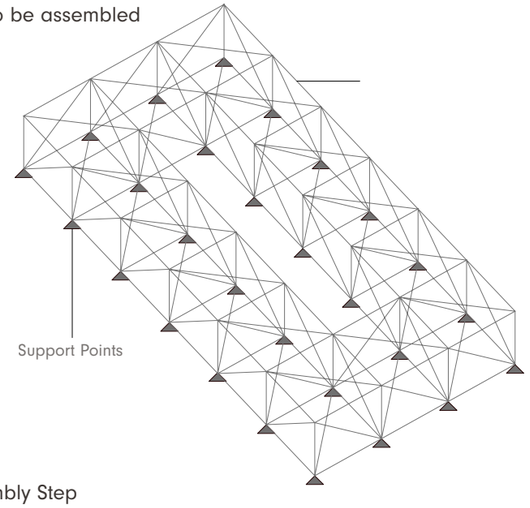
B1.Rigidity

The assembly sequence algorithm is primarily concerned with searching for an assembly sequence of a lattice structure which preserves the rigidity of the structure along every assembly step. The ideas used to ensure rigidity along every step are taken from graph theory, where rigidity is defined as the inability of any one node to translate in x,y,z relative to another node. This is to ensure that the structure is at least infinitessimally rigid, i.e under infinitessimally small forces, the structure will not deform. This is useful because it tells us that every step is 'geometrically' rigid, i.e no node has the ability to translate, assuming proper structural design (seen in later chapter).

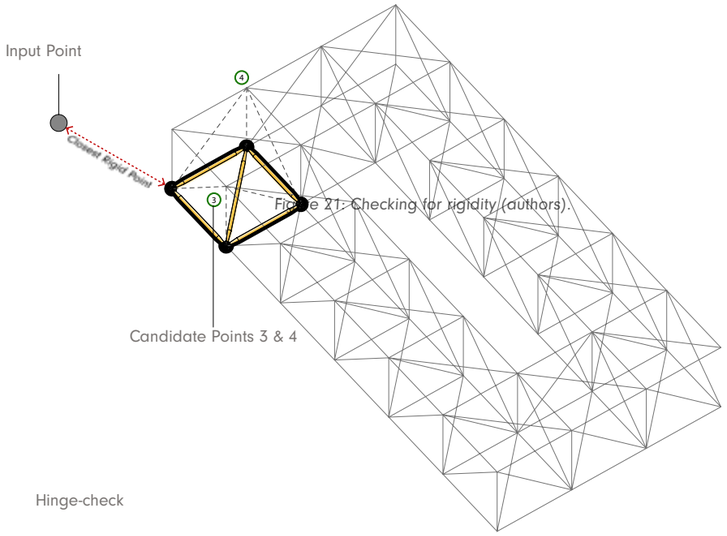
B2.2D vs 3D

While rigidity can be proven given an input graph in 2D, there exists no general algorithm or mathematical proof for 3d graph (Chubynsky and Thorpe, 2007)(Roth, 1981)(Sitharam, Zhou, 2004). While the sources verifying this are relatively old, little work has been done to find a general solution, with some approaches resorting to approximate or numerical solutions, much like FEA. Nevertheless, we used ideas from 2D Graph rigidity, such as the Henneberg step. Which ensures that if a node is added to a graph and connected to two existing rigid nodes, the node added is rigid. And if a graph is constructed this way, the graph is rigid. However, this does not work in 3D as the support points of the node to be added can be co-linear (and thus co-planar), or only co-planar. if the points are co-linear then the node that is added can fully hinge. If the supports are only co-planar, then the step can be geometrically rigid in theory. However, this assumes elements to be infinitely stifg, in reality the internal forces induced by the loads on the structure would very likely cause exceedingly high deformations or stresses. This is why, support point co-planarity is also avoided.

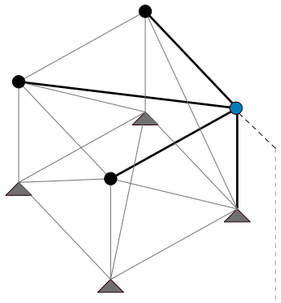
Structure to be assembled



First Assembly Step

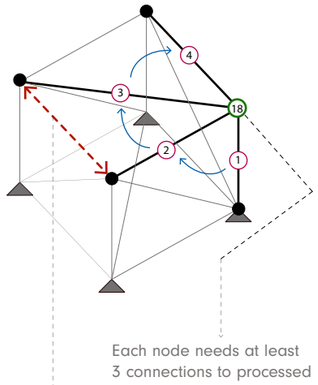


3-neighbour check



Each node needs at least 3 connections to other nodes.

Step sub-sequence



Each node needs at least 3 connections to processed nodes.

Gap of 3rd edge has to be above 1/3 of the effective span

Figure 22: Ruleset for checking(authors).

B3.Initialisation and rules

When considering the initialisation of the assembly sequence, it is imperative that it starts from a set of three rigid points. As explained above, every added node needs at least 3 points to connect to, to be considered rigid (along with several other geometric constraints). However if no nodes are added, i.e if the algorithm is searching for the first step, it needs a set of rigid points, which can be used to add nodes to. This is done by requiring an input of rigid points (Support Points) that act as the catalyst to begin the assembly sequence from. The required rules of each step were developed from version 1 to version 5 of the algorithm and are explored later.

Base-spread check

Hinge-check

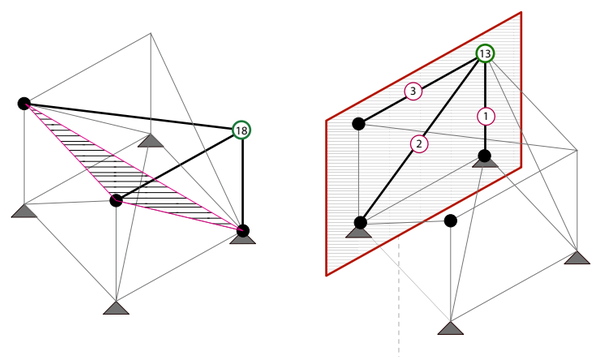
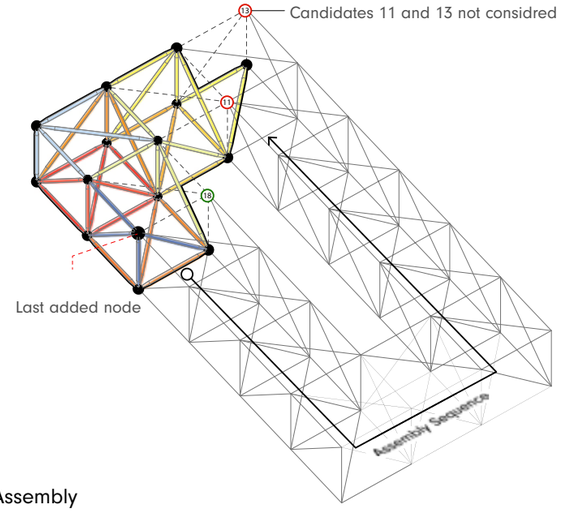
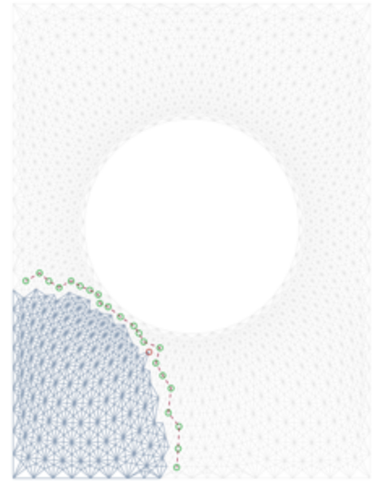


Figure 23: Hinge check(authors).

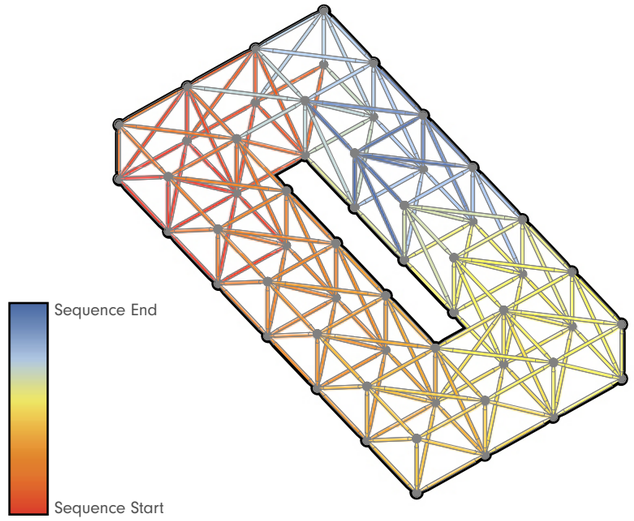
Branching Behaviour

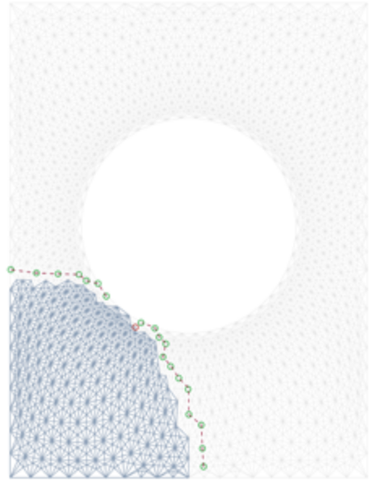




Candidates form one subgraph

Complete Assembly





Candidates form two subgraphs

Figures 24,25: Assembly step & Subgraphs (authors).

B4.Skipping Check

A general principle of the algorithm is also that it should assemble a given structure by continuing to assembly self-connected regions. This means that if a set of candidate nodes are considered to be added next, the set should be reduced to the nodes that are connected with each other and also with the last added nodes. In doing so, the algorithm only considers the subgraph of nodes which includes the node that is directly connected to the last added node.

Once a set of candidate nodes are found, i.e all nodes that can be added next in a rigid-manner and that are self-connected, they are all processed before considering new candidate nodes. This ensures that the algorithm doesnt get 'tunnel vision' and continue assembly only the first neighbour of the last added node.

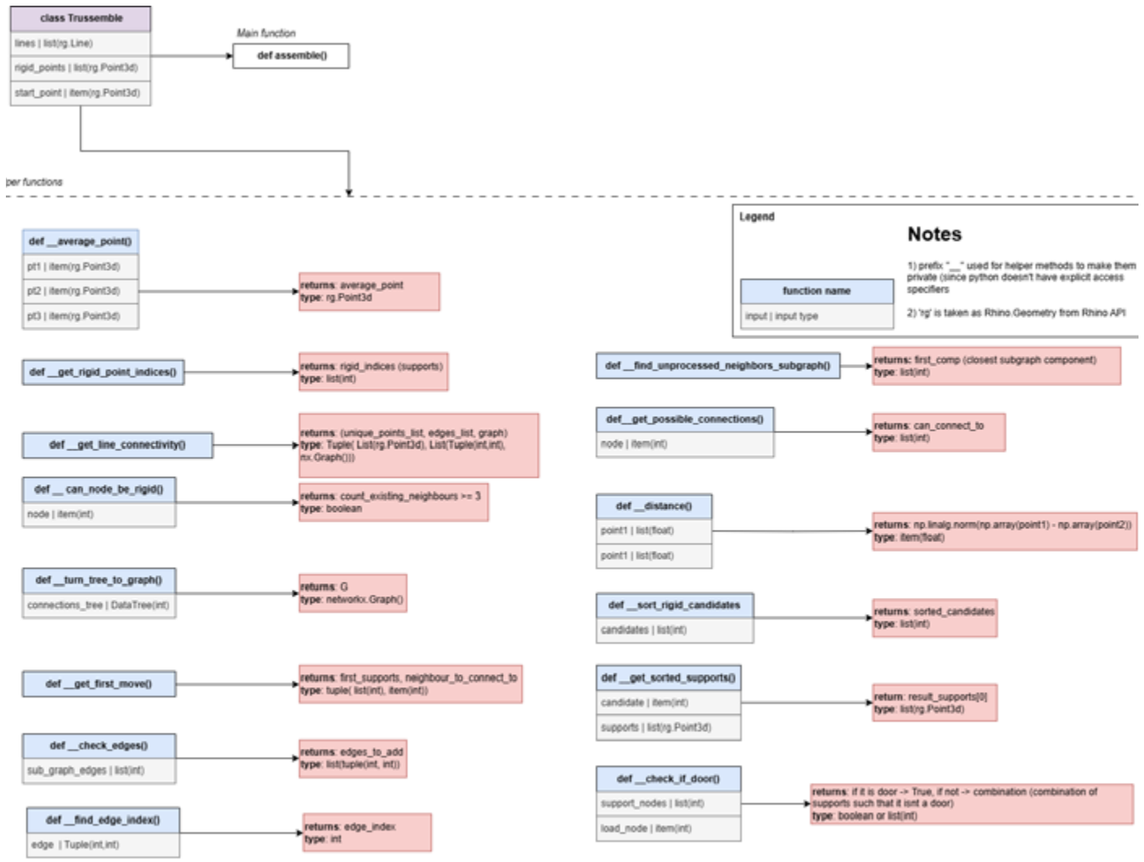


Figure 26: Analysis diagram of the assembly sequence algorithm (authors).

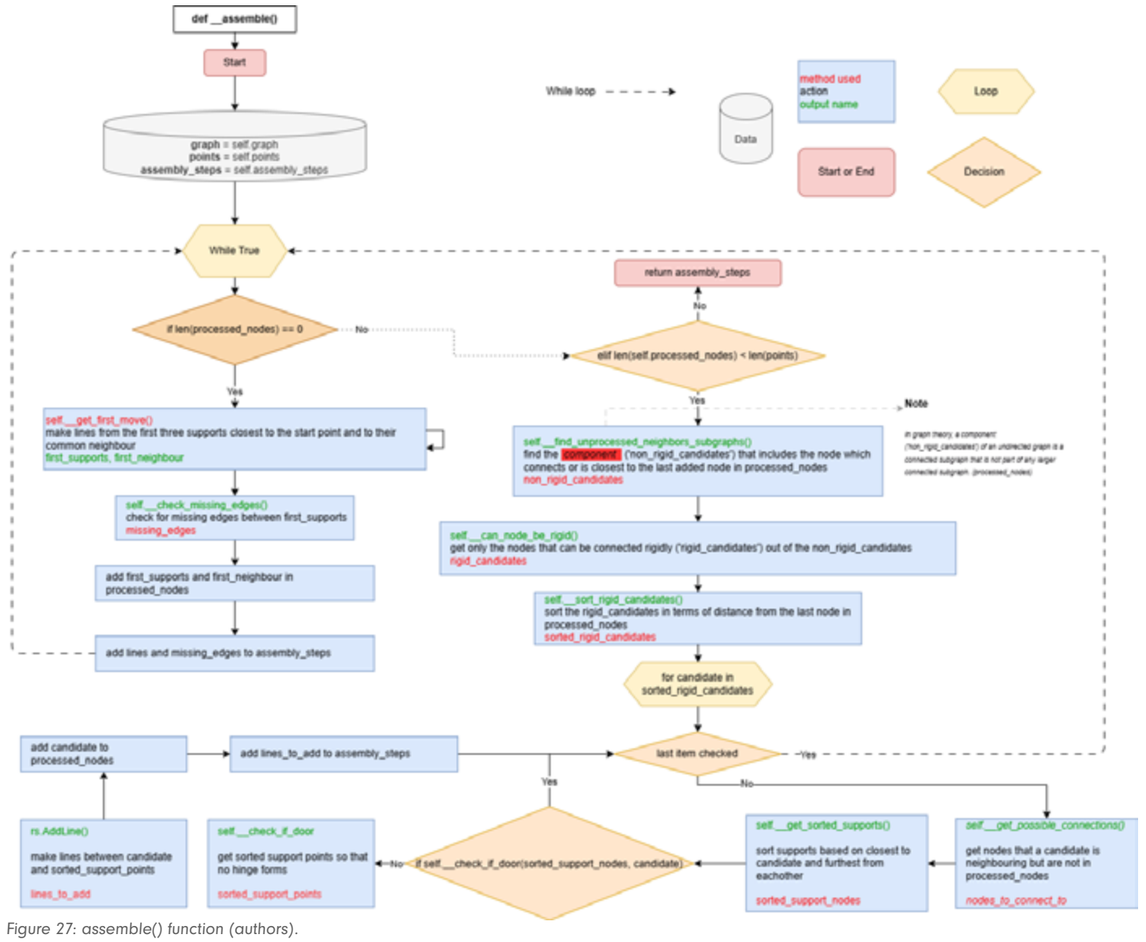


Figure 27: assemble() function (authors).

B5.Skipping Check

The assembly sequence algorithm is structured in a python class, and fully implimented within that class without the use of any grasshopper plugins or components. It was made into a class as it made going from version to version easier, by updating individual helper functions . Furthermore, it made the code easier to debug as the assemble() function remained relatrively short, because most sub-steps are completed by calling helper functions, instead of implimenting the solution in-line.

First, the inputs are validated in the __init__() function:
e.g:

```
if not all(isinstance(line, rg.Line) for line in in_lines) :
        raise ValueError('Input Lines are not Lines')
```

then the inputs are processed into a networkx graph, which is the 'environment' of the simulation where all nodes' neighbours are stores. Rigid points are converted to indices corresponding to the networkx graph. The start point is used to find the first three rigid points, from which the simulation begins.

B4.Skipping Check

The main function: assemble() revolves around a while True loop, which breaks if the number of processed nodes is equal to the number of nodes in the networkx graph. if there are no processed nodes, the first move is found by finding the closest three rigid points to the start point, which also share a common neighbour. The three rigid points and their common neighbour is then the first step of the assembly sequence. If the number of processed nodes is less than that of the netwroxk graph, the algorithn finds the desired candidates by considering the subgraphs of all candidates. Ensures all candidates can be rigid, sorts them by distance to the last processed node. And sequentially adds each candidate and the lines to its neighbours as a new step in the assembly sequence. Meanwhile, each step is checked for hinge-behaviour (door-check), but also the order of the lines from the candidate to its neighbours is considered so that the first 3 lines of the step form a rigid connection.

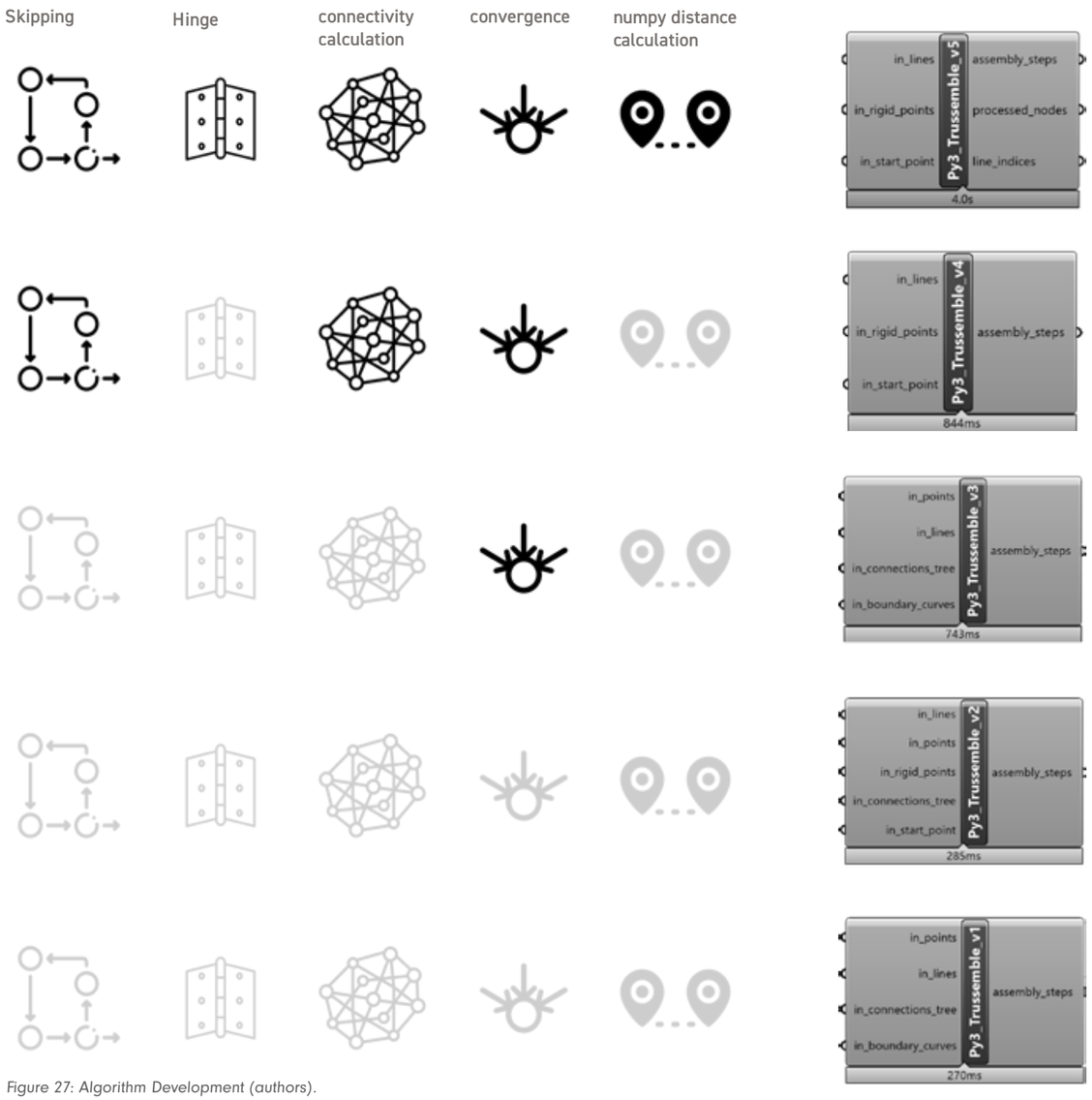


Figure 27: Algorithm Development (authors).

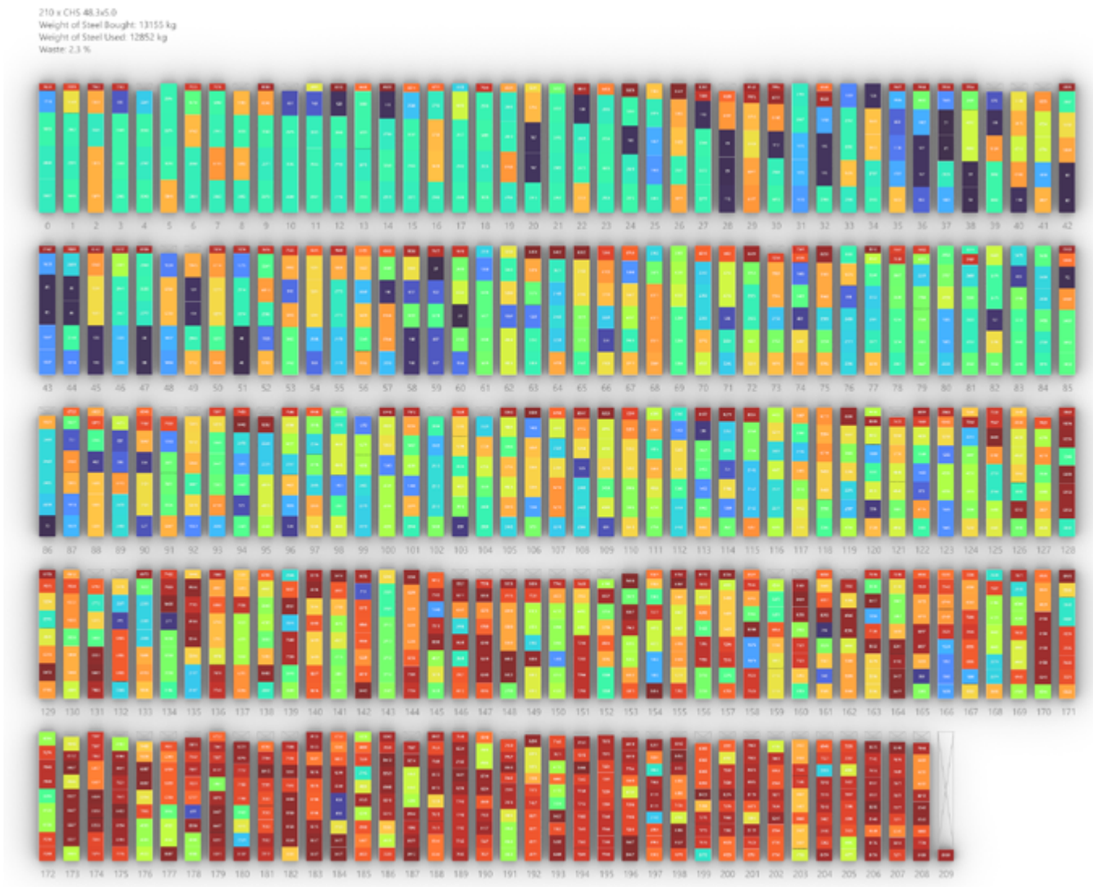


Figure 27: 1D Cutting (authors).

