Precise Assembly of Truss Structures by Distributed Robots

by

Erik Komendera

B.S.E., University of Michigan, 2007

M.S., University of Colorado, 2012

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

2014

This thesis entitled:
Precise Assembly of Truss Structures by Distributed Robots
written by Erik Komendera
has been approved for the Department of Computer Science

Prof. Nikolaus Correll

Prof. Sriram Sankaranarayanan

Prof. Tom Yeh

Prof. Eric Frew

Prof. Daniel Scheeres

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Komendera, Erik (Ph.D., Computer Science)

Precise Assembly of Truss Structures by Distributed Robots

Thesis directed by Prof. Nikolaus Correll

Assembly robots have been in operation in industry for decades, predictably repeating the same precise motions in closed workspaces to assemble products cheaply and in mass quantities. However, in the field, robotic assembly has seen only spurts of progress, and no short-term feasible applications. NASA and the space industry desire robotic construction methods to remove the upper limit on size. Space telescopes are highly desired, but require structural precision on the order of microns. Previous approaches were ruled out because the precisely machined components were expensive, heavy, and prone to failure. The recent advent of cheap robotic swarms has revived interest in academia, but most research requires self-correcting, interlocking components, instead of commodity materials.

In this thesis, I describe the Intelligent Precision Jigging paradigm, a solution to the problem of practical robotic assembly, with application to precision truss assembly. Intelligent Precision Jigging Robots (IPJRs) are robots that work in groups of three to incrementally assemble a structure. They set and hold distances with high precision, enabling coarse