B5.Algorithm Development

The developement of the algorithm was done in 5 different versions. In the v1 and v2, the algoriothm wouldnt converge yet as it didnt have sufficient logic to account for missing edges, or edge cases. v3 converges but doesnt include any of the advanced features of hinge-checks or clever sequencing by considering connected subgraphs. All version from v1 to v3 rely on a plugin called Heteroptera to find the adjacency list as a datatree between nodes of the lattice. V4 considers connected subgraphs to find candidate node, and also creates the adjecncy list internally without relying on Heteroptera. Version 5 calculates all distances with numpy instead of rs.Distance() because we found that the algorithm crashed when the assembly sequence was to be found for a model in millimetres. This was because distance calculations could reach 10,000-100,000 mm and these were too large for rhinoscriptsyntax to process. Numpy allows for distances ot be calculated more reliably using numpy.linealg.norm(pointA-pointB). Furthermore, the initial check that sorts the support points of a candidate (def __get_sorted_supports()) is made using matrices to calculat pairwise distances between points, and get their product and the area of their triangle. This is done to assign a score to each ordering, based on choosing the first three supports that are the closest to the candidate but the furthest away from eachother. This allows for each step with more than three substeps, to start closer to the node and move outwards, making it more likely to be rigid. This is then verified by the hinge-behaviour check, called (def __ check_if_door())

B6.Cutting Stock Optimisation

Because of the high level of structural optimisation, elements end up being bespoke in their length, and thus cutting those elements from stock lengths of steel tubes is a problem in itself. We developed a naive solution as a control, which takes all the elements with a given cross section in order of their node index and cuts them from stock pieces in order. This was used to compare performance with a 1D Linear Cutting stock algorithm found in: https://github.com/AlexanderMorozovDesign/GH_Linear_Cutting. We tried to impliment this locally, without using the above library, but the algorithm couldnt handle more than 50 elements, even when we used Google's ortools library.

This is why we resorted to using Alexander Morozov's plugin for grasshopper which worked well for up to 1000-2000 elements. The improvements from the naive approach can be most prevelent when having to cut a large number of elements. When the elements are below 200-300 then the naive approach is either slightly better or slightly worse, i.e optimised approach might improve waste by 1-2%. However, when the elements that have to be cut are more than 600-700 then the optimised algorithm works a lot better and sometimes improves waste from 22% to 14%. The pattern of each stock element is given as the indices of beam elements to cut from each piece. i.e if stock piece one was used to cut beam element 303, 2001 and 854, its cutting pattern would be 303_2001_854. Alexander Morozov's plugin is also an improvement on the naive approach as it considers a 'tool width' as an input. Documentation on the tool is limited, but it works by sorting elements by descending length, then the "Calculate" method is used to initialise a random pattern on a stock piece, which is optimised with linear programming and then added to the result cutting patterns.

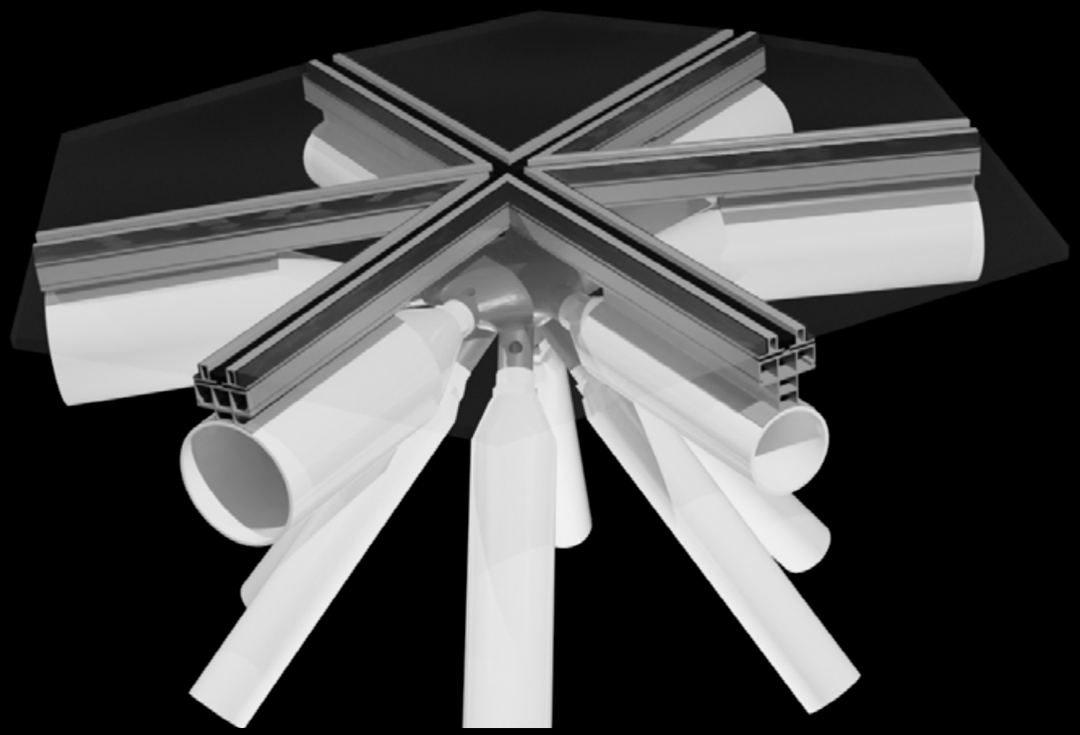


Figure 28: 3D Model of finalised detail(authors).

C.STRUCTURAL DESIGN & OPTIMISATION

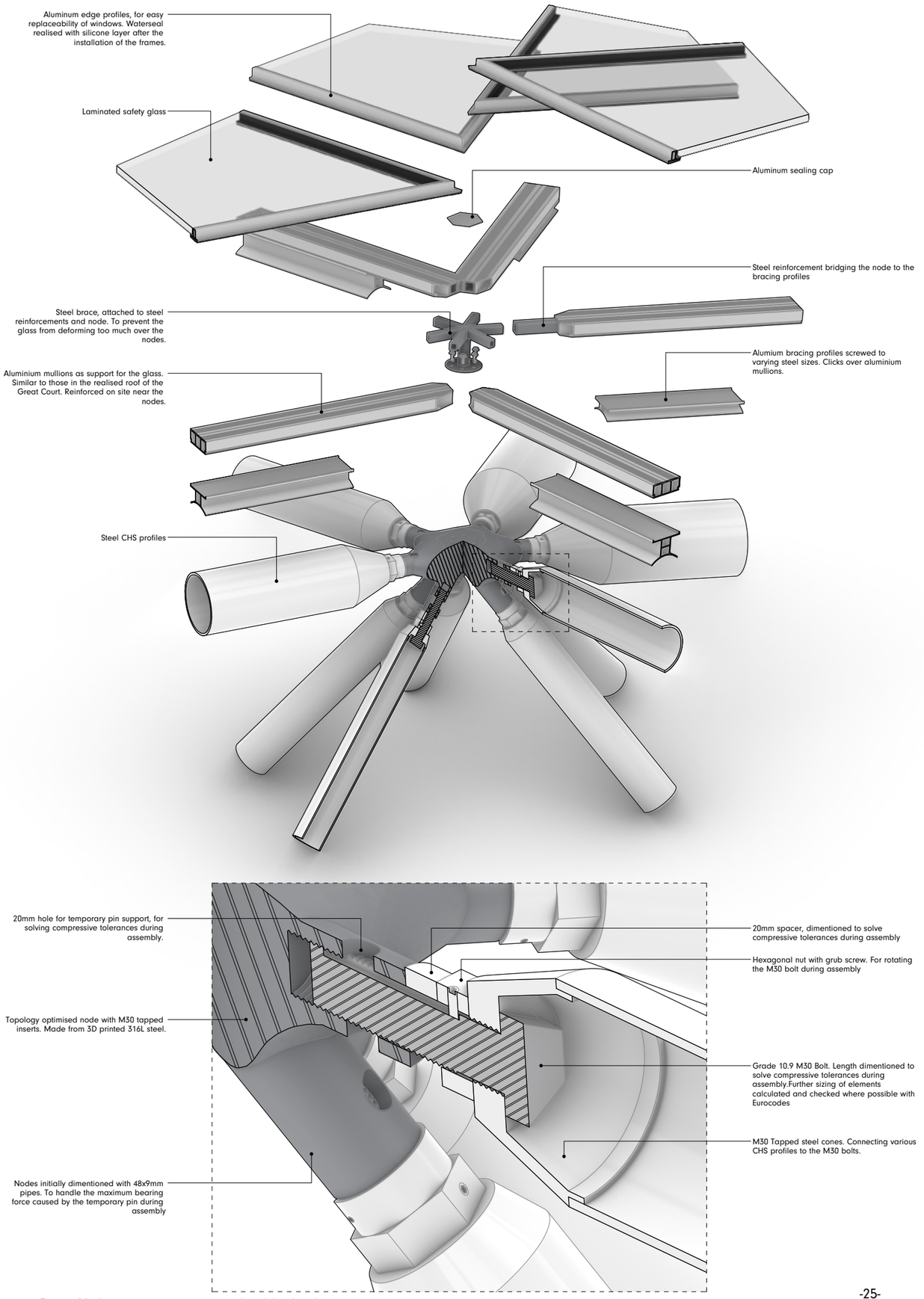


Figure 29: Axonometric construction detail (authors).

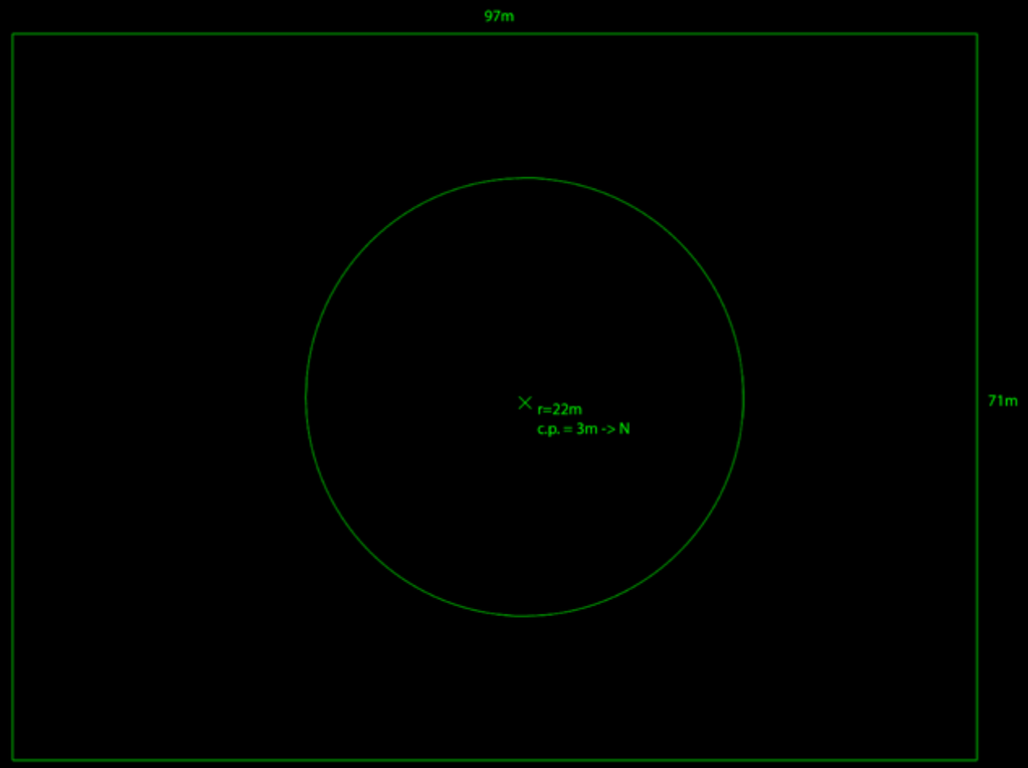


Figure 30:Input curves from the Great Court of the British Museum (authors).

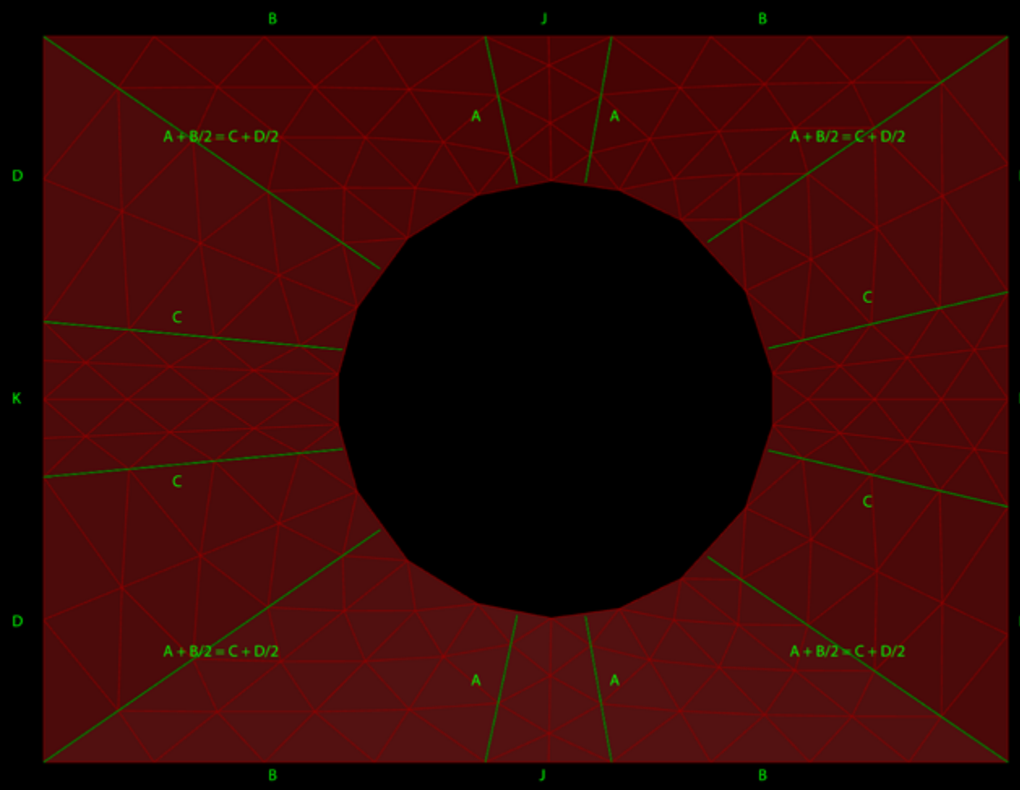


Figure 31:Mesh generation, with a rule set so that the corners correlate (A+B/2 = C+D/2) (authors).

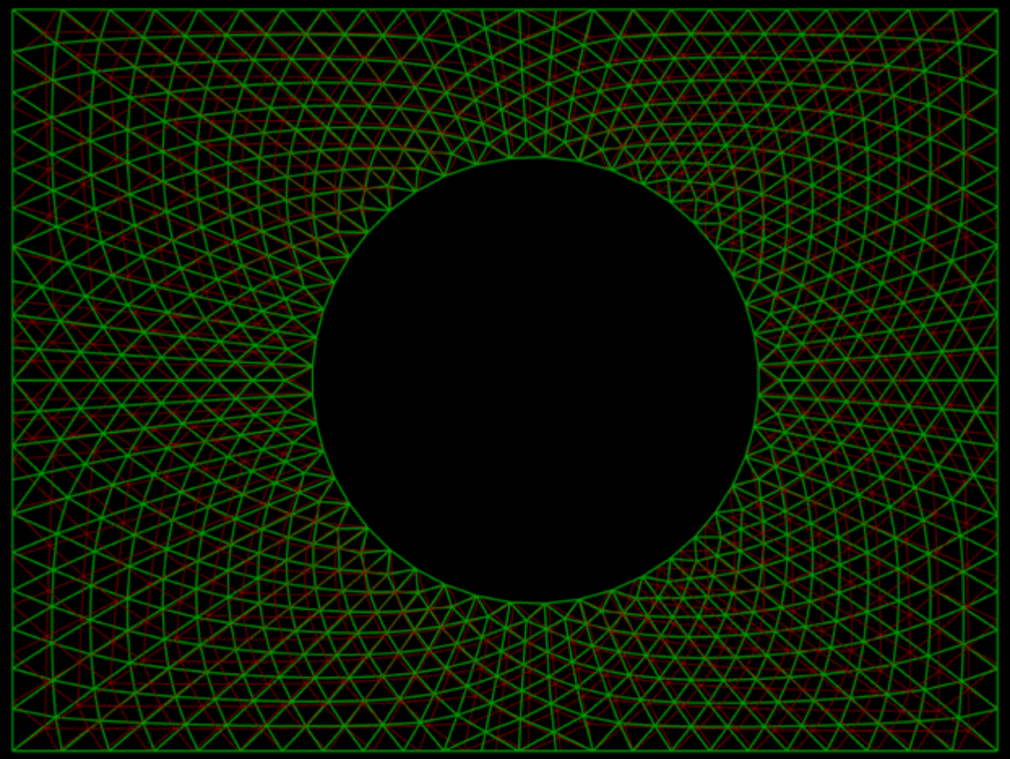


Figure 32:Mesh Generation & dynamic relaxation (authors).

C1. Initial Reconfiguration of the existing structure

The design begins with the definition of the boundary, which spans a rectangular area of 97 meters by 71 meters. Within this, a circle with a radius of 22 meters is centered, with its center shifted 3 meters to the north to recreate the existing context of the Great Court. This geometric foundation serves as the starting point for developing the roof's structure.

Next, the circular portion of the roof is divided into twelve radial segments. This segmentation introduces modularity to the design, managing the complexity by allowing different patterning rules to govern the meshes of each segment.

The process then continues with the generation of the meshes that follows a specific rule set. The rule set ensures that the vertices of the mesh satisfy a correct relationship between the corners of the triangles, ensuring geometric alignment and structural consistency. This approach results in an evenly distributed mesh.

As the design progresses, with manual input, the mesh undergoes further subdivision based on the constraints of glass pane sizing. This step ensures that the largest triangular sections of the roof are within the practical limits for fabrication, addressing concerns about material sizes and the structural integrity of the glass panels.

Following this subdivision, the mesh is prepared for dynamic relaxation using Grasshopper Kangaroo, which is a process that allows the structure to adjust to internal tension forces.

Before this relaxation process, the mesh appears rigid and uniform. However, as the forces are applied, the mesh relaxes into a form that balances the load distribution across the structure better. The result is a more fluid roof design.

The final mesh, after undergoing dynamic relaxation, represents the completed roof as seen from space. The structure is now aligned with the initial design constraints, and ready for further simulations.

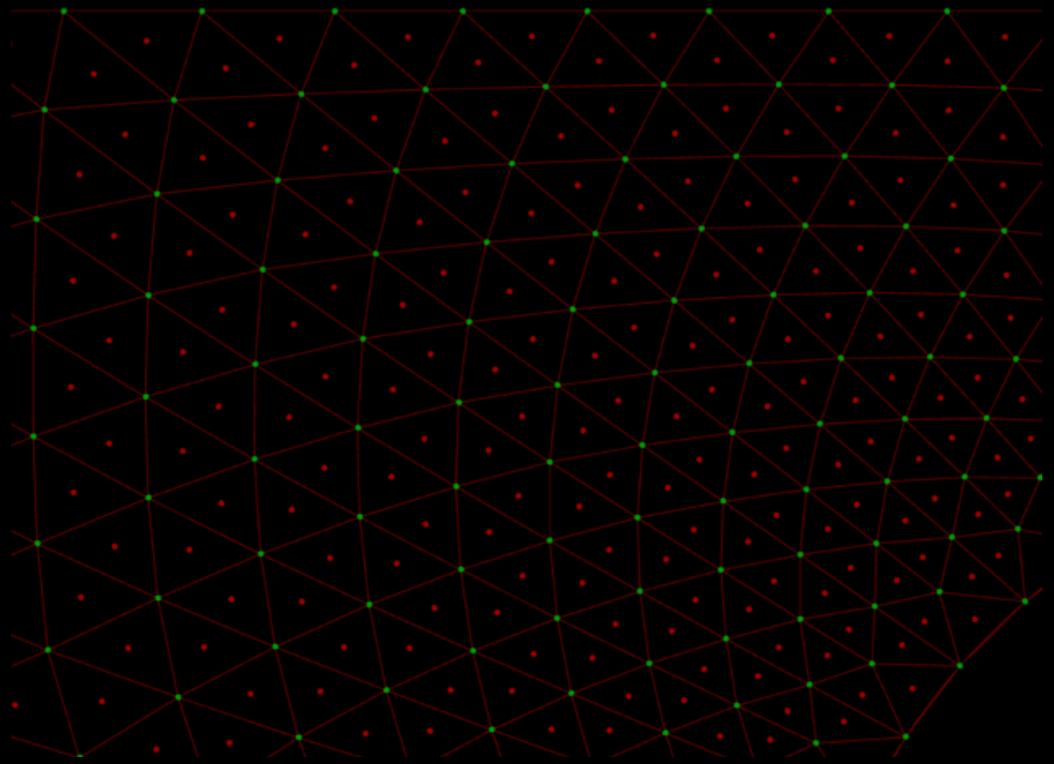


Figure 33:Finding the midpoints and the corners of the faces (authors).

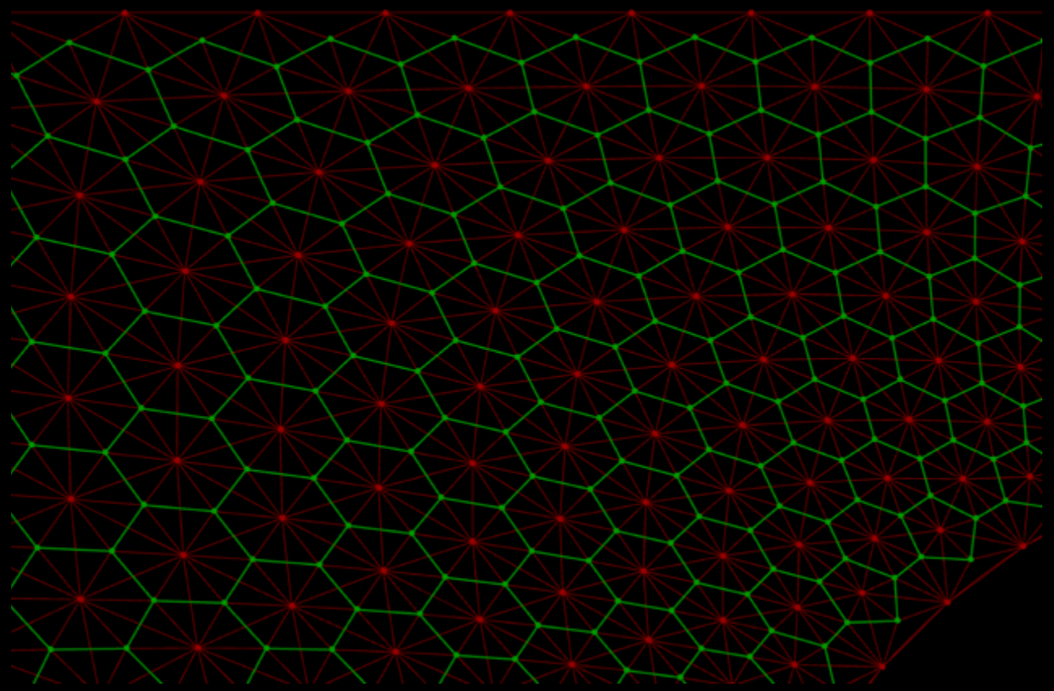


Figure 34:Connecting the endpoints of the inner trusses, base on meshface adjacency. These will be the lower truss elements (authors).

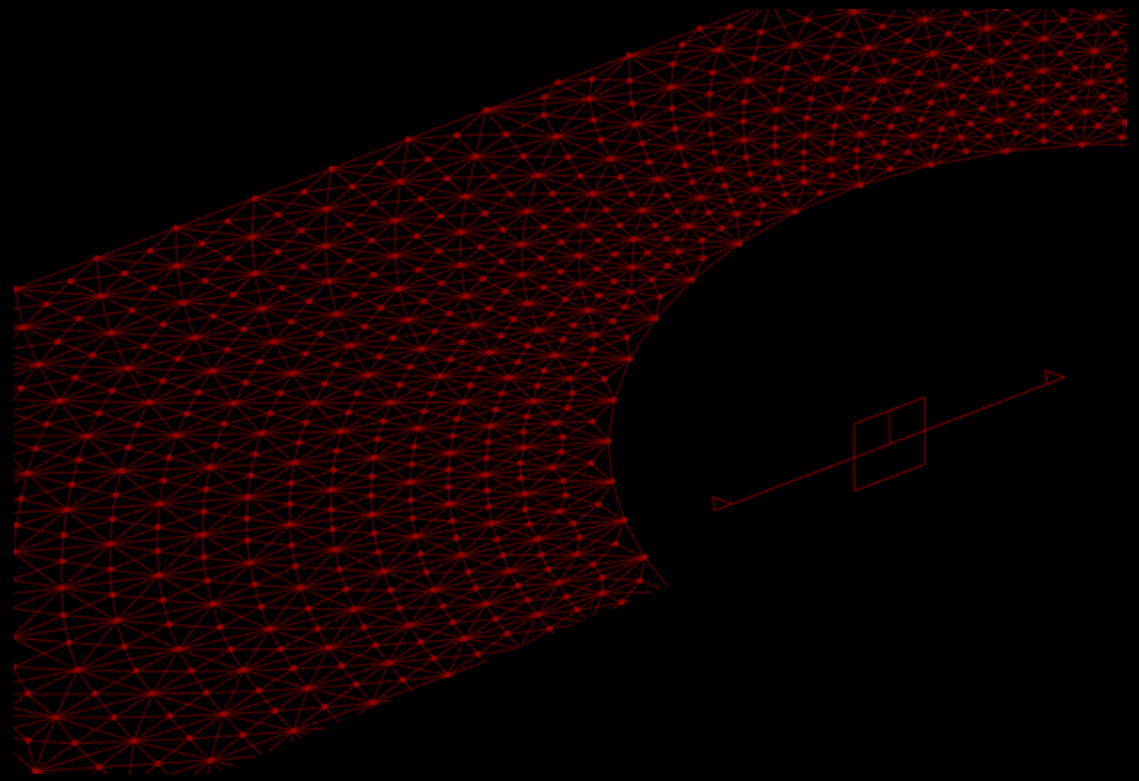


Figure 35:Initializing 3D Kangaroo dynamic relaxation simulation (authors).

C2. From 2D mesh towards 3D truss

The mesh is further developed with the creation of truss elements. The initial step zooms into a corner of the mesh, highlighting how the existing mesh will be used as the basis for generating all the other truss elements. The initial mesh represents the top truss elements on which the glass panes will rest.

The process continues by finding the midpoints of each triangular mesh face. Following this, the corners of each mesh face are also identified. These corners are then connected to the midpoints. These edges represent the middle truss elements.

Next is the process of connecting the midpoints of each triangular mesh face. Each midpoint is connected to the midpoint of neighbouring meshfaces, meaning connecting the midpoints of meshfaces that share an edge. These edges will represent the lower truss elements and are also connected to the middle truss elements.

These rules ensure that the entire truss consists of many tentrahedral elements, each individually rigid. This is important later on during the assembly process to ensure rigidity.

Nearing the the end, the naked edges of the structure are identified. These represent the boundary where the roof will connect to the existing building. These naked edges are important to identify for placing support points, ensuring that the roof is securely anchored to the surrounding architecture.

In the final stages, a 3D Kangaroo dynamic relaxation simulation is initialized. This simulation is used to pull the truss elements into an efficient 3D shape. Ensuring that the final roof structure can resist out of plane loads such as gravity and maintenance loads.

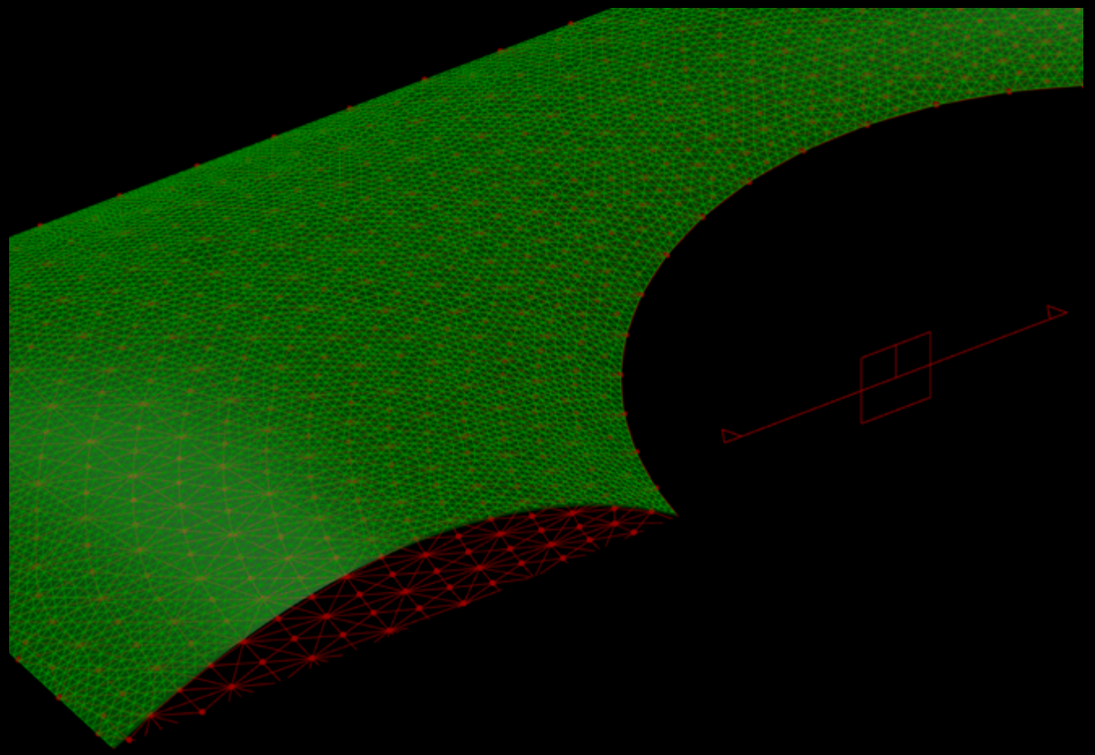


Figure 36:Finding the perfect compression only surface based on seperate Kangaroo Simulation (authors).

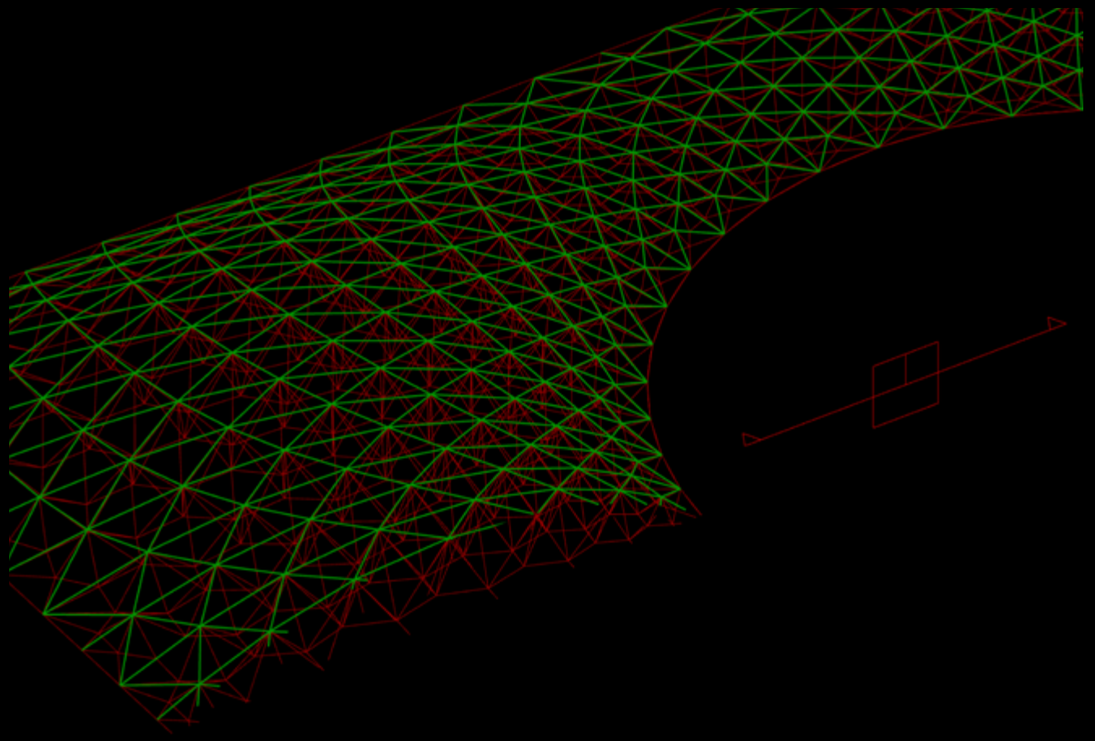


Figure 37:Top truss elements are grouped (authors).

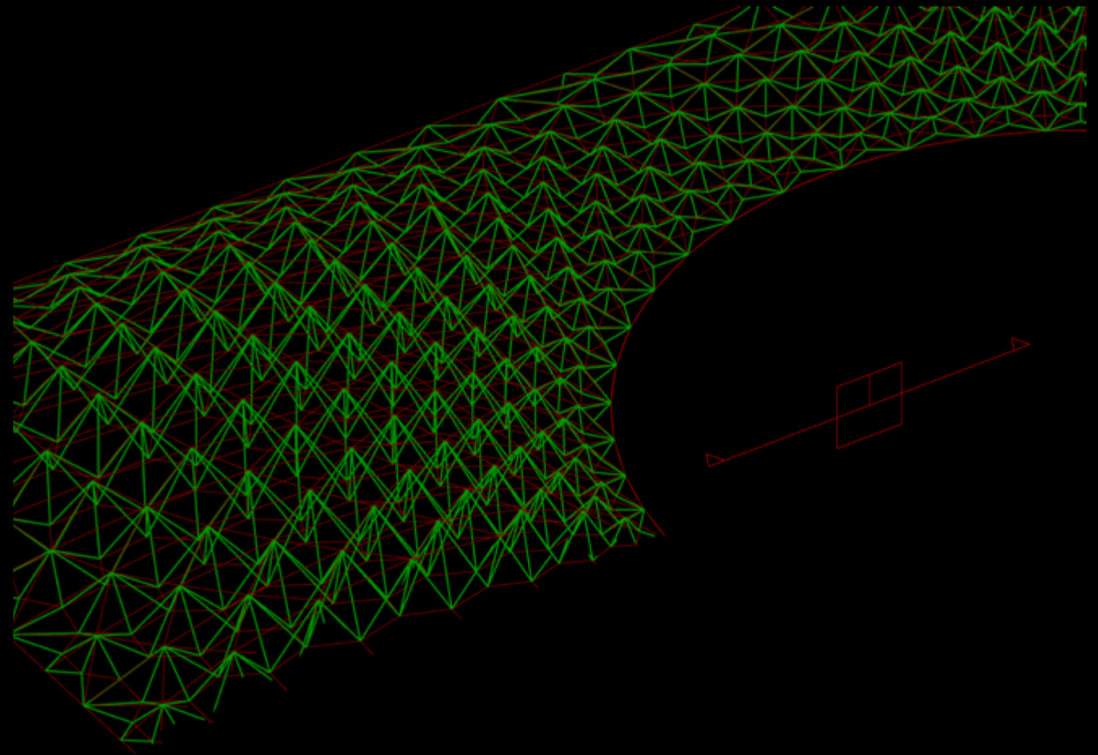


Figure 38:Inner truss elements are grouped (authors).

C2. From 2D mesh towards 3D truss

Next, the focus shifts towards finding the optimal compression-only surface and refining the truss geometry for future analysis.

The first image demonstrates the process of identifying a compression-only surface using a separate Kangaroo simulation. By allowing a mesh with equally sized faces and weights to relax, we can find a form that naturally conforms to compression forces.

Once a compression-only surface has been identified, the Kangaroo simulation is run to pull the 2D truss apart. Pulling the top truss elements towards the compression mesh, the bottom elements to a straight tension surface and applying spring forces to the grouped elements to control the outcome of the simulation.

As you can see in the second and third image on this page and first and second image on the next, the output truss still contains a grouped structure of the elements. Similarly the origional mesh is deformed to fit the new space truss shape and is also grouped as a seperate output. This is useful later on when setting the Karamba simulation.

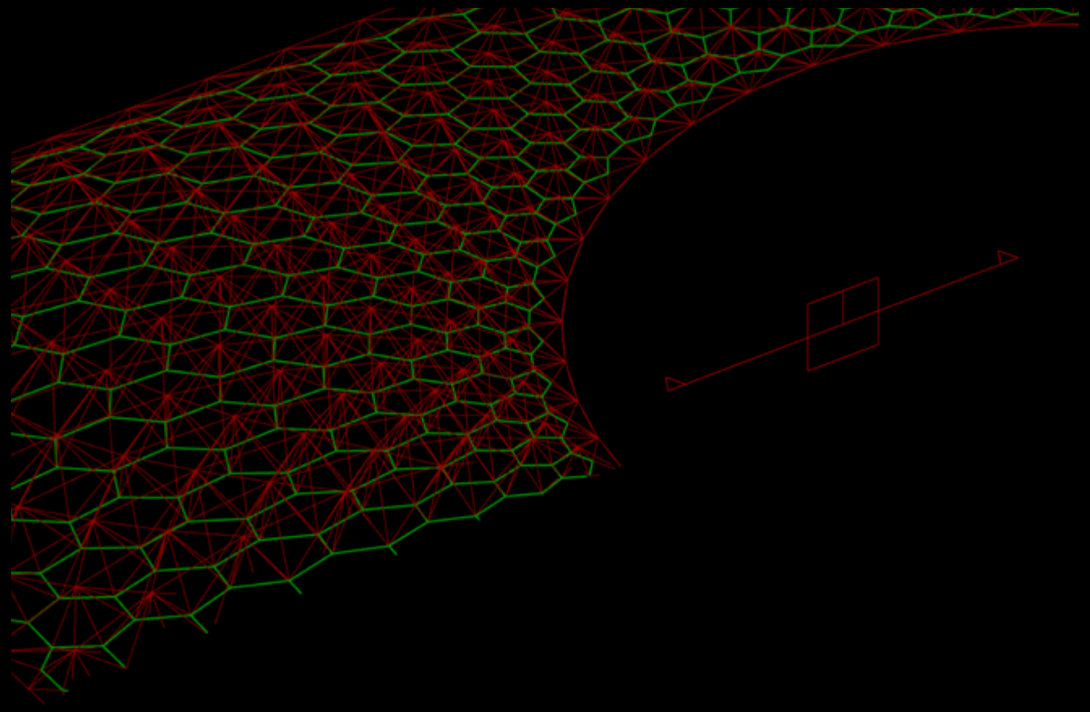


Figure 39:Lower truss elements are grouped (authors).

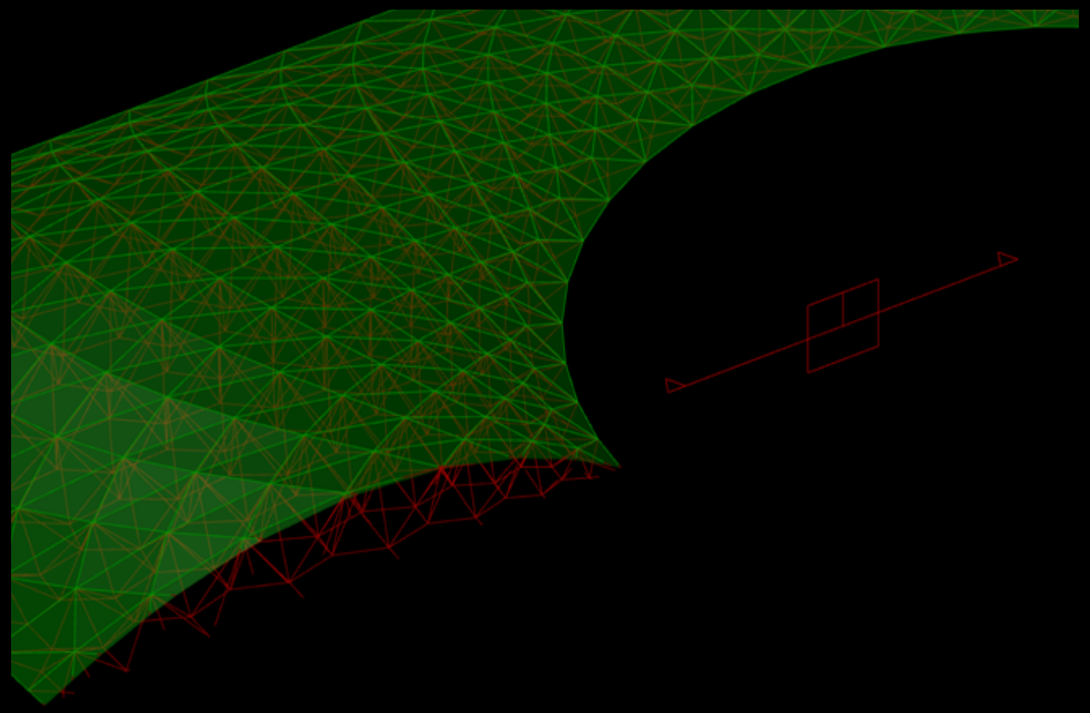


Figure 40:Deformed mesh is also derived. Needed for area and force calculations (authors).

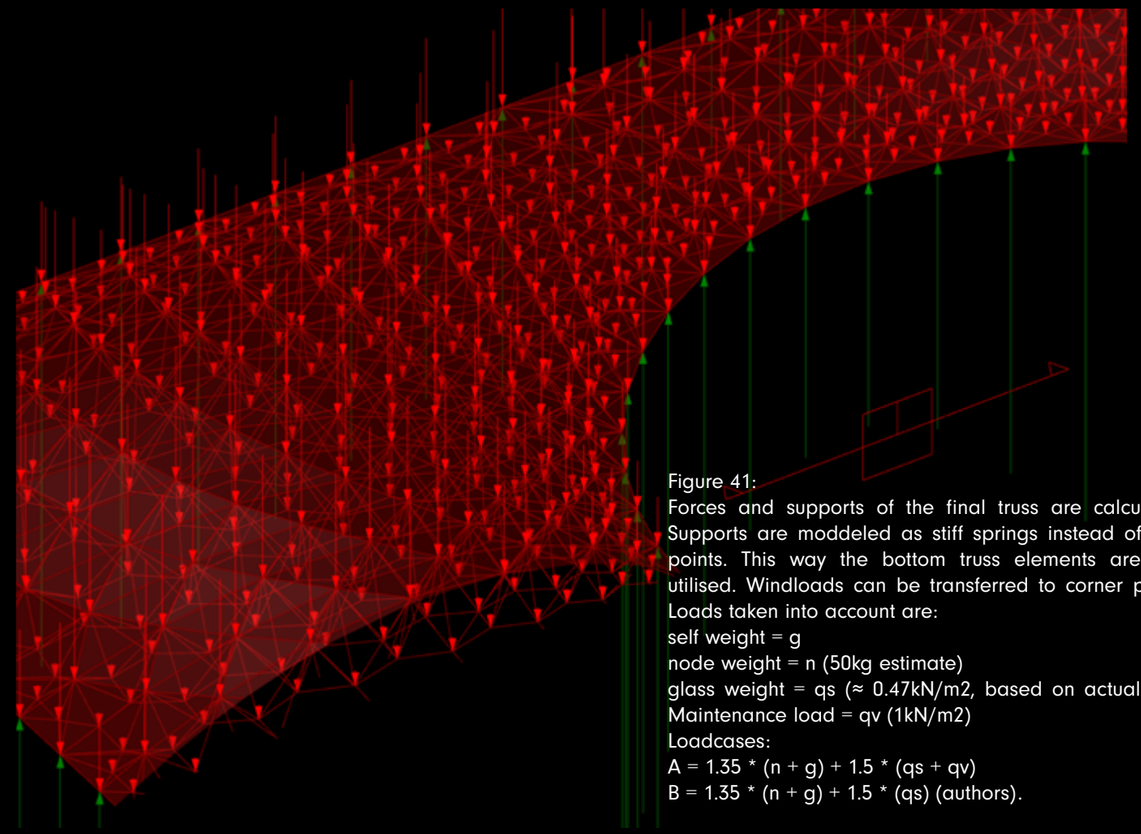


Figure 41:
Forces and supports of the final truss are calculated. Supports are moddeled as stiff springs instead of rigid points. This way the bottom truss elements are also utilised. Windloads can be transferred to corner points.
Loads taken into account are:
self weight = g
node weight = n (50kg estimate)
glass weight = qs (≈ 0.47kN/m2, based on actual roof)
Maintenance load = qv (1kN/m2)
Loadcases:
A = 1.35 * (n + g) + 1.5 * (qs + qv)
B = 1.35 * (n + g) + 1.5 * (qs) (authors).

C3.Calculating loads and setting up the supports

With all of the 3D space truss grouped and identified, the next step involves refining the structural support system where the roof meets the surrounding context. This process includes the setting of load-bearing supports that anchor the roof to the existing structure of the courtyard.

Initially the supports were modelled as fully rigid supports, necessary for ensuring rigidity during assembly. However this resulted in some problems for the load paths. If the supports are not capable to slightly deform, due to the 'compression only' shape of the top truss elements, the bottom truss elements would not be utilized in tension. Instead, large horizontal forces on the walls of the existing structure are generated. This would likely result in cracks forming in the walls of the existing structure and is of course not allowed to happen.

With a lot of tests and simulations, the final supports are modelled in the following way. The 4 corners of the roof are modelled as spring supports connected to X,Y,Z-rigid pinned supports. 'X,Y,Z-rid' meaning the support point is rigid in both X,Y&Z and 'pinned', meaning it doesn't transfer moment. They are connected with springs with variable resistance in X,Y translations as compared to a much stiffer Z resistance. This way both deformation can occur that allows for the utilization of the bottom truss elements in tension and the possibility to redirect wind loads from the roof to the reinforced corners of the structure. A strategy to deal with wind loads similar to that in the actual roof of the Great Court. The other 128 support points are modelled as spring supports connected to (X,Y)Z-rigid pinned supports, with a much lower spring resistance in X&Y direction as compared to the corners. Slight X&Y resistance is still mathematically required to ensure rigidity during assembly. This prevents the rotation of the roof around a single corner point early on in the assembly process, when only a single corner point is connected. However, the lower the X&Y resistance in these support points, the lower the horizontal forces on the walls of the existing structure, which is preferable.

With the supports finished, the loads are also modelled for the simulation to run accordingly. First, the self-weight of the structure and nodes is calculated. The self-weight (g) of the structure is determined by the Karamba algorithm itself once enabled and the node weight (n) is then calculated as an overestimated 20-30% of the total structures self-weight. The glass weight (qs) is calculated with the actual glass roof weight of 315 tons and the maintenance load (qv) is set at 1kN/m2, as snow loads (0,3kN/m2) are non-governing.

Then load cases are added that include safety factors for variable and deadloads. The final loads Karamba will take into account are as follows:
A = 1.35 * (n + g) + 1.5 * (qs + qv)
B = 1.35 * (n + g) + 1.5 * (qs)

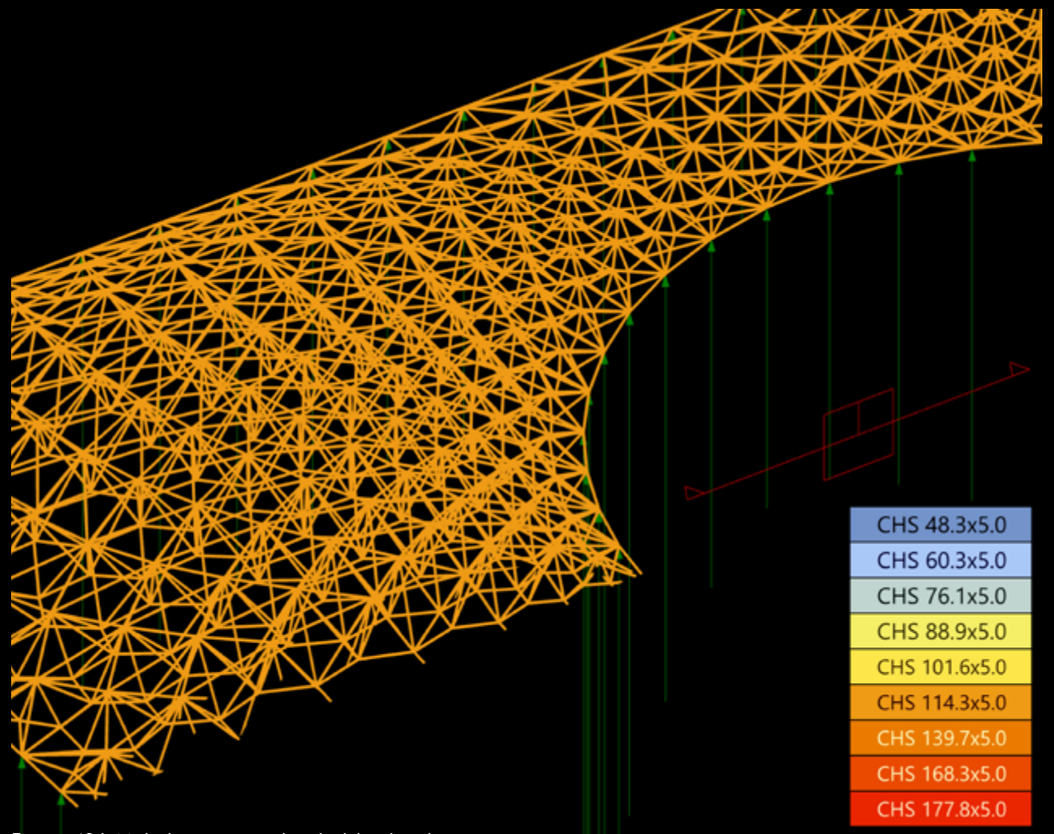


Figure 42:Initial elements are loaded (authors).

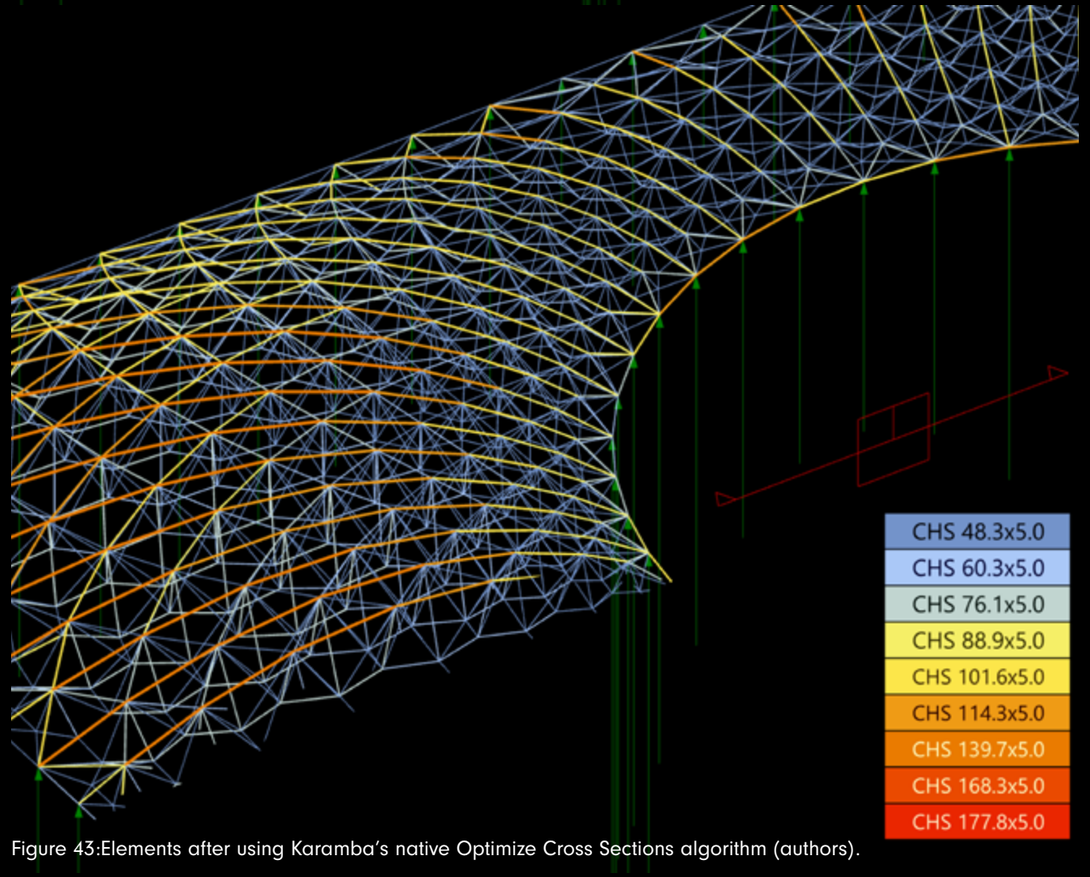


Figure 43:Elements after using Karamba's native Optimize Cross Sections algorithm (authors).

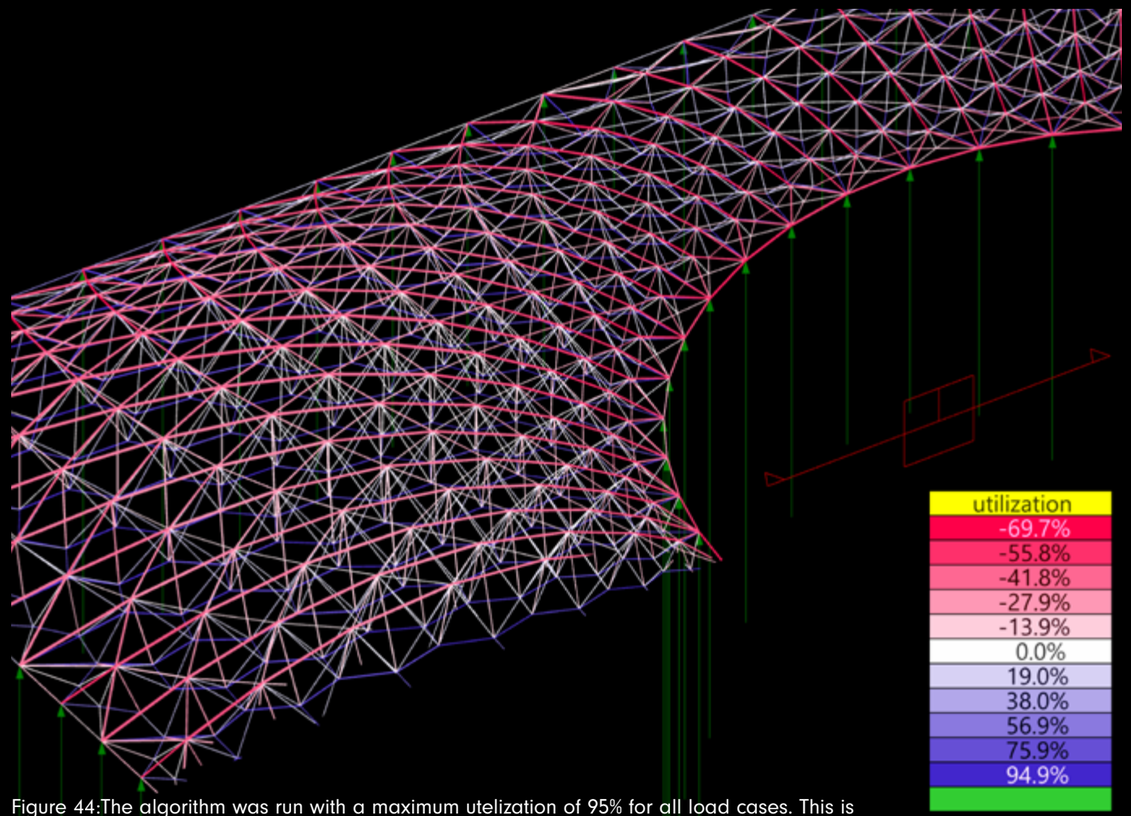


Figure 44:The algorithm was run with a maximum utelization of 95% for all load cases. This is including safety factors (authors).

C4.Element Optimization

With the appropriate geometry, loads, and supports determined, the next step is to perform Karamba simulations and optimizations.

Initially, the truss is modeled with identical cross-sections capable of handling the loads, but this setup is not yet optimized. In this state, the truss has a weight comparable to the actual roof of the Great Court. By using the Karamba Optimize Cross Sections algorithm and adjusting parameters such as allowable deflection and maximum utilization factors, a more efficient distribution of truss beams is identified. This improvement is visible in the second image. The optimized truss weighs approximately 283 tons, including the 50kg node weights, which is lighter compared to the 478 tons of steel used in the realized roof of the Great Court. Although this calculation does not yet account for wind or thermal loads, or many other load scenarios, it suggests that this solution could be a more efficient load-bearing structure.

In the third image, the utilization factors within the structure are displayed. The legend shows a maximum utilization of 95%, as set in the optimization algorithm. Interestingly, the top elements are larger but utilized less than the bottom elements, indicating that buckling likely governs the sizing of the top elements. This suggests that using cross-sections with larger diameters but thinner edges could result in an even more optimized structure.

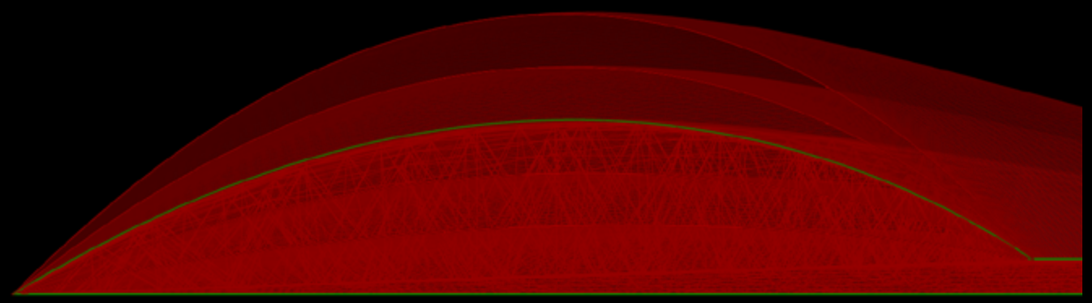


Figure 45:Finding the optimal shape based on final assembly weight after optimization. Buckling calculations included (authors).

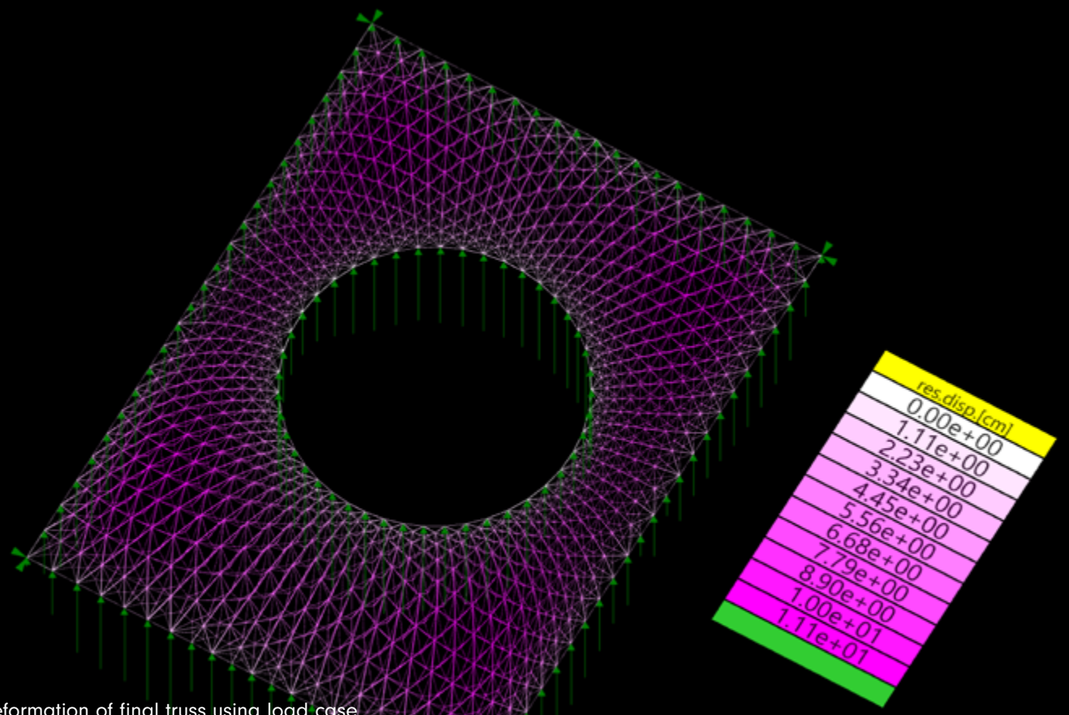


Figure 46:Deformation of final truss using load case
A = 1.35 * (n + g) + 1.5 * (qs + qv).
The deformation is within limits of span/240.
(41m/240 > 0,11m)
Horizontal deformation at supports is max 3,5cm.
Variable displacement at 1,9cm due to maintenance load.
(authors).

C5.Shape Optimisation & Final Design

The next step involves the exploration of a variety of geometric configurations by adjusting the input parameters for the compression domes. As a reminder, the compression domes serve as the form-finding basis for the structure, though it is still unknown if this shape is truly optimal.To identify the best shape, multiple compression domes are simulated, and their corresponding trusses are generated. These trusses undergo the full range of Karamba optimization steps, and their weights are recorded. From the various input domes, the most efficient one is selected.

The following images display the deformation, element lengths, and types of elements used throughout the entire roof structure.As shown, the deformation is tested and remains within the typical limits, often determined by the span/240 rule. The horizontal deformation at the supports is measured at 3.5 cm, with a variable displacement of 1.9 cm due to maintenance loads. These deformations are important considerations for the design of the load-bearing supports, ensuring the structure can accommodate these movements while maintaining stability. That being said they should be manageable.

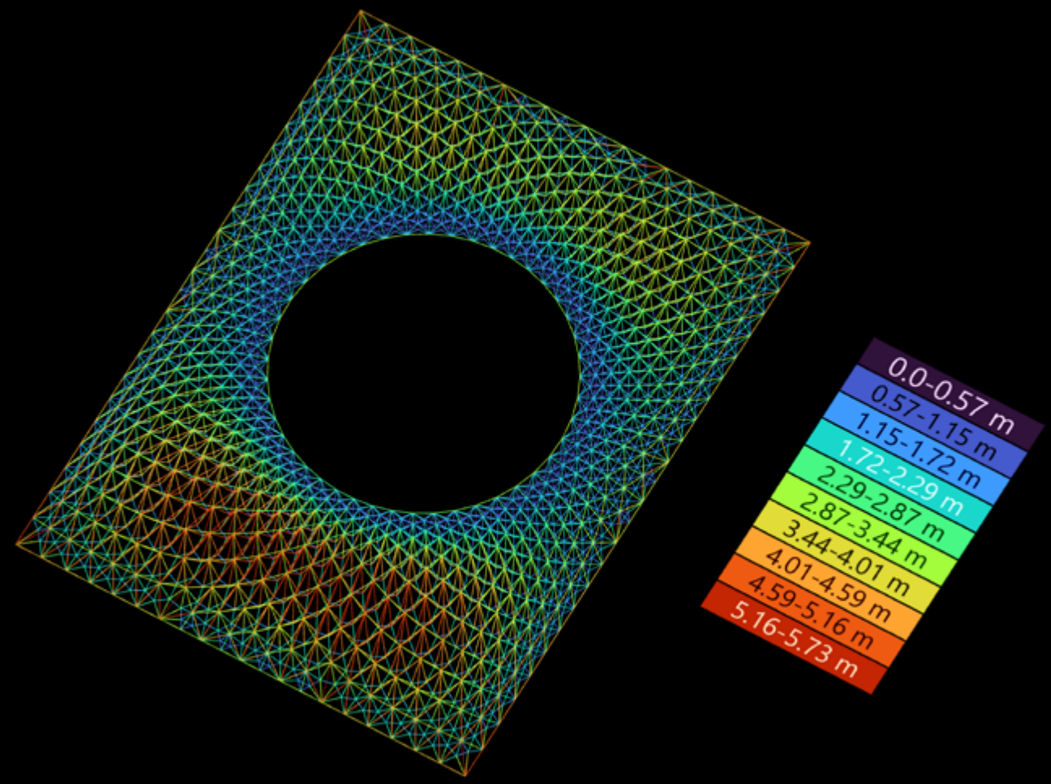


Figure 47:Lenghts of final truss elements (authors).

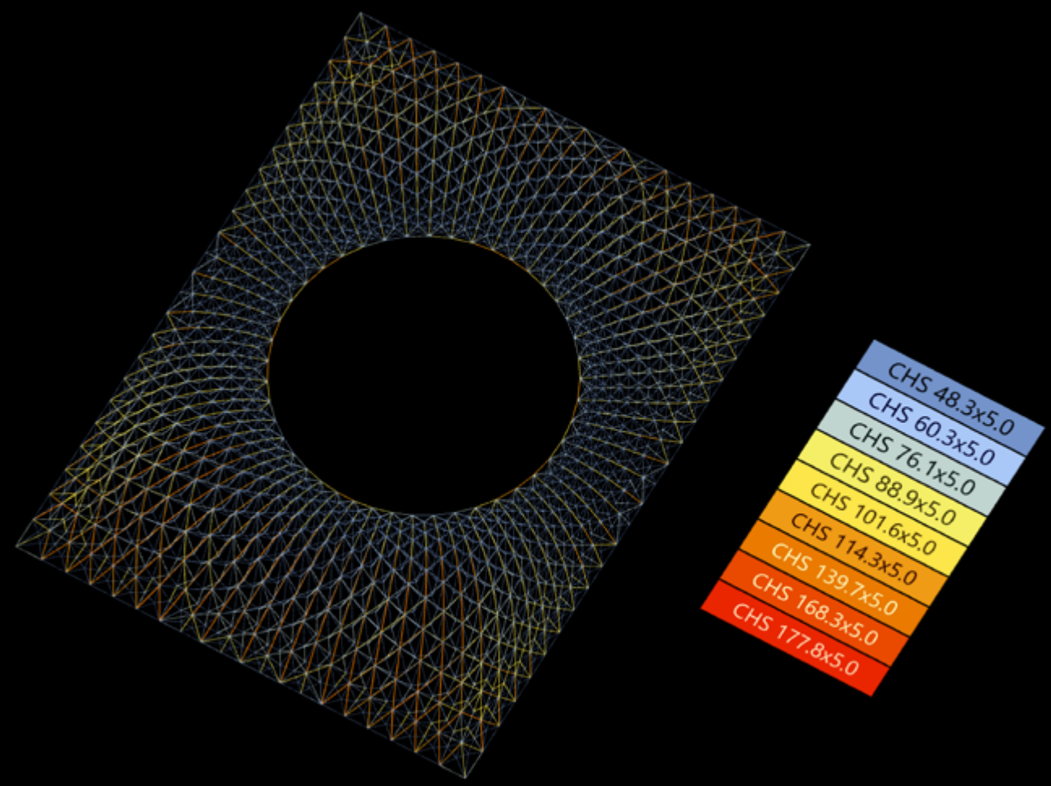


Figure 48:Types of final truss elements (authors).

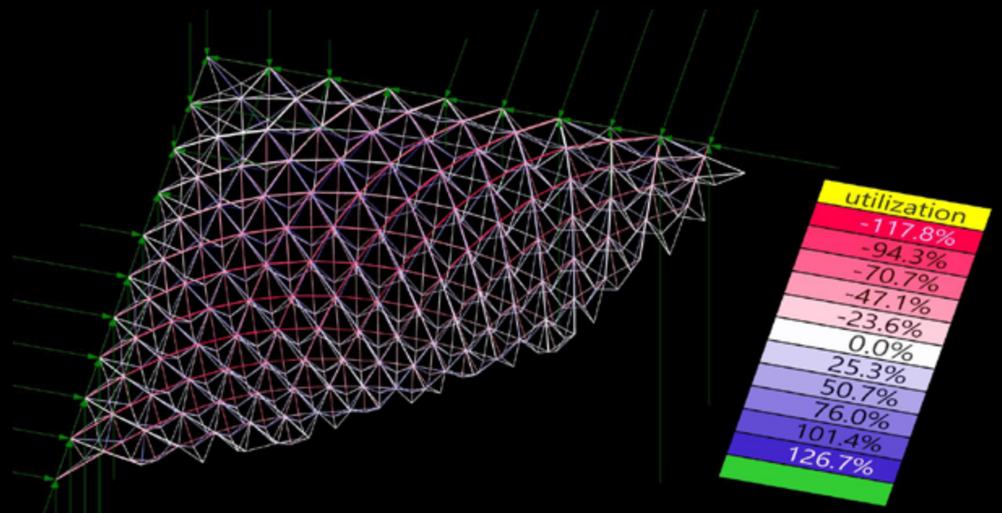


Figure 49:Axial loads during assembly without additional crane support.
Load case = 1.35 * (n + g) (authors).

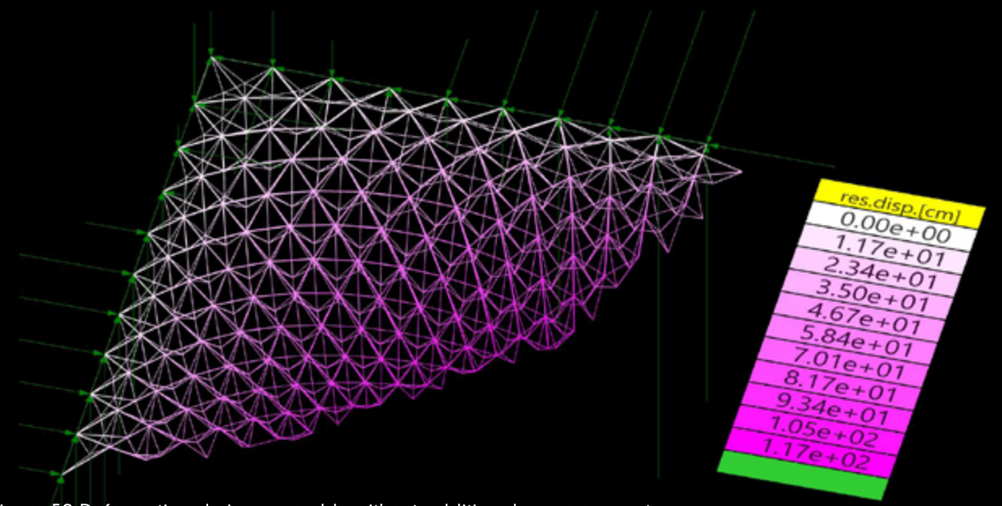


Figure 50:Deformation during assembly without additional crane support.
Load case = 1.35 * (n + g) (authors).

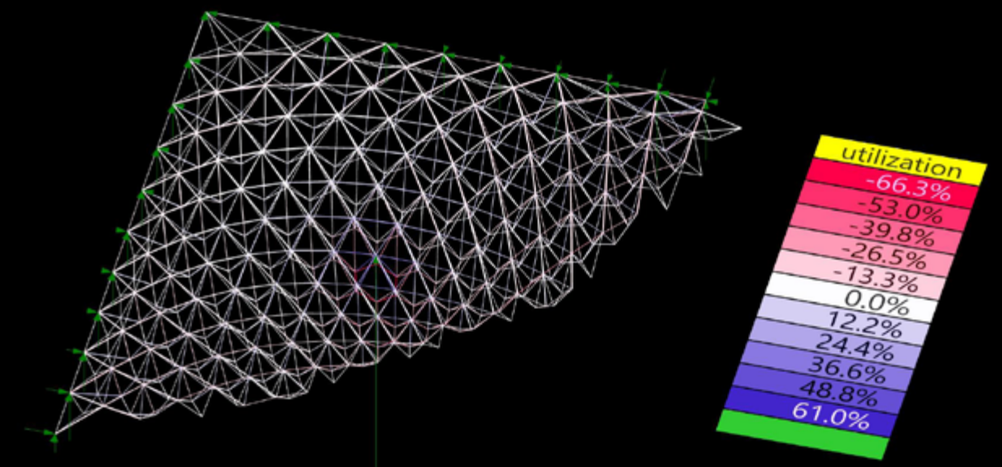


Figure 51:Axial loads during assembly with additional crane support.
Load case = 1.35 * (n + g) (authors).

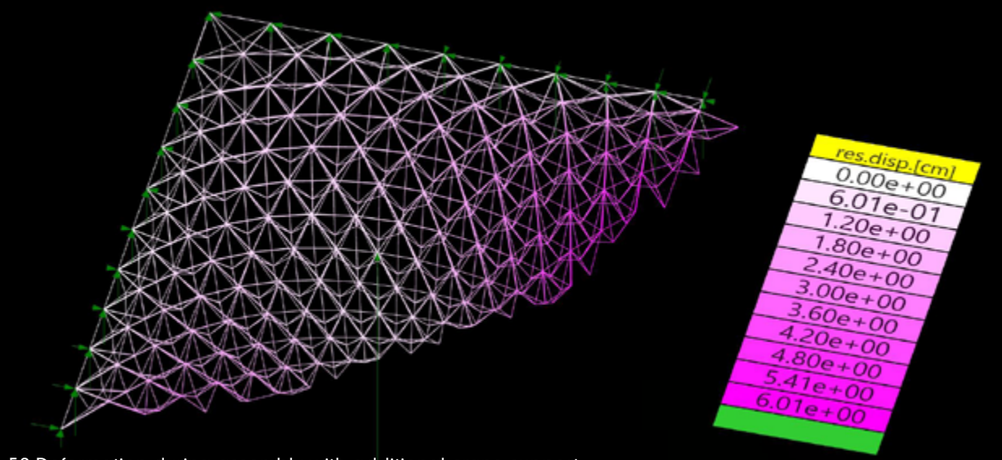


Figure 52:Deformation during assembly with additional crane support.
Load case = 1.35 * (n + g) (authors).

C6.Assembly Sequence Validation

Once the truss is defined and optimized, it must be validated to ensure rigidity during the assembly process. At this stage, only self-weight, node weight, and dead load safety factors need to be considered, as glass and maintenance loads are not yet relevant. However, localized stresses may still occur during assembly due to the interconnected nature of the structure and support placement.

To address this, the Karamba model can be deconstructed, then reconstructed into a model for each assembly step N of the assembly sequence algorithm. For each step, the model undergoes a new Karamba simulation to calculate deformation, utilization, axial forces, and the deformation of elements to be placed next. More on this element deformation later.

Running the simulation without additional supports will result in the collapse of the structure, as utilization factors and deformations exceed acceptable limits when a too large cantilever starts to form. To prevent this, temporary supports have to be introduced.
Due to time constraints, the process of identifying optimal support locations is still underdeveloped. Currently, the method involves three steps:

1. Initial Simulation Without Crane Supports: Run the entire simulation without crane supports. For each step N, identify areas of large deformation and the locations of the most recently added edge.

2. Threshold Analysis: Record the following values at each step—maximum compressive force, maximum tensile force, maximum compressive deformation, maximum tensile deformation, minimum buckling factor, maximum utilization, and maximum global deformation. Set thresholds for each of these values and identify the steps N where these limits are exceeded.

3. Resimulation with Crane Supports: Resimulate the critical steps identified, adding crane supports at the initially determined positions. If any values still exceed thresholds, the process is repeated and more cranes are placed until all values fall within acceptable limits.This iterative approach provides a practical, if not yet fully working, solution for ensuring stability throughout the assembly process.

There are several areas where the support placement script could be improved. Currently, the script lacks continuity in support placement and struggles to accurately identify locations of high deformation. Furthermore, if a single support is insufficient, it does not automatically adjust the origional support positioning when adding additional supports.As some steps still returned utilization factors above 1, particularly during the initial bridging phase. A manual attempt was made to address this by splitting each crane into two or even three cranes distributed along tweened curves, but this approach was manual and did not completely resolve the issue.

Future solutions could involve developing a more effective algorithm for initial support placement, using larger profiles where utilization spikes (as only a few elements need reinforcement), or adding supports at critical locations rather than high-deformation areas, all the while maintaining continuity to minimize repositioning during assembly. And ofcourse, further adjustments to the assembly sequence could be considered.

Specifically, identifying areas of high deformation could be improved using 4D vector k-means clustering, where the first three dimensions represent the spatial coordinates of each node, and the fourth dimension corresponds to the deformation at that point. This approach would likely allow for better identification of high-deformation areas compared to the current script, which uses a manually set deformation limit and averaging the points that surpass it.

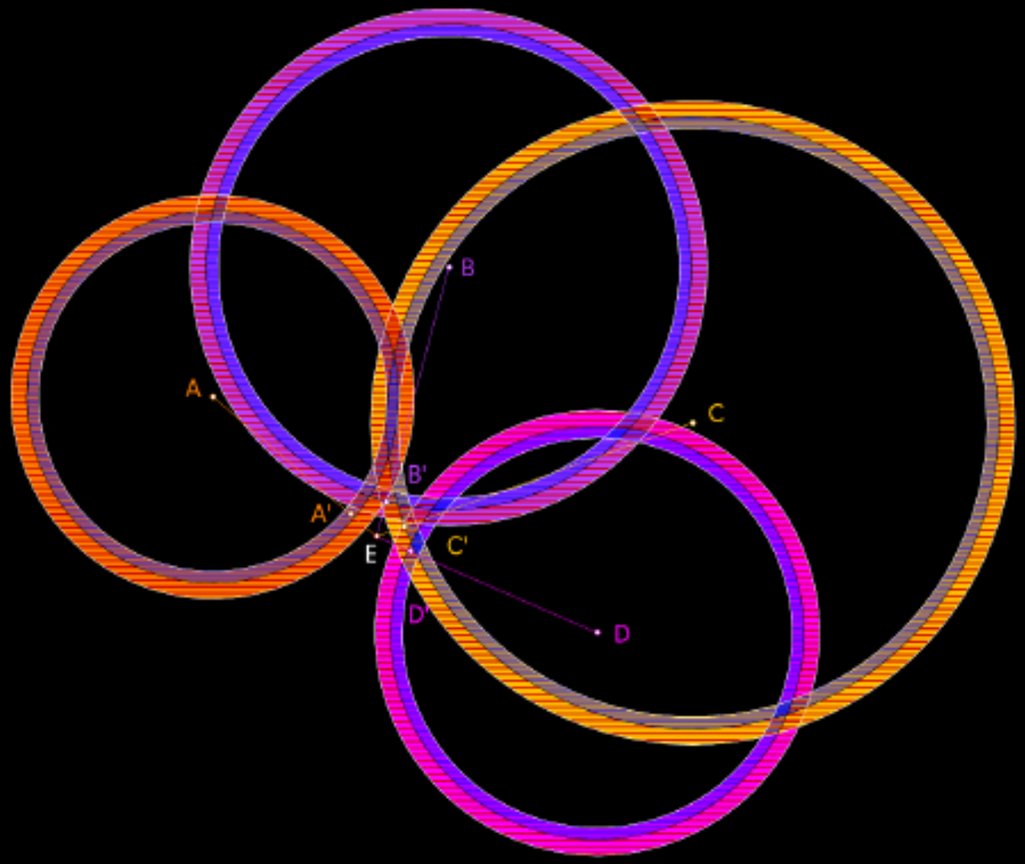


Figure 54: The geometric challenge for the connections (authors).

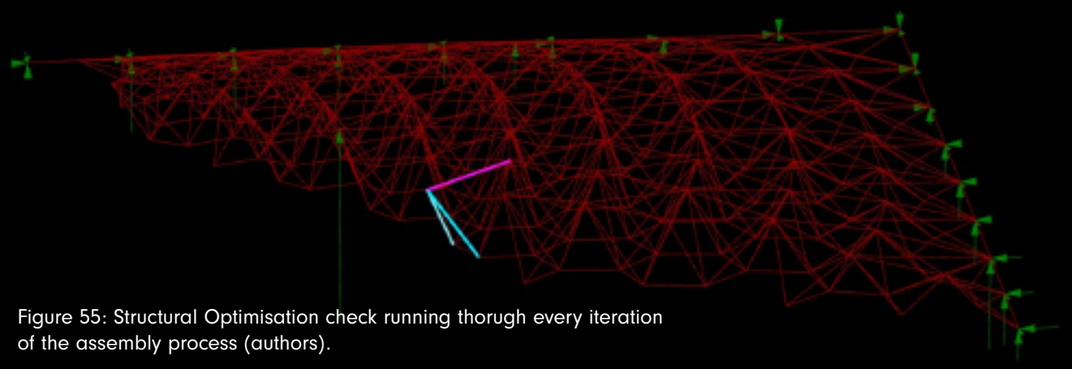


Figure 55: Structural Optimisation check running thorugh every iteration of the assembly process (authors).

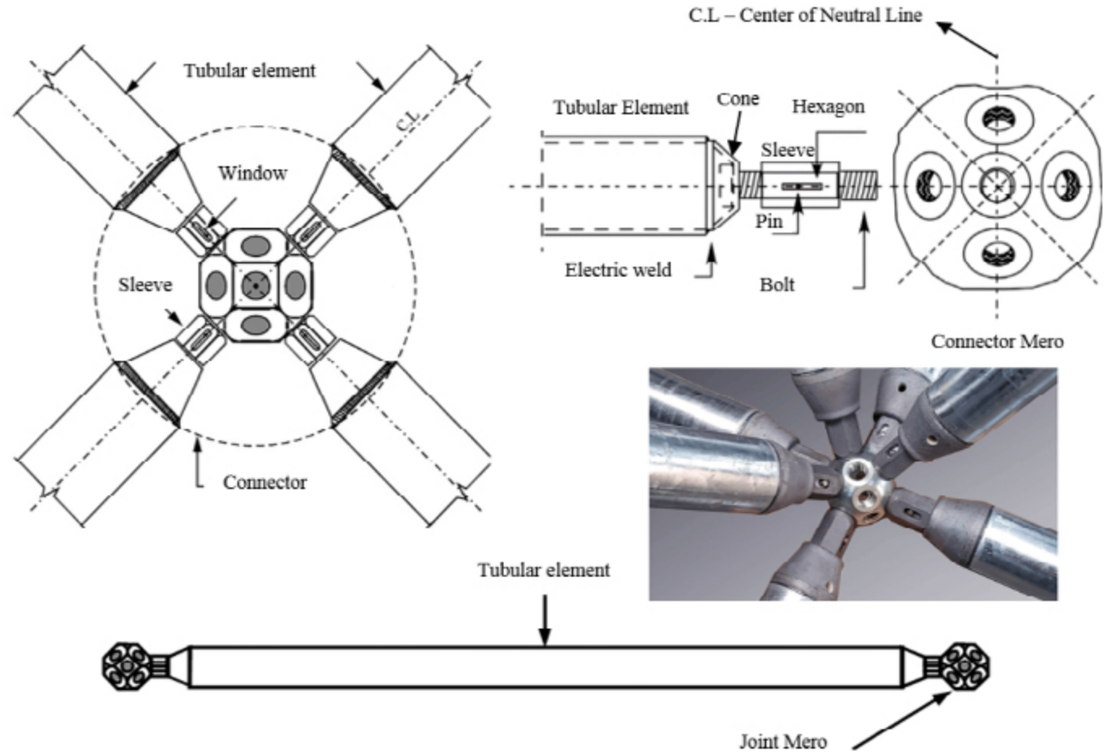


Figure 55:Connection manufactured in 1943, the system more widely used in space trusses in the world. source: https://www.mdpi.com/1996-1944/13/10/2305

C7. Connection Design

Space frames have been constructed for decades, and the detailing of their connecting joints is something most of us recognize, as seen in the image above. Initially, we planned to reuse the existing connection detailing for one of these joints. However, this presented a few challenges. The primary question was: how do we connect the beams to the node? Traditional space frame connections typically use a slotted bolt, passed through a cone, which can be turned externally with a hexagonal sleeve and then secured to a threaded Mero joint. This method is reliable in most cases, but it poses challenges in robotic assembly.

The issue lies in the fact that the bolts protrude slightly beyond the final connected length of the edge element. This requires some movement of the Mero joint for the bolt to engage, which necessitates a degree of deformation. Calculating this deformation for each node in an assembled and loaded space frame is difficult, and it can sometimes be unachievable if the joint is already part of a rigid structure. A robotic arm simply wouldn't have the force needed to adjust the node's position.

This geometric challenge is better illustrated in the diagram on the top left. Nodes A, B, C, and D are already part of the rigid structure and cannot be moved easily. During assembly, internal deformations mean that points A', B', C', and D' no longer align precisely with their intended connections on the Mero joint at E. Each edge element must therefore accommodate these small positive and negative deformations to fit properly.

Simulating the structure enabled us to determine the exact tolerances needed for assembly. In theory, all that's required is the length of the elements prior to deformation and the actual gaps between points A-E, B-E, C-E, and D-E when fitting the edge elements. The difference between these element lengths and the measured gaps defines the necessary tolerances. However, identifying these gaps is more challenging than it sounds.To calculate the gaps, you first need the precise locations of points A, B, C, and D, as these define the potential positions for point E. Calculating this solution space is highly computationally intensive, and it becomes impractical when connecting to rigid edge points where point E's location is already fixed.

As a workaround, an estimated location for point E is sometimes assumed based on plausible positioning within 3D space. Ofcourse for the cases involving rigid edges or single-edge connections—where all point positions can be defined using Karamba deformation simulations—the exact locations of the points are used. However, when connecting three new edges to an undefined point in 3D space, point E's position is approximated by averaging the deformations of the three connecting points.

Simulating each of the 4,239 assembly steps allowed us to determine the required tolerances for a successful assembly. However, due to time constraints and an error during one of the final full-day simulations, only 1,100 steps were fully calculated. Nonetheless, this sample should still provide an accurate reflection of the normal distributions, and the results are expected to be consistent with those from a complete simulation.

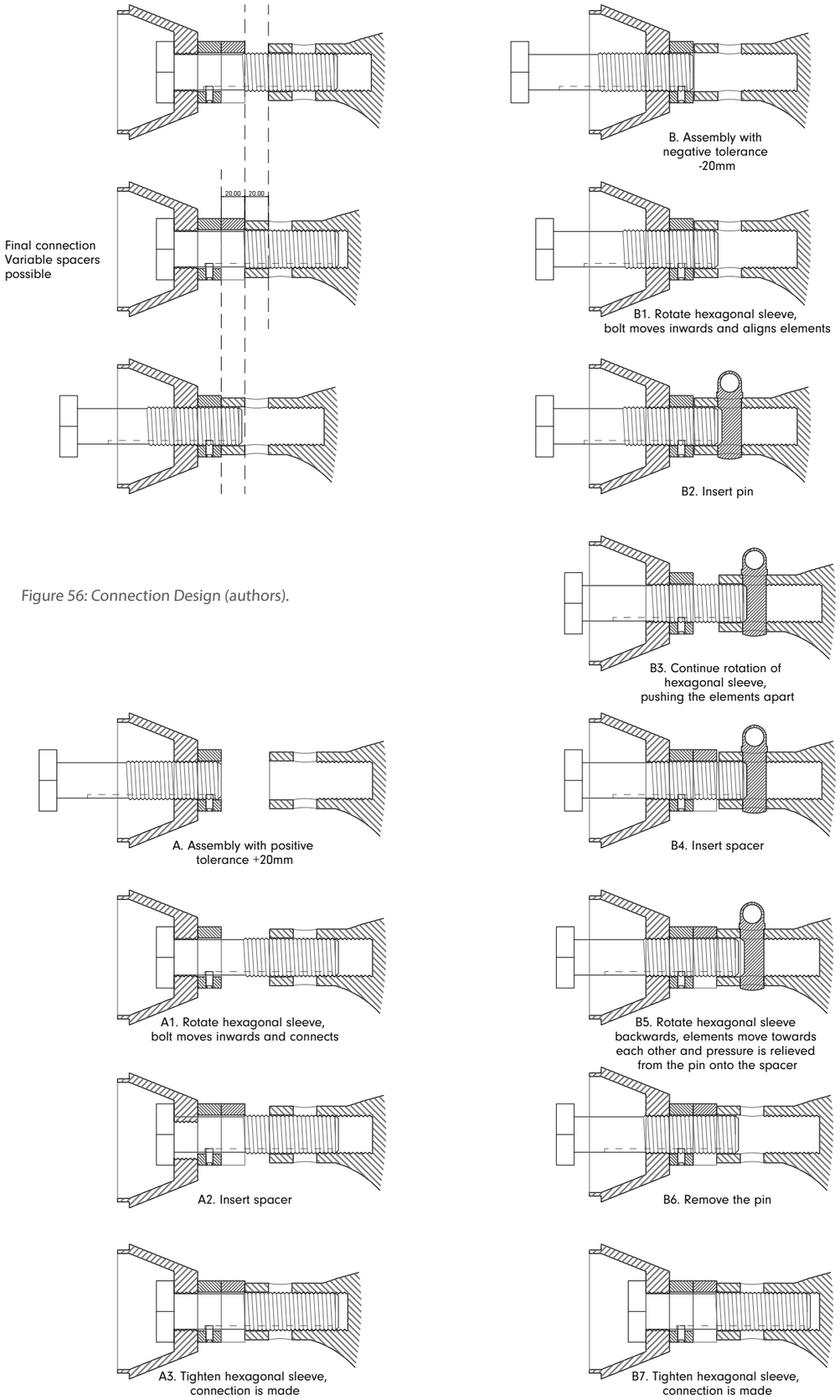
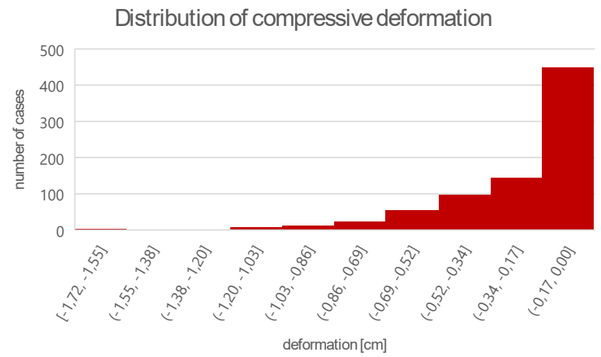
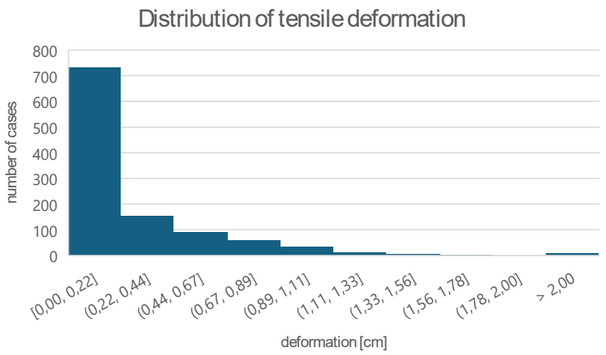
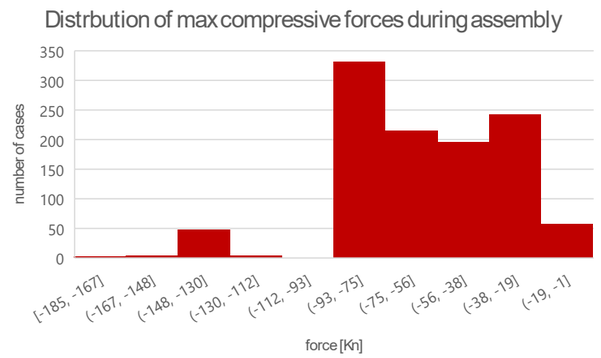


Figure 56: Connection Design (authors).







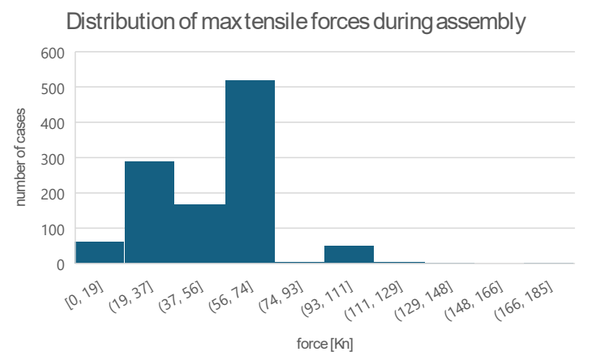


Figure 57: Forces & Deformation (authors).

C7.Connection Design

With the tolerances calculated, the connection elements can now be detailed. They remain largely similar to existing solutions, with a few key modifications.

First, the cone connecting the steel profile to the bolt is now threaded, allowing the bolt to retract fully, flush with the hexagonal sleeve. Additionally, a spacer element has been added to accommodate compressive or "negative" tolerances. The Mero or node part of the connection is also modified, with a smooth surface for the first 20mm and holes for a temporary pin support. This smooth section is essential, as it prevents the node from rotating or damaging the threads when the elements are pushed apart.

The bolt itself now has threads strategically positioned to engage only where necessary, allowing for free rotation or connection at various stages of assembly. Notably, the smooth portion of the bolt matches the inner diameter of the threads, rather than the outer diameter—a departure from standard industry solutions. This adjustment is necessary to ensure that the element can be fully tightened now that the cone is threaded.

On the left, a diagram illustrates each step in the new connection process. Although there are multiple stages, they flow seamlessly into one another, making the connection sequence straightforward for users in practice.

The approximate sizing of the structural elements was calculated based on both final load cases and assembly loads, using Eurocodes wherever applicable. The following designs choices were considered:

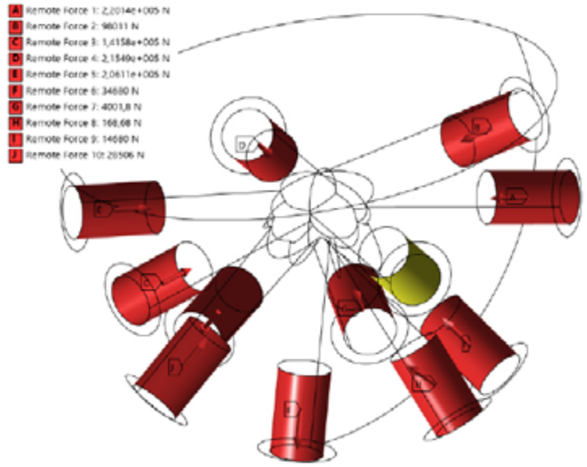
Bolts:
> Type: M30 grade 10.9
> Requirement: Axial resistance as per Eurocode standards for a final load case of approximately 340 kN

Thread engagement lengths at node and cone:
> Material: S275 steel
> Requirement: Based on shear resistance calculations for cones and node connections for compressive and tensile loads during assembly

Temporary Pin:
> Diameter: 20mm
> Material: Grade 8.8 steel
> Requirement: Calculated based on double shear resistance during the maximum compressive loads in the assembly sequence

Pipe Wall Thickness at Node Attachment:
> Thicness: 9mm
> Material: S275 plate steel
> Requirement: Determined by bearing resistance formulas to handle loads transferred by the temporary pin during the maximum compressive loads in the assembly sequence

These details will still need to be verified through finite element analysis, but they provided a solid foundation for designing the connection.



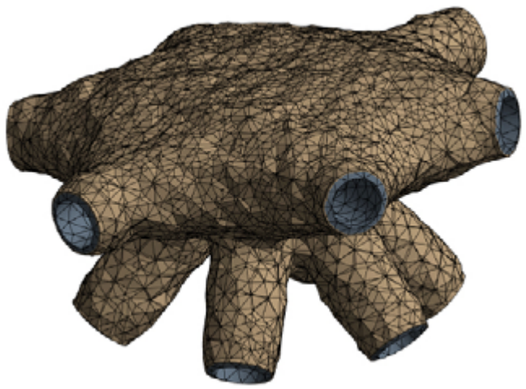
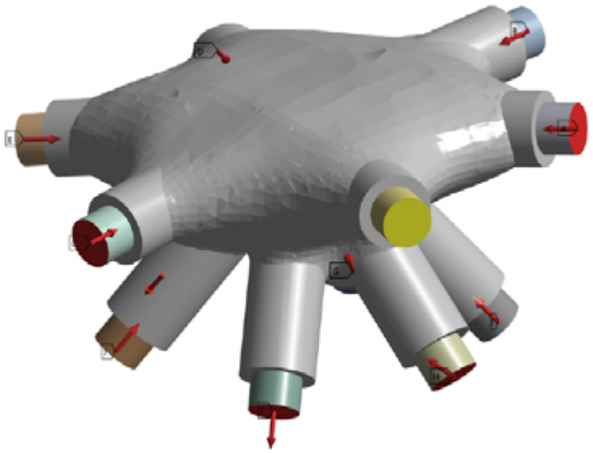


Figure 58: Node Optimization with Ansys(authors).



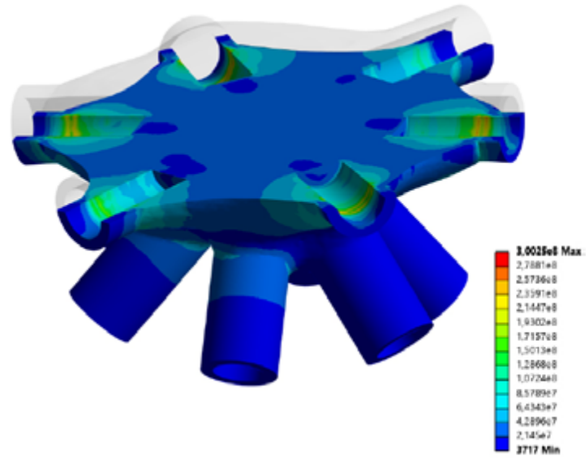


Figure 59: Force Distribution(authors).

C8.Node Design & Optimisation

To minimize the overall weight of the structure, it's crucial to optimize both the individual elements and the nodes. This is demonstrated when using the Karamba Optimize Cross Section component, as doubling the node weight also requires bigger cross-sections, thereby further increasing the total weight. For instance, with nodes weighing 50 kg each, the structure's weight would be approximately 283 tonnes. However, if the node weight increases to 100 kg, the structure's weight rises to around 397 tonnes—an increase of 114 tonnes, composed of 109 tonnes from additional node weight and 5 tonnes from bigger tubular sections.

For the structure analyzed in this report, an estimated 50 kg per node weight was used in the simulation, which required optimizations to the node design to achieve this target.

We selected Ansys Workbench 2024 for this optimization, as it includes built-in topology optimization tools.

To begin, we selected an arbitrarily chosen but common top node under relatively high compressive load, located near the corner of the structure. Using the forces from each connected element provided by the Grasshopper Karamba script, we could analyze load distribution. The Karamba script also accounts for bending forces in the top elements, ensuring accurate buckling calculations and load paths from the glass onto the beams, meaning shear forces from these elements also had to be considered. Summing all these forces enabled us to verify if they were correct. Checking if they resulted in a vector of [0,0,675], indicating a resultant force equal to the node's weight.

675 = 50 kg * 10 N/kg * 1.35 (safey factor)

In Ansys, topology optimization removes material in areas that experience low stress, meaning the input geometry needs to be larger than the expected final shape to fully encompass the optimized design. For this purpose, a spherical volume was created with cylindrical cutouts where the bolt connections are positioned. These connection areas were defined as faces rigidly connected to remote points, allowing the forces calculated earlier to be applied as remote forces at these remote points, ideally replicating real-world conditions as deformations would be severely limited when a high strength bolt is connected inside these hollowed out sections.

In the simulation, one of the forces had to be modeled as a support, as a structure without supports cannot be calculated. To ensure that everything was applied correctly, a reaction force probe was added to the remote support point. By confirming that the force vectors matched those that would otherwise be applied, we could verify that the Ansys model was set up correctly.

However, during initial simulations, an issue arose: due to the low forces on two of the connecting remote points, the optimizer did not generate any connecting geometry from the center of the node to those points. Since Ansys lacks an option to retain a minimum volume at certain places, an alternative approach was needed.

Fortunately, Ansys does allow certain surfaces to be retained during optimization. To address the issue, the tubular attachment points were extended to the center, forming an initial hollow structure. This you can more clearly see in the first image. The hollow tube sections could later be filled when conducting the final force checks, ensuring structural connectivity in the optimized design.

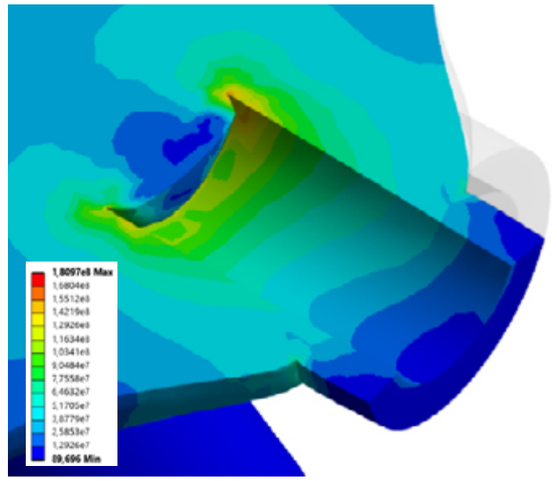
With the model set up, a topology optimization was performed to retain 25% of the initial volume—an amount calculated to target a node weight of approximately 40 kg. After exporting the optimized mesh back into Rhino, significant manual cleaning and smoothing were required to reconstruct a volume suitable for further simulation.

Initially, the model was set up with similar rigidly connected remote points, but analysis showed significant stress concentrations at the corners of the rigid tubular faces, as these were set unable to deform. In reality, these areas would undergo axial deformation along with the connecting bolts. To address this, the final setup included directly modeling the bolts and applying bonded connections between the outer bolt faces and the inner surfaces of the tubular attachment points. The bolts were sectioned and remote forces were applied to these section faces. This approach provided a more accurate force distribution across the connecting faces, giving a clearer picture of where stresses would emerge. You can clearly see this in the two images on the right.

Since the node will be 3D printed, we chose 316L steel for its compatibility with additive manufacturing.

However, a primary concern with using 316L steel is its variability in Young's modulus and yield strength, which can fluctuate based on treatment. This variability raises questions about the potential for plastic deformation in the bolt threads. With normal values for the elastic yield strengths of 316L steel the starting threads would plastically defoform during high maintenance loads. Perhaps high strength inserts could solve this issue. Additional investigation is therefore needed to evaluate the threads' behavior and ensure that the node will maintain structural integrity and reversability under operational conditions.

After completing all calculations, the total weight of each node, including bolts, hexagonal sleeves, spacers, and cones, came to approximately 48 kg. While this exceeds the initial 40 kg target, it still allows us to use the estimated 50 kg per node for this project. The additional weight primarily resulted from the required sizing of the tubular sections of the nodes for the assembly sequence.



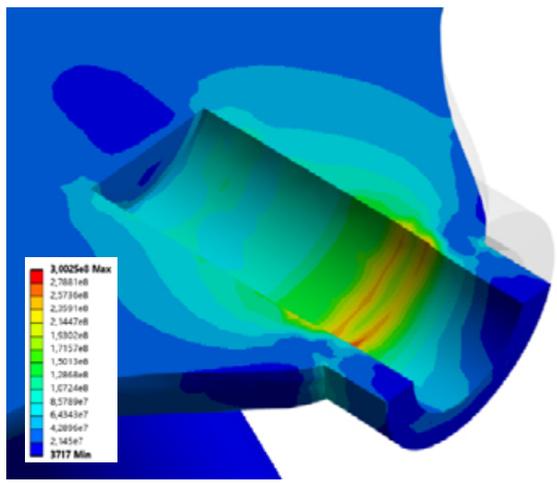


Figure 60: Node Section(authors).

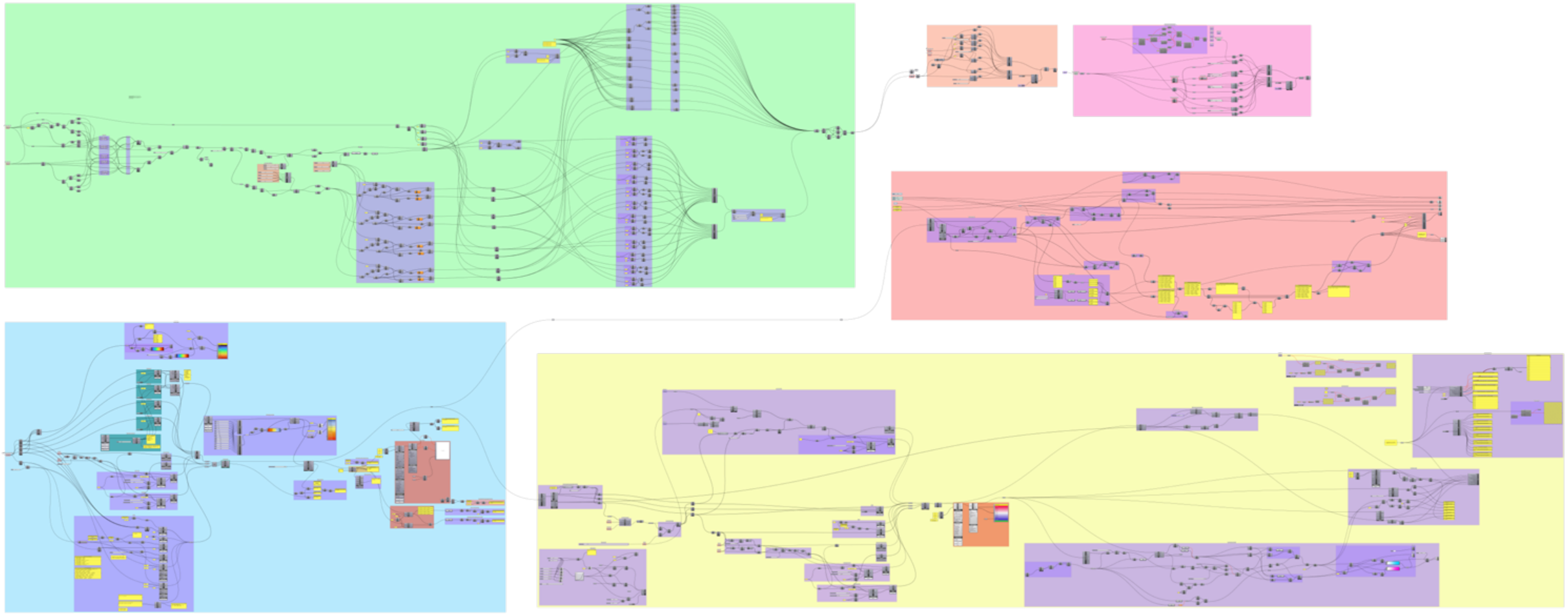


Figure 61: Grasshopper script for structural optimisation(authors).

C9.Script & Package Overview

The geometry generation and simulations for this project required a few additional Grasshopper packages.

First, the Panelling Tools package was used to define a mesh pattern aligned with the design of the Great Court. Next, the Kangaroo package provided dynamic relaxation components, which offered a fast and user-friendly way to handle geometric adjustments. The most used package, however, was the Karamba structural calculation package, chosen for its powerful, accessible parametric structural analysis capabilities, enabling efficient setup and optimization of the large-scale space frame. Throughout the script, numerous custom Python components were employed to streamline modeling, as certain calculations are more efficiently handled in Python. Most components serve straightforward purposes, like identifying naked mesh edges, adjusting data structures, or managing CSV imports and exports. However, more complex scripts were also created, such as the assembly sequence algorithm, which required installing numpy, scipy, and networkx to leverage additional functionality and significantly enhance performance.

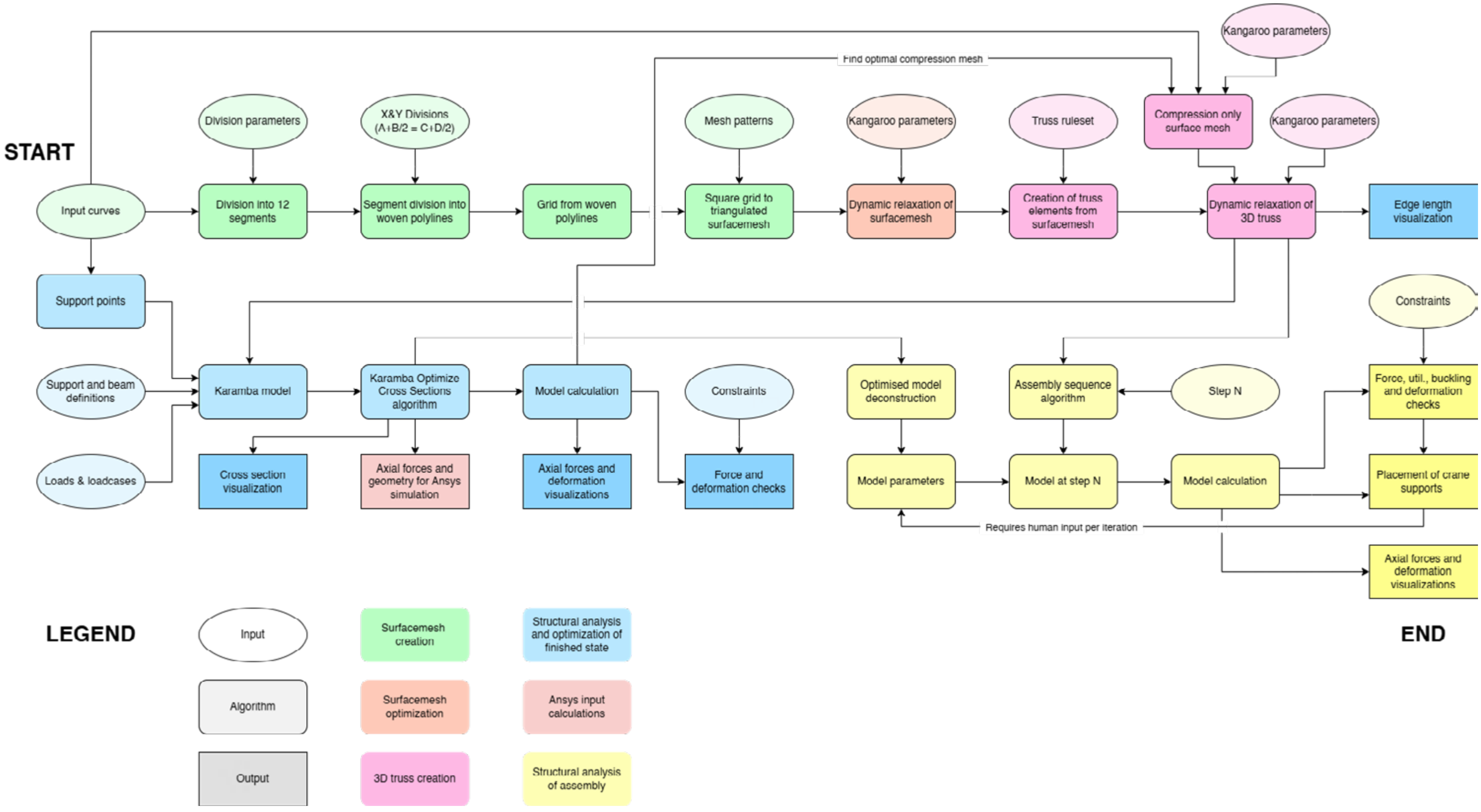


Figure 62: Flowchart explaing structural redesign and optimisation(authors).

C9.Script & Package Overview

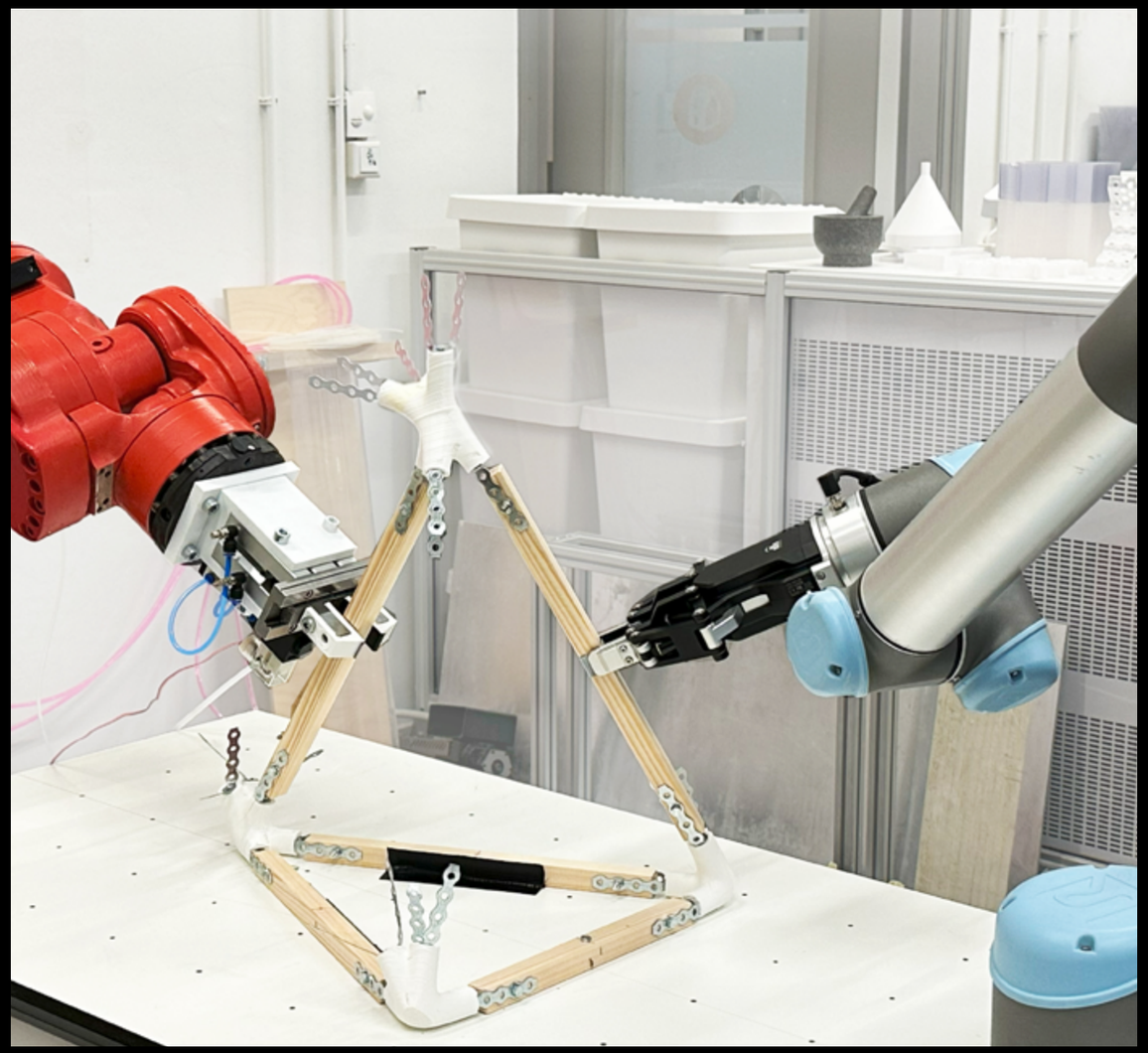
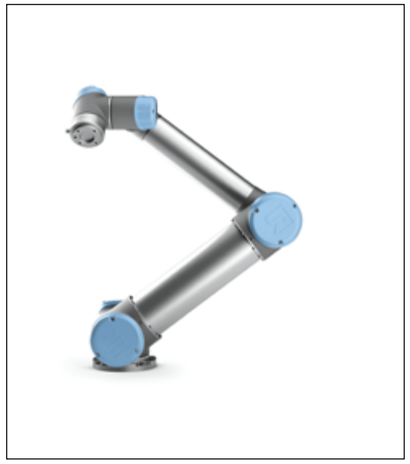


Figure 63: Experimentation with the UR5 and COMAU(authors).

D.MULTI-ROBOT ASSEMBLY



Characteristic	**UR5**	**Comau NJ 60-2.2**
DOF	6 revolute joints	6 revolute joints
Joint Axes	Base (z), Shoulder (y), Elbow (y), Wrist1 (z), Wrist2 (y), Wrist3 (z)	Base (z), Shoulder (y), Elbow (y), Wrist (z), Pitch (y), Roll (z)
Reach	850 mm	2200 mm
Payload	5 kg	60 kg
Link Lengths	a2 = 425 mm, a3 = 392.2 mm	a2 = 600 mm, a3 = 1200 mm
Link Offsets (d)	d1 = 162.5 mm, d4 = 133.3 mm, d5 = 99.7 mm, d6 = 99.6 mm	d1 = 740 mm, d4 = 1350 mm, d6 = 260 mm
Max Joint Rotation	J1: ±360°, J2: ±180°, J3: ±180°, J4: ±360°, J5: ±360°, J6: ±360°	J1: ±185°, J2: ±110°, J3: ±140°, J4: ±185°, J5: ±120°, J6: ±400°
Kinematic Chain	Open serial chain	Open serial chain
DH Parameters	Standard DH, α: 0, $\pi/2$, $-\pi/2$	Standard DH, α: 0, $\pi/2$, $-\pi/2$
Application	Collaborative, lightweight	Heavy-duty, industrial

Figure 64:The robots' characteristics(authors).

D1.The UR5 & The COMAU NJ 60-2.2

UR5: Collaborative Precision and Flexibility
The UR5, developed by Universal Robots, is known for its collaborative nature. Designed to work safely alongside humans, it features force sensors and safety systems that allow it to stop immediately if it detects any obstacles. This makes the UR5 well-suited for tasks requiring proximity to human operators or real-time adjustments.

One of the UR5's main strengths is its precision and flexibility. With six degrees of freedom, it can make fine, controlled movements, which is essential for tasks requiring small adjustments. In our setup, the UR5 will help align components that are being held in place, ensuring they are positioned for the human operator to make final connections. Its user-friendly interface also makes it highly adaptable for different tasks.

However, there are a few things to consider with the UR5:
- Its collaborative design means it operates at lower speeds to ensure safety. While it excels in precision near humans, it may not be as fast when handling larger-scale movements.
- The UR5 has a limited payload capacity of 5 kg. Although this won't be a problem in our setup where components have similar weights, it is something to keep in mind if future tasks require lifting heavier items.

Comau NJ 60-2.2: Strength and Stability
The Comau NJ 60-2.2 is an industrial robot designed for strength and durability in heavy-duty applications. It has a long reach and high payload capacity, making it ideal for tasks that require stability and precise control. While the components in our scenario are not particularly large or heavy, the Comau's ability to securely hold parts in place will be crucial for the stability of the assembly process.

Unlike the UR5, the Comau NJ 60-2.2 is not a collaborative robot and operates at higher speeds. It's typically used in industrial settings where human interaction is minimized. In our setup, it will act as a stabilizing scaffold, holding components in position while the human operator connects them.

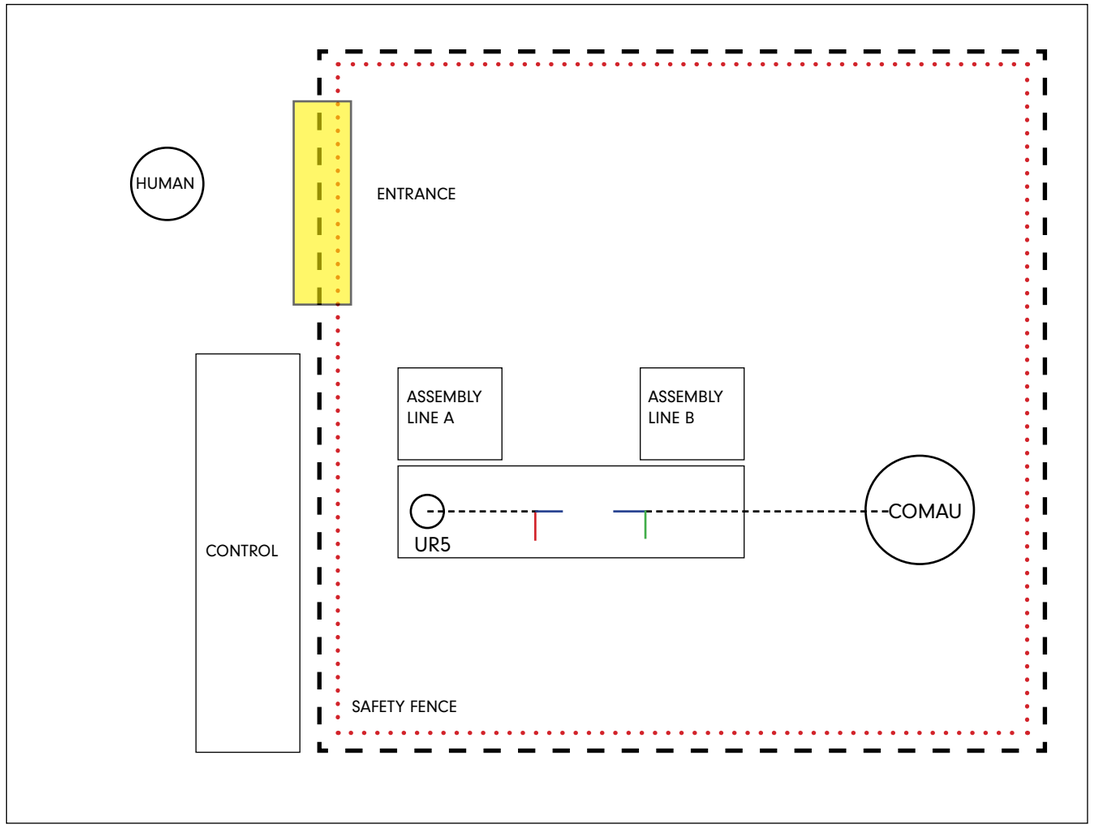


Figure 65: Prototype Setup(authors).

D2.Differences & Considerations

1. Collaborative vs. Industrial Design:
The UR5 is built for collaborative work and is designed to operate near humans safely, making it ideal for real-time adjustments. The Comau NJ 60-2.2, however, is an industrial robot that works at higher speeds and requires careful programming to ensure safety when humans are nearby.

2. Precision and Flexibility:
The UR5 excels in flexibility and precision, making it well-suited for tasks requiring small adjustments. The Comau NJ 60-2.2, by contrast, is more suited for holding parts in place over long periods due to its strength and stability.

3. Safety and Control Systems:
The UR5 has built-in safety features allowing it to stop or pause automatically when a human approaches. The Comau NJ 60-2.2 will need external safety protocols to ensure that it remains stable and does not move unpredictably when humans are near.

Summary of Considerations
In this co-working setup, where the UR5 and Comau NJ 60-2.2 share the task of holding and positioning components while a human makes the connections, several factors are key to ensuring efficient and safe operation:
-Safety protocols will be essential for the Comau NJ 60-2.2 since it lacks collaborative features, while the UR5 can work more dynamically near the human operator.
-Coordination between both robots is crucial. The UR5 should handle precision adjustments, while the Comau provides long-term stability.
- Real-time adaptability is best handled by the UR5, while the Comau focuses on providing a steady hold throughout the assembly process.

In conclusion, by combining the UR5's adaptability and collaborative design with the Comau NJ 60-2.2's strength and stability, we can create a balanced workflow that ensures precision, safety, and efficiency during assembly. The key will be in synchronizing or better specified "sequencing" the roles of both robots to complement each other while ensuring the human operator works seamlessly within the system.

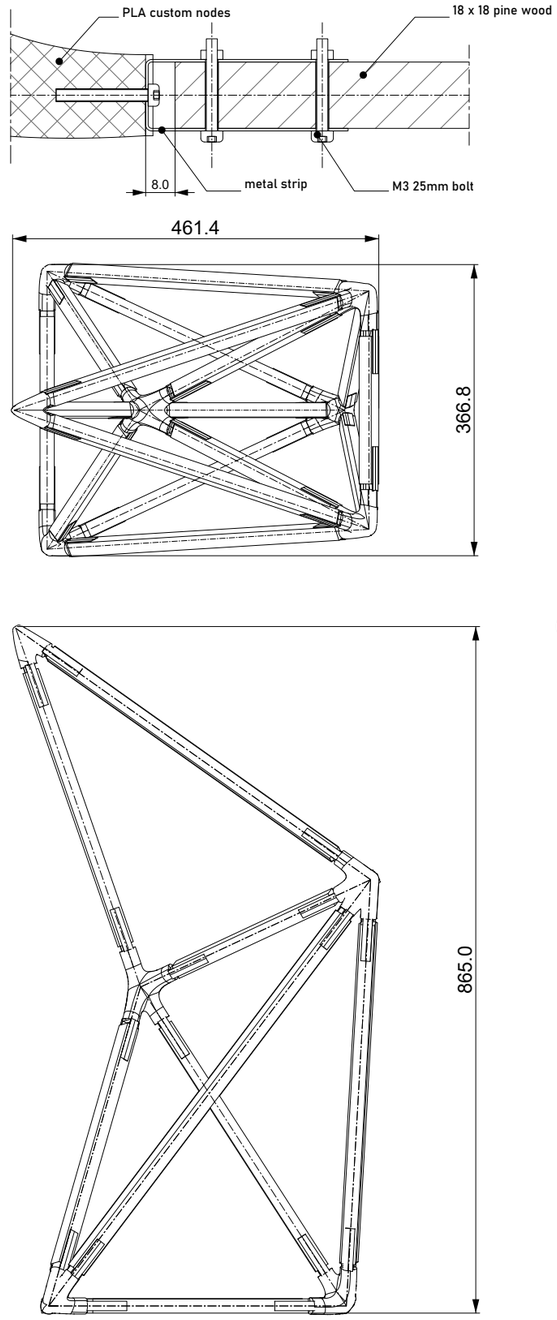
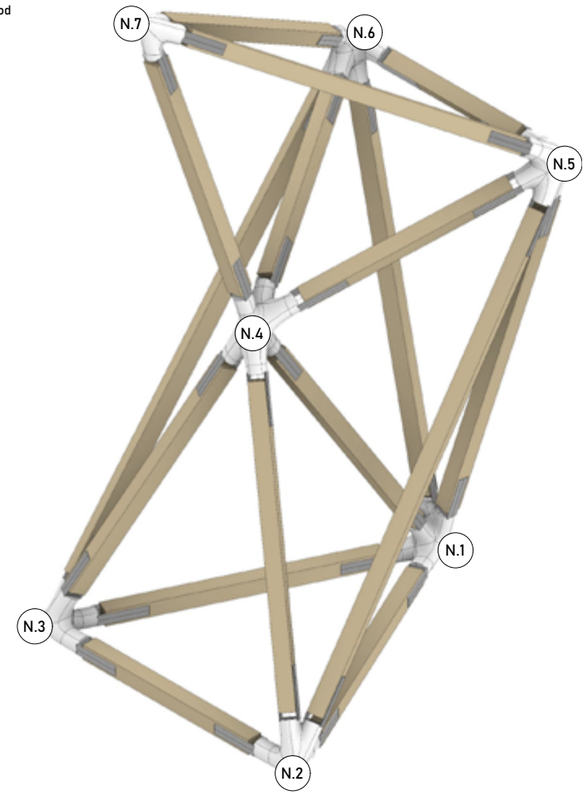


Figure 66: Prototype Drawings(authors).

D3.Prototype Design

To establish a proof of concept, we constructed a simplified version of a space frame that adhered to the same rules and principles as the full-scale design. Our prototype consisted of 13 elements, plus three additional components for the rigid base, and was assembled using our multi-robot system. We streamlined the node design by excluding the complex computational calculations necessary for the actual structure and incorporated extra tolerance to account for potential imprecisions.



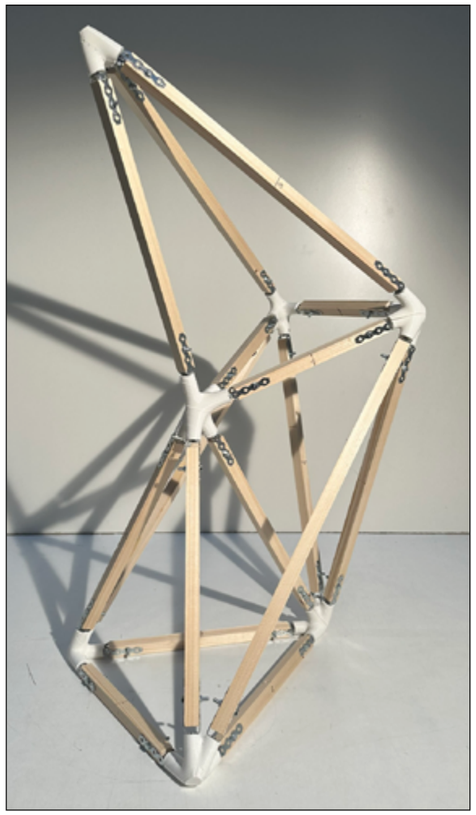
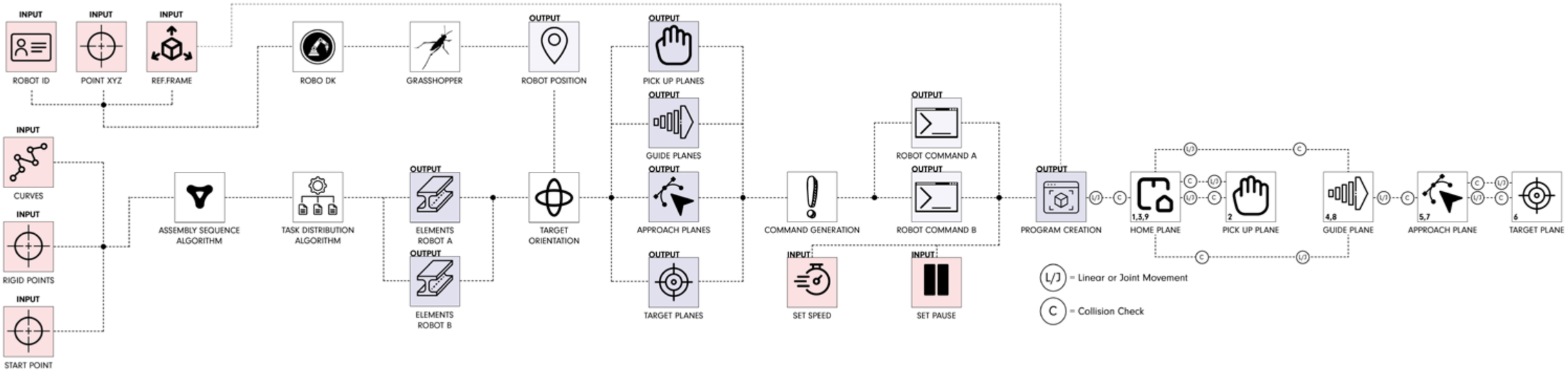


Figure 67: Prototype Photos(authors).



D4. System General Overview

Input Data and Initial Setup

• Robot ID: Unique identifier for each robot.

• Position Coordinates (POINT XYZ):
Defines the initial spatial context for accurate positioning.

• Reference Frame: Establishes the spatial reference for coordinated operationsbetween Grasshopper and RoboDK.

• Curves: Represent structural elements to be assembled.

• Rigid Points: Fixed spatial reference points ensuring that the robots understand the construction space and assembly targets.

• Start Point: Marks the initiation position for robot movements.

A link is established to RoboDK through Grasshopper where GH acts as the mediator informing RoboDk for the context of the setup, where the robots are situated, what are the other possible obstacles etc. Grasshopper retrieves the necessary data for each robot by using Robolink and searching for active robots inside of RoboDk. The Robot IDs are distributed then to an inverse kinematics and visualisation alogrithm where now its posible to see the position of each robot in the rhino workspace and its joints,synchronising in real time Rhino & RoboDK.

Assembly Sequence Algorithm

• Organizes construction steps into a stable build order using predefined sequences.

• Ensures structural rigidity throughout the assembly process, minimizing risks instability or hinge-like behavior.

Task Distribution Algorithm

The ordered elements from the assembly sequence algorithm are parsed into the task distribution algorithm which distributes the elements to Robot A or B depending on the element's location and orientation to the robot and the robot's reach.

Plane Generation and Orientation Planning:

Home Plane: The robot's resting or idle position, manually defined.

Pickup Plane: The position where elements are picked up by the robot, a predefined plane where stacked elements are staged for retrieval.

Target Planes: Generated dynamically based on each structural line's location and orientation, defined by the midpoint and starting point of each line to ensure accessible and collision-free assembly positions.

Approach Planes: Derived from target planes but slightly offset to avoid existing elements in the structure, acting as transitional points to minimize collision risks.

Guide Planes: Inserted between the pickup and approach planes when potential collisions are detected, providing additional waypoints to help the robots navigate around obstacles.

Orientation Planning:

Utilizes Grasshopper plugins like "Pufferfish" for creating average planes and "fox" for easy plane rotation, ensuring that target planes face the robot arm and avoid sharp angles while preventing collisions with other lines.

Robot Command Generation and Program Creation:

Translates assembly and orientation plans into executable robot commands. Orchestrates movements, gripper actions, and pauses to ensure synchronized actions between both robots without conflicts. Commands for Set Speed and Set Pause allow dynamic control over the robot's movements based on task requirements.

General Functionality:

Initializes a connection to RoboDK and retrieves the specified robot's base pose and current joint angles. Uses Denavit-Hartenberg (DH) parameters to calculate transformation matrices for each joint, determining their exact positions in 3D space. Visualizes joint positions and TCP orientation in Rhino, creating a plane at the TCP to represent its orientation and drawing axes to show its direction. Enhances safety and coordination by allowing users to monitor robot movements in real-time, preventing potential collisions.

This Grasshopper / RoboDK framework uses the following python scripts:

1)RobotSetup
Connects active robots from RoboDK to GH.

2)Visual_TCP
Retriece robot joints & TCP positions and visualises it in real time in Grasshopper.

3)TaskDistribution
The main framework that acts as the mediator from the assembly sequence to the GH recipe.

4)RobotCommands

5)RoboDkProgramCreation

In the following chapters there will be a thorough analysis of the computational and programming framework.

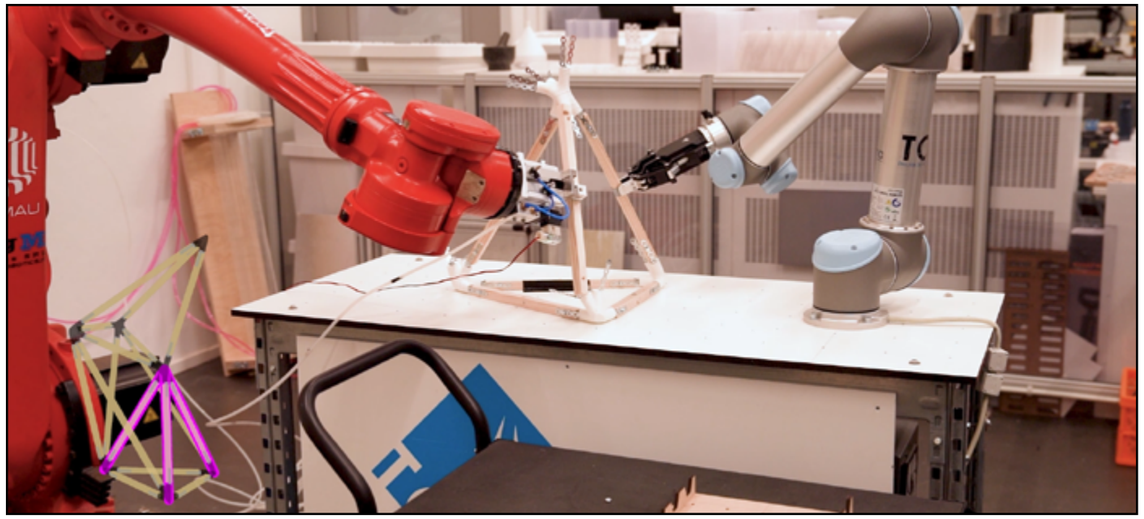
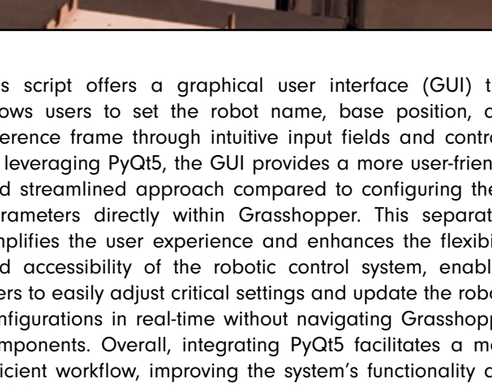


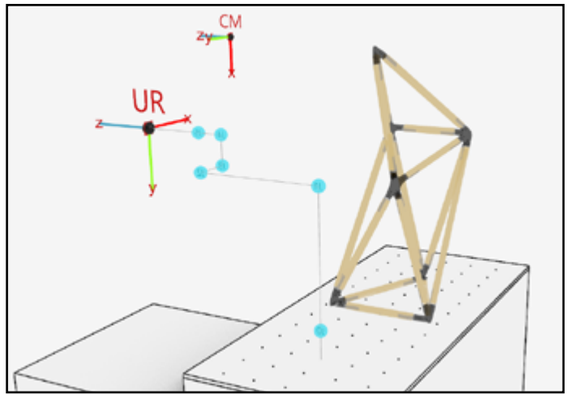
Figure 68: Prototype Setup(authors).

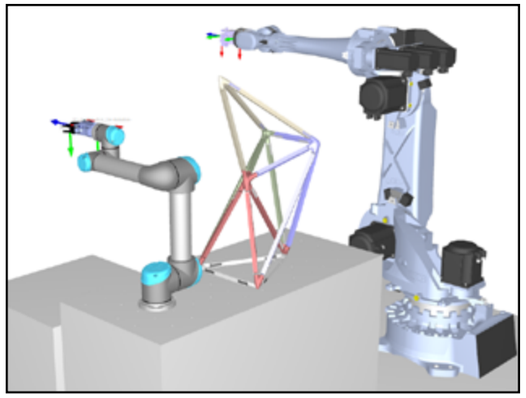
D5. Connecting Grasshopper & RoboDK

The script begins by importing essential libraries for interfacing with RoboDK (`robolink` and `robodk`) and handling mathematical conversions, specifically converting degrees to radians. A debug flag is set to enable detailed output for troubleshooting. It then attempts to import `Rhino.Geometry` to determine if it's running within the Rhino/Grasshopper environment; if not, it falls back to importing `rhino3dm`, ensuring flexibility in different execution contexts. A connection to the RoboDK application is established using `Robolink`, and a custom `RobotError` exception class is defined to handle specific robot-related errors gracefully. The script includes several functions: `connect_to_robot` establishes a connection to a specified robot in RoboDK, raising an error if unsuccessful; `set_robot_base` adjusts the robot's base orientation based on provided rotation angles while maintaining its position; `get_current_base_position` retrieves the robot's current base coordinates and the first joint's rotation angle; `validate_point_index` ensures that a given point index is within the valid range of a points list; and `rhino_to_robodk_coordinates` converts Rhino/Grasshopper point data into a format compatible with RoboDK. In the main logic, the script initializes parameters either from predefined values (when running outside Rhino) or expects them to be provided externally. It then connects to the specified robot, validates the point index, converts the selected point's coordinates, and either sets the robot's base orientation or retrieves its current position based on the `SetBase` flag. Throughout the process, debug messages provide real-time feedback. Any encountered errors are caught and reported, ensuring robust execution. Finally, the script assigns outputs appropriately: if running inside Rhino, it outputs the success message and base position to designated variables; otherwise, it prints the results to the console. This structured approach facilitates seamless integration between RoboDK and Rhino/Grasshopper, enabling dynamic robotic control and monitoring within diverse environments.

In addition to the integration within Grasshopper, an external Python script using the PyQt5 library has been developed to enhance user interaction and configurability. This script offers a graphical user interface (GUI) that allows users to set the robot name, base position, and reference frame through intuitive input fields and controls. By leveraging PyQt5, the GUI provides a more user-friendly and streamlined approach compared to configuring these parameters directly within Grasshopper. This separation simplifies the user experience and enhances the flexibility and accessibility of the robotic control system, enabling users to easily adjust critical settings and update the robot's configurations in real-time without navigating Grasshopper components. Overall, integrating PyQt5 facilitates a more efficient workflow, improving the system's functionality and usability.







Figures 69,70: Grasshopper-RoboDK connection(authors).

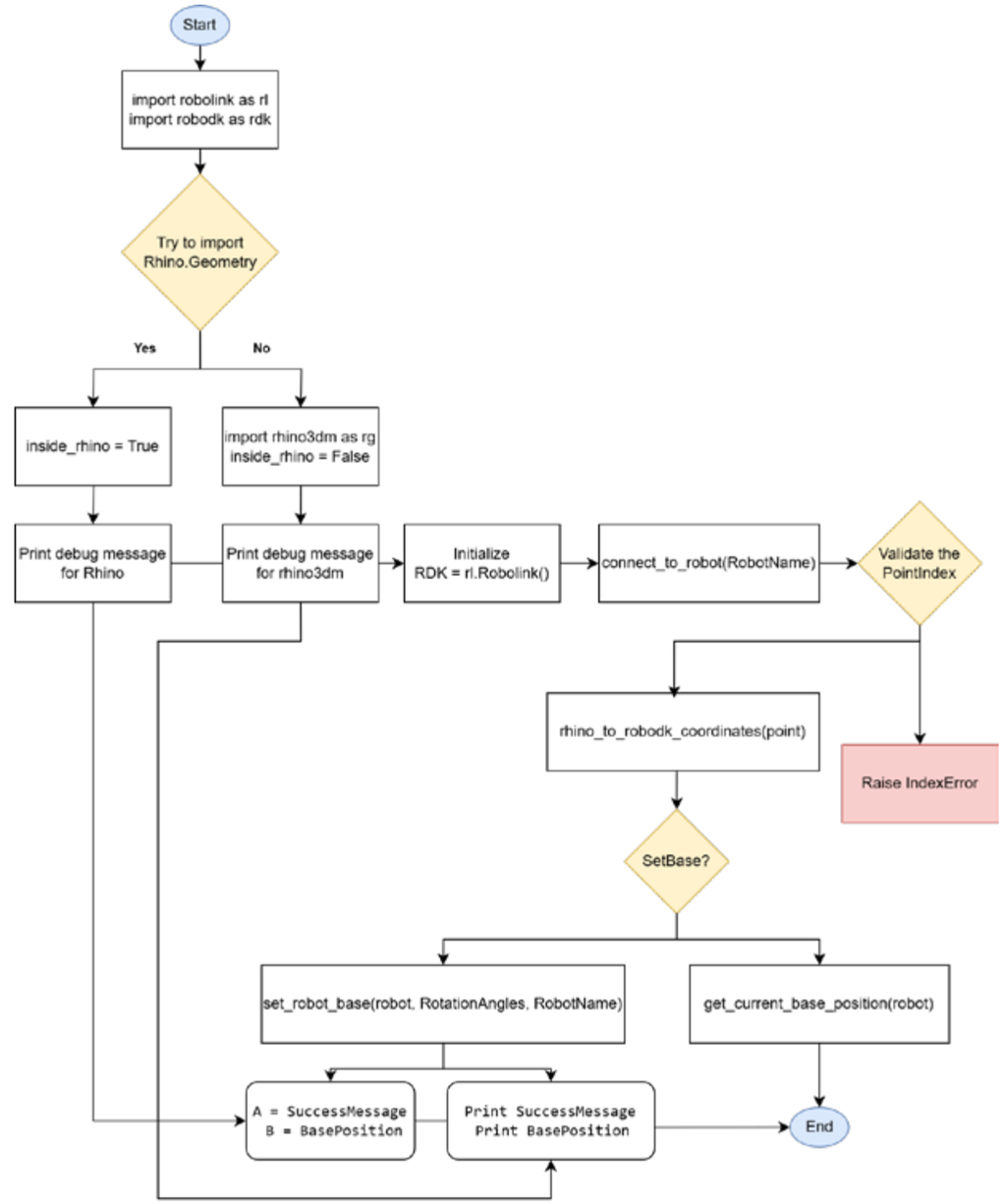


Figure 71: Setting up the robots(authors).

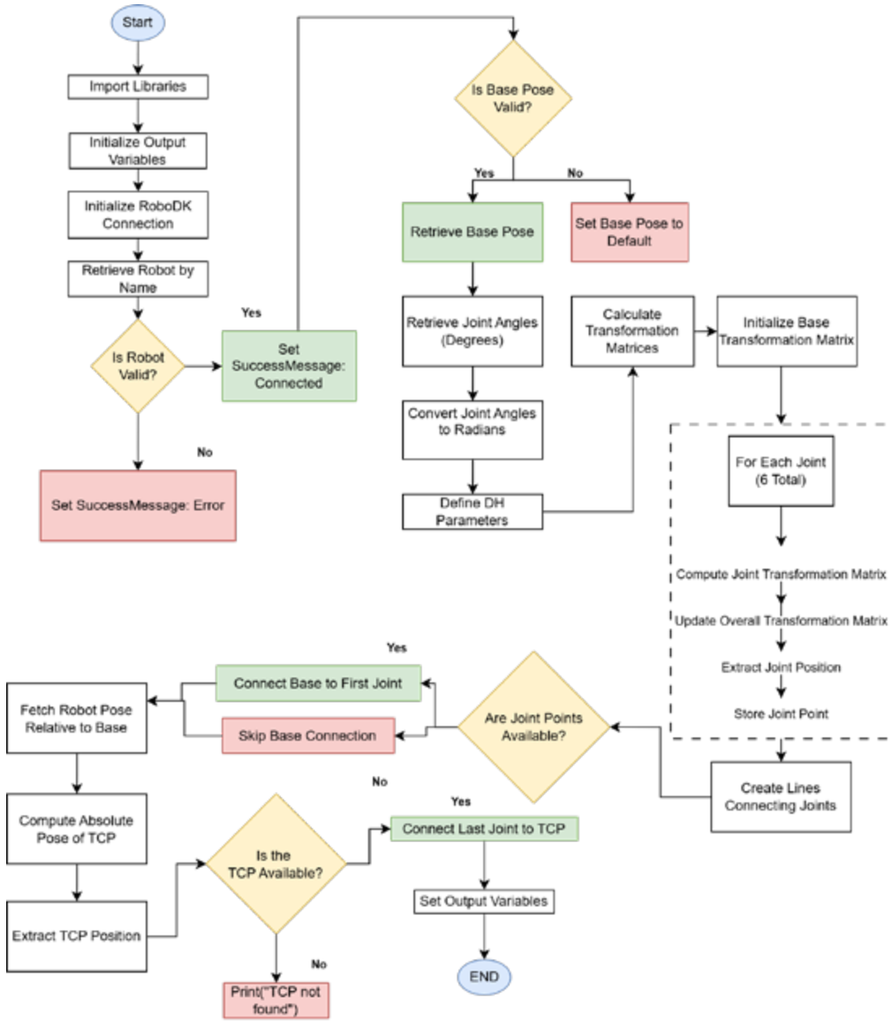


Figure 72: Script flowchart for joint and tcp retrieval(authors).

D6. Visual Synchronisation & TCP retrieval

This Python script is designed to work with RoboDK and Rhino to visualize a robot's joint movements, track the position of its Tool Center Point (TCP), and ensure safe coordination between multiple robots. It starts by importing necessary libraries: `robolink` and `robodk` for communicating with RoboDK, `Rhino.Geometry` for handling 3D geometry in Rhino, and `math` and `numpy` for performing the required mathematical calculations.

The script initializes a connection to RoboDK and attempts to find the specified robot by name. If the robot isn't found, it sets an error message; otherwise, it proceeds to gather the robot's base pose and current joint angles. Using the Denavit-Hartenberg (DH) parameters for the UR5 robot, the script calculates the transformation matrices for each joint. These matrices help determine the exact position of each joint in 3D space, which are then visualized in Rhino as points and lines, allowing users to see the robot's movements in real-time.

Additionally, the script calculates the TCP's position by combining the base pose with the robot's current pose. It creates a plane at the TCP to represent its orientation and draws axes to show its direction. This visualization is crucial for understanding how the robot interacts with its environment and ensures that it maintains safe distances from other robots, preventing any potential collisions.

By integrating these functionalities, the script not only provides a clear visual representation of the robot's movements but also enhances safety and coordination when multiple robots are operating in the same space. This makes it a valuable tool for anyone looking to manage and monitor robotic systems within Rhino effectively.

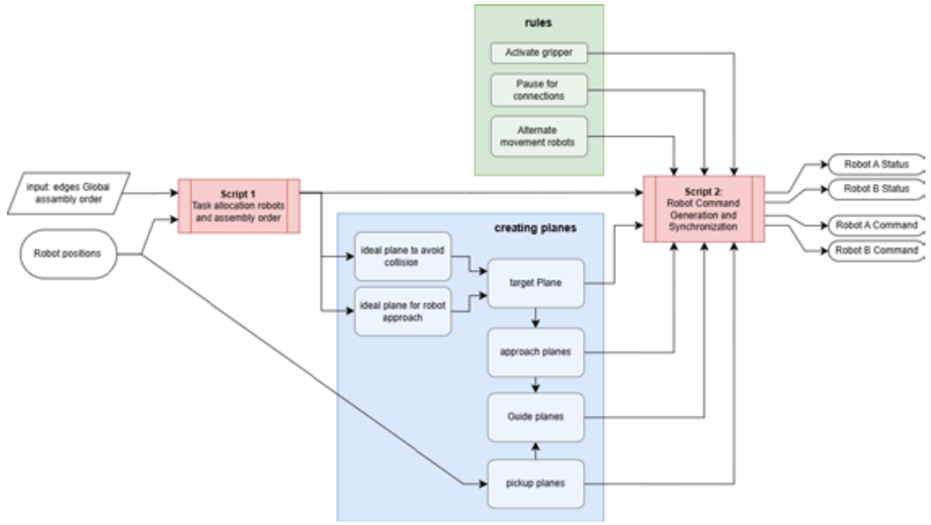


Figure 73: Framework for conversion of the assembly sequence to the robot programs(authors).

D7. Assembly sequence to robot programs

To construct the structure, we must transform the previously established assembly sequence into a series of assembly steps and corresponding programs for two robots. The input data consists of all the lines from the assembly sequence, grouped per node that they stabilize during the assembly process.
While the order of these groups remains fixed, the sequence of lines within each group can, and must, be reorganized to create a stable assembly program that maintains structural rigidity throughout the build, as demonstrated in prior research. It is crucial to avoid hinge behavior, as discussed in the assembly sequence. The first three lines in each group should be elements that are least likely to cause this issue. To address this, we subdivide the branches into {*,0}, containing the initial three lines needed to create a rigid node, and {*,1}, which includes all remaining lines. These subdivisions have different rules and restrictions and are created in Grasshopper.

Script 1: task allocation robots and assembly order:

The first script assigns tasks to the two robots based on their proximity to the lines. By calculat-ing the distance from each line to both robots, the script determines which robot should handle each line based on positional advantage.

For the first three elements in branch {*,0}, it is crucial to alternate between the robots to main-tain rigidity during assembly. This results in one robot placing two elements and the other plac-ing one. The script identifies the robot closest to all three lines. The closest robot is assigned two of the nearest elements, while the other robot is given the remaining element. The placement sequence follows an alternating pattern: the closest robot places the first and last elements, and the other robot places the middle element (A, B, A).

For the elements in branch {*,1}, there are fewer restrictions since the node is already considered rigid. Each element is simply placed by the robot closest to it.

The outputs of this script are essential for the next stage, as they provide a clear assembly order and define task allocation, ensuring precise coordination between the robots during the assem-bly process.

Steps:

1. Distance Calculation:
- For each line, calculate the distance from its midpoint to Robot A's base (RobotPointA) and Robot B's base (RobotPointB).
- Normalize these distances relative to the maximum distance for each robot to get relative distances. This is done because the robots used may have a different reach (like in the case of our prototype setup).

2. Branch {*;0} Special Handling:
- **Determine Closer Robot:** Sum the relative distances of all lines in branch {*;0} to each robot. The robot with the smaller total relative distance is considered the closer robot.
- **Line Assignment:** he closer robot is assigned the first two closest lines. The farther robot is assigned the third line
- **Line Ordering:** Ensure that the first two lines are assigned to alternate robots. If the first two lines are assigned to the same robot, swap the second line with the third.

3. Branche {0;1} Handling:
- For branches other than {*;0}, assign each line to the robot with the smaller relative distance to that line.
- Maintain the original assembly order of the lines within each branch.

4. Output Generation:
- **Assembly Order (assembly_order):** A data tree containing all lines in the correct assembly sequence.
- **Task Allocation (task_allocation):** A data tree indicating which robot (0 for Robot A, 1 for Robot B) is assigned to each line.

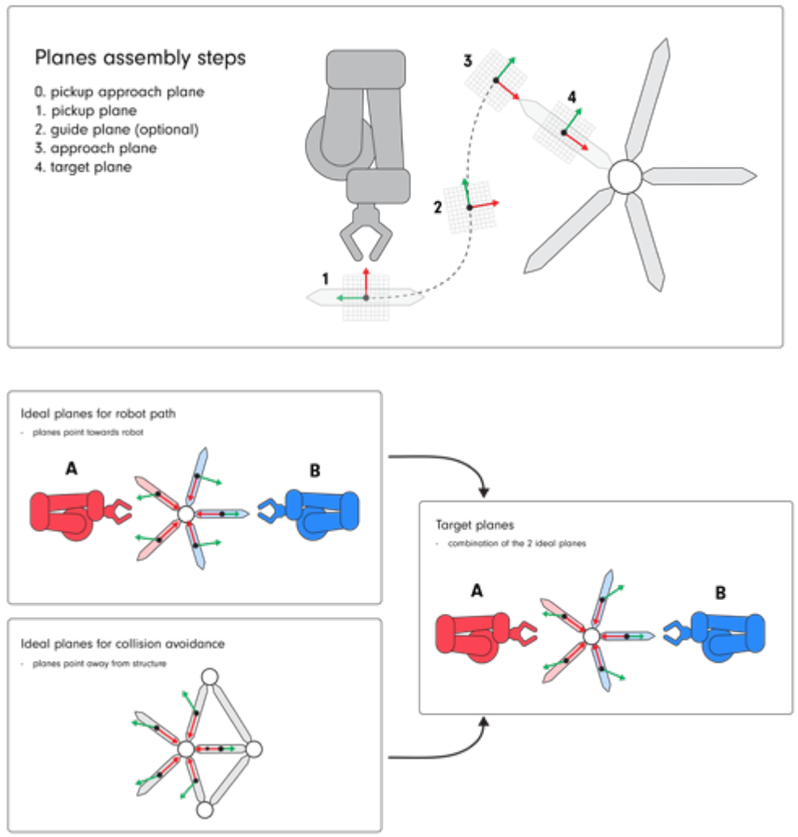


Figure 74: TCP Deconstruction(authors).

D8.Creating Planes for Robot Assembly

To develop a complete robot assembly program, a sequence of steps must be defined, with most steps repeated for each line placed. To facilitate robot movement, various planes need to be created:

1. Home Plane

This plane serves as the robot's starting point and idle position. It is manually set.

2. Pickup Plane

This is the location where the stack of elements is positioned. The robot moves to this plane to pick up an element before transferring it to a target plane. This plane is also manually set.

3. Target Planes

Target planes are generated based on the assembly order lines. Each plane is defined by three points: the base point, which is the midpoint of each line, and a second point, the line's starting point. This setup leaves one degree of freedom. To ensure that the target planes are accessible to the robot arms, they should ideally face the robot arm and avoid sharp angles. At the same time, to prevent collisions with other lines, the planes should ideally point away from the already assembled structure.

For the target planes, we use a combination or average of these two "ideal planes." We achieve this by using a Grasshopper plugin called "Pufferfish," which includes a component for creating an average plane between two planes. We also use the plugin fox for easy plane rotation in all desired axis.

4. Approach Planes

Approach planes are created from the target planes by moving them in it's z axis away from the assembled structure and are meant to avoid collisions with already placed elements.
While these steps are intended to minimize collisions, they do not entirely eliminate the risk. Therefore, a collision check should still be performed, with additional measures taken for any steps where collisions are detected.

5. Guide Planes

If collisions are detected despite the previous steps, guide planes can be added between the pickup plane and the approach plane. These guide planes are generated by averaging the pickup and approach planes and shifting the resulting plane away from the structure where the collision occurs. However, these should only be added when collisions are detected to avoid unnecessary movements.

D9.Robot Command Generation and Synchronization

The second script translates the created planes into detailed, step-by-step instructions for each robot, ensuring their actions are synchronized to prevent conflicts and maintain structural stability. By addressing task allocation and assembly requirements, the script generates commands to manage robot movements, gripper actions, and pauses.

Special attention is given to the initial assembly phase (branch {*;0}), where the robots must hold their elements in place without pausing until the third line is placed to ensure stability. The script also handles multiple pickup points and corresponding approach planes, allowing efficient operation when components are stored in different locations or need to be accessed sequentially.

Commands for pausing and waiting for other robots are left as open variables, as these functions may differ between systems and require synchronized setups. By managing gripper states and coordinating robot actions, the script supports a consistent and efficient assembly process.

Output will look as followed:

Steps:

1. Input Processing:
- **Assembly Order (target_planes):** Read the ordered list of target planes where each line needs to be placed.
- **Task Allocation (task_allocation):** Read the corresponding robot assignments for each line (0 for Robot A, 1 for Robot B).
- **Approach Planes:** Optionally, read the approach planes for both target positions and pickup positions.
- **Pickup Planes:** Read lists of pickup planes and pickup approach planes for both robots.
- **Other Inputs:** Read commands for pausing (pause_command), waiting (waiting_command), and home positions (home_plane_A, home_plane_B).

2. Initialization:
- Set up counters and indices to track the robots' gripper states, last lines placed, and pickup plane usage.

3. Command and Status Generation:
- **Per Line:** For each line in the assembly order, generate a sequence of commands and statuses for the assigned robot, including:

  - Moving to the pickup approach plane (if provided).
  - Moving to the pickup plane.
  - Closing the gripper.
  - Moving to the target approach plane (if provided).
  - Moving to the target plane.
  - Pausing for connection (if applicable).
  - Opening the gripper.
  - Moving back to the target approach plane (if provided).
  - Moving to the home position (if provided).
  - Update the gripper state and last line counter for the assigned robot.

- **Synchronization:**
  - Ensure that when one robot is performing actions, the other robot is either waiting or holding its last line, depending on its gripper state.
  - For pauses required during assembly, both robots pause simultaneously if applicable.

4. Branch {*;0} Special Handling:
- **First Two Lines:**

  Both robots place their assigned lines without pausing for connection and hold them in place (gripper remains closed).

  The robots' statuses reflect holding the line until the next phase.
- **After First Two Lines:**

  Both robots pause for connection simultaneously.

  The robot assigned to the third line proceeds with its usual steps, including pausing after placement.

  The other robot continues holding its line until the pause for connection is completed.
- **Completion:** After the third line is placed and the pause is complete, both robots open their grippers and proceed to their home positions if specified.

5. Multiple Pickup and Approach Planes:
- Cycle through the provided lists of pickup planes and pickup approach planes for each robot, looping back to the start when the end is reached.

6. Optional Steps Handling:
- If certain inputs (e.g., approach planes, pause commands) are not provided, the script skips the corresponding steps.

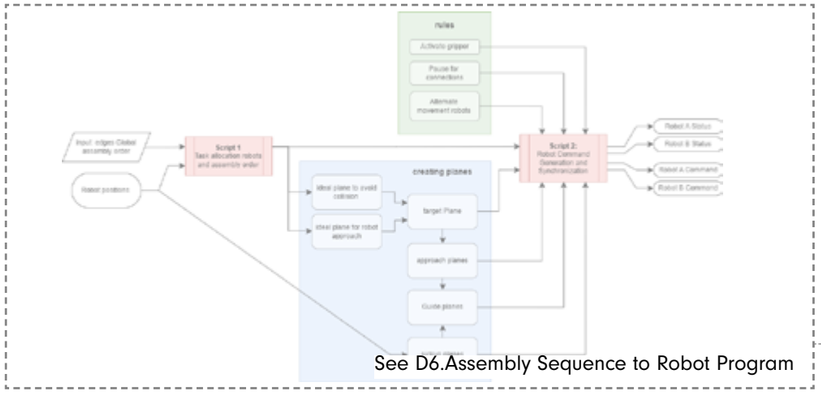
7. Output Generation:
- **Robot Commands and Statuses:** Generate synchronized data trees for each robot, detailing the sequence of commands and statuses for the entire assembly process.

COMMAND ROBOT A

0	O(1280.89,0.00,802.00) Z(0.00,0.00,-1.00)
1	gripper = 1
2	O(983.88,-573.33,1181.61) Z(-0.85,0.46,-0.24)
3	O(770.39,-457.80,1121.85) Z(-0.85,0.46,-0.24)
4	waiting
5	waiting
6	waiting
7	waiting
8	waiting
9	waiting
10	pause
11	O(1400.89,0.00,802.00) Z(-0.00,0.00,-1.00)
12	gripper = 1
13	O(692.60,-855.16,1257.37) Z(-0.04,0.84,-0.54)
14	O(682.89,-645.30,1121.85) Z(-0.04,0.84,-0.54)
15	pause
16	gripper = 0
17	O(692.60,-855.16,1257.37) Z(-0.04,0.84,-0.54)
18	world xy
19	waiting
20	waiting
21	waiting

STATUS ROBOT A

0	Move to pick up
1	Close gripper
2	Move to approach plane 1
3	Move to target 1
4	Holding edge 1
5	Holding edge 1
6	Holding edge 1
7	Holding edge 1
8	Holding edge 1
9	Holding edge 1
10	Pause for connection
11	Move to pick up
12	Close gripper
13	Move to approach plane 3
14	Move to target 3
15	Pause for connection
16	Open gripper
17	Move to approach plane 3
18	Moving to home plane
19	Waiting for other robot
20	Waiting for other robot
21	Waiting for other robot



See D6.Assembly Sequence to Robot Program

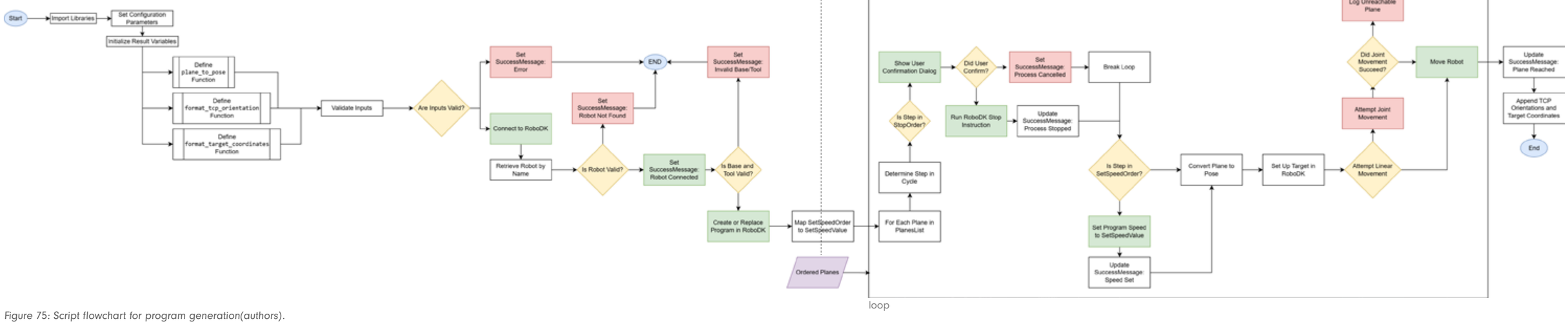


Figure 75: Script flowchart for program generation(authors).

D10.Robot Program Generation

The script begins by importing essential libraries necessary for interfacing with RoboDK and Rhino, as well as performing mathematical computations. Specifically, it imports `robolink` and `robodk` for communication with the RoboDK robotic simulation environment, `robodk.robodialogs` to handle user dialogs within RoboDK, and `Rhino.Geometry` for managing and visualizing geometric data within Rhino. Additionally, the `math` library is included for basic mathematical operations, and `numpy` is utilized for advanced numerical calculations, particularly for handling transformation matrices based on Denavit-Hartenberg (DH) parameters.

Configuration parameters are defined to control various aspects of the robot's operation. The `cycle_length` determines the number of planes processed per cycle, while global variables like `speed_ms`, `accel_mss`, and `blend_radius_m` set the robot's speed, acceleration, and blending radius, respectively. These parameters can

be adjusted to tailor the robot's performance to specific tasks. The script initializes several output variables, including `SuccessMessage`, `TargetPoses`, `Program`, `TCPOrientations`, `TargetPointCoordinates`, and `PlaneExtra`, which will store the results of the robot's movements and any additional information related to unreachable planes.

Two helper functions are defined to streamline the script's operations. The `plane_to_pose` function converts a Rhino plane into a RoboDK pose matrix, facilitating the translation

of geometric data between the two environments. The `format_tcp_orientation` and `format_target_coordinates` functions format the TCP's orientation and target coordinates for logging purposes, ensuring that the information is presented clearly and systematically.

Before executing the main logic, the script includes an input validation function, `validate_inputs`, which ensures that all necessary input parameters—such as `RobotName`, `PlanesList`, `UpdateRoboDK`, `StopOrder`, `SetSpeedOrder`, and `SetSpeedValue`—are properly defined and meet the required criteria. This validation step is crucial for preventing runtime errors and ensuring that the script operates with valid and meaningful data.

Upon successful validation, the script establishes a connection to RoboDK and attempts to retrieve the specified robot using the provided `RobotName`. If the robot is not found, an error message is set, and the script halts further execution. If the robot is successfully connected, the script proceeds to create or replace a program within RoboDK named "MoveThroughPlanesProgram," setting the robot's base frame and tool. This program orchestrates the robot's movements through the defined planes.

The script then maps any speed adjustments specified in `SetSpeedOrder` to their corresponding values in `SetSpeedValue`, allowing for dynamic speed changes at designated steps within the cycle. It iterates through each plane in `PlanesList`, calculating the target pose relative to the robot's reference frame and adding these targets to RoboDK. For each plane, the script attempts a linear movement; if unsuccessful, it falls back to a joint movement, ensuring that the robot can reach the desired position using the most appropriate method. Throughout this process, the script logs success messages and records any planes that remain unreachable, thereby maintaining a log of operations and potential issues.

Additionally, the script focuses on visualizing the robot's joint movements within Rhino by calculating the positions of each joint based on the DH parameters and transformation matrices. These joint positions are stored as `rg.Point3d` objects and connected with lines to represent the robot's structure. The TCP's position and orientation are also calculated and visualized, providing a clear graphical representation of the robot's operational status. This visualization is crucial for setting up safe distances and coordinating movements between multiple robots, as it allows users to monitor and adjust the robots' positions in real-time, preventing potential collisions and ensuring harmonious operation within a shared workspace.

Throughout its execution, the script updates the `SuccessMessage` variable to reflect the status of each operation, providing clear feedback on the robot's connectivity, program creation, and movement execution. Finally, it prints out the `SuccessMessage`, along with other result variables as needed, allowing users to verify the outcomes of the script's operations. This structured approach not only facilitates effective robotic control and monitoring within Rhino but also enhances safety and coordination in environments where multiple robots operate concurrently.

E.Reflections

Sander Bentvelsen

Looking back, the structural engineering of this project proved some unique challenges that are inherent to this new method of assembly. Problems created to 'free' a structure from scaffolding.

Although it's nowhere near the final design and calculations that are needed for such a big undertaking, I think we did manage to identify the mayor problems and how to potentially solve them. Especially the structural checks of the assembly sequence and the re-detailing of the connections proved insights to what might be possible when creating large scale rigidity preserving space frames. Even if the method for getting the elements in place might vary to a potential world application -Seeing as there is currently no system on the market that allows for the movement of two large scale robot arms - The topological rules and assembly sequence for the creation of a rigidity preserving structures could be applied to many more structures. Perhaps even to future scaffolding itself if it's not possible to place it on a ground plane.

On a personal note, I believe that the topology optimization of the nodes and the connecting elements can be pushed much further, simplified or approached differently with adjustable nodes could yield similar benefits. Currently, the performance gains of our attempt aren't great compared to simpler spherical designs, so we do not fully exploit the advantages of 3D steel printing yet. 3D printing is currently also likely too costly, although it does address the complexity of varying node details caused by the differing angles of the beams throughout the structure. However, with advancements in steel 3D printing technology and potential cost reductions in the future, we may see more opportunities to create complex and efficient spaceframe structures.

I am particularly pleased that I had the chance to experiment with two new software packages related to structural engineering, which I plan to use for my graduation work. CORE has provided me with intuitive insights into structural design and the design of spaceframe structures, and I am excited to apply these skills and knowledge to my thesis.

Tony Mavrotas

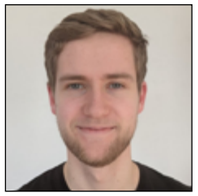
My experience with CORE proved invaluable in introducing me to a new field—robotics—while also deepening my understanding of writing code for reuse, collaboration, and testing.

Throughout the project, I primarily focused on the assembly sequence algorithm. My role involved implementing the core logic, engaging in discussions with the team, and iterating on the algorithm based on the structural performance of each assembly step. This iterative process resulted in five key versions of the algorithm, which were rigorously validated and refined.

One of my most significant learning moments came from addressing the hinge behavior issue that emerged when the algorithm produced undesired structural outcomes. To resolve this, I developed a solution that efficiently used matrices and pairwise distance computations, ensuring high computational performance. Additionally, I was pleased to have structured the solution as a class from the outset, which made updating and debugging far clearer and more manageable.

A crucial factor in the development process was the direct feedback from my teammates, who acted as the end-users of the algorithm. For example, Sander provided insights into where the script was failing to prevent unrealistic hinge behavior, while Simos and Pepijn highlighted issues with element ordering that conflicted with a robot's kinematics. This constant feedback loop was instrumental in extensively testing and improving the algorithm, making it a highly educational experience in refining software for real-world applications.

Despite its accomplishments, the algorithm still has room for improvement, both structurally and in terms of robot kinematics. Future work could focus on optimizing the logic to reduce or eliminate crane usage during assembly and integrating robot kinematics more comprehensively. Nevertheless, given the time constraints, our final script effectively enabled the simulation of a complete assembly sequence.



Pepijn Feijen

I genuinely enjoyed this project, especially working in a group. Our collaboration was effective, with everyone having a clear role, which created a positive and productive atmosphere. The teamwork made tackling challenges more manageable, and it was rewarding to see our progress as a team.

The Python crash course was very useful, particularly for learning to write our own Grasshopper components. However, I did face a dilemma about how much to code manually versus using AI tools. While I initially tried coding on my own, I found that using AI became much faster for com-plex tasks. Over time, I improved in using these tools effectively through prompt engineering and understanding AI-generated scripts better.

In this project, my main focus was on developing the "Assembly Sequence to Robot Programs." This was a challenging task where "task allocation between robots," "assembly order," and "op-timal approach angles to avoid collisions" all intersected and needed to be addressed together. Eventually, I found an effective solution by breaking the problem down into two separate scripts, which allowed us to manage these interrelated aspects more efficiently.

One aspect we didn't address was handling elements that are out of the robot arm's reach. For larger structures, a moving robot base is essential to reach all necessary areas. Ideally, the base position for each group of elements should be automatically optimized, along with a process to manage any unreachable elements. This would improve the scalability and flexibility of the as-sembly system for more complex structures.

Building the actual prototype was the most frustrating part of the course. Working with two dif-ferent robots and grippers proved challenging, especially with technical issues like the Comau robot's lack of available expertise and the grippers' adaptive pressure function causing dropped elements. The absence of a clear contact for support made it even more difficult, which was draining and time-consuming.

Simos Maniatis

I thoroughly enjoyed participating in the CORE project. Initially, the general and abstract nature of the topic—automation in construction—was quite intimidating. However, after extensive self-study and research, I was able to gain a solid understanding of the area I wanted to focus on.

My personal contribution to this project was primarily in setting up the computational framework necessary for the multi-robot setup. Establishing a link between Grasshopper and RoboDK wasn't much of a challenge due to my coding experience and the comprehensive RoboDK API. The real challenge lay in developing a flexible framework for the first time, especially within a 10-week timeframe filled with many unknown variables. For more than half of the project duration, we were uncertain about which robots we would use for our prototype. This uncertainty led to considerable stress, as we initially reached out to external resources without success, only to eventually rely on our own faculty. In hindsight, I believe that if both our team and the teaching staff had pushed for this decision earlier, we could have advanced much further.

Our project involved coordinating two robots in a specific sequence: the UR5 and the Comau NJ 60 2.2. This choice presented two significant challenges: a lack of expertise with the Comau robot and complications with the dual gripper setup. A newer gripper was used for the UR5 since the other was moved to the Comau. Despite these obstacles, and with the support of our supervisors and the teaching assistants, we managed to navigate through the setbacks. Unfortunately, these issues limited our ability to conduct multiple tests. Even so, the experience was intense but ultimately rewarding.

I want to thank my team for their dedication and collaboration. The work packages were well distributed, and each of my colleagues took responsibility for their individual challenges and succeeded. This project not only enhanced my technical skills but also taught me the importance of flexibility, teamwork, and effective communication in overcoming complex challenges.

F.References

1.Audonnet, F., Hamilton, A., Aragon-Camarasa, G., & Gpu, X. (n.d.). A Systematic Comparison of Simulation Software for Robotic Arm Manipulation using ROS2.Barnes, M. R. (1999).

2.Form Finding and Analysis of Tension Structures by Dynamic Relaxation. International Journal of Space Structures, 14(2), 89–104. https://doi.org/10.1260/0266351991494722

3.Bruun, G., Besler, E., Adriaenssens, S., & Stefana Parascho. (2024). Scaffold-free cooperative robotic disassembly and reuse of a timber structure in the ZeroWaste project. Construction Robotics, 8(2). https://doi.org/10.1007/s41693-024-00137-7

4.Mass Customization of Reciprocol Frame Structures: Digital Optimization and Robotic Manufacture. (2016, October 24). Harvard Graduate School of Design. https://www.gsd.harvard.edu/project/mass-customization-of-reciprocol-frame-structures/Panagiotis Kyratsis, Athanasios Manavis, & J. Paulo Davim. (2023).

5.Computational Design and Digital Manufacturing. Springer Nature.Paolo, R., Andrea, M., Zanchettin, I., & Maderna. (2018). Collaborative robot scheduling based on reinforcement learning in industrial assembly tasks.Stefana Parascho. (2019).

6.Cooperative Robotic Assembly: Computational Design and Robotic Fabrication of Spatial Metal Structures. https://doi.org/10.3929/ethz-b-000364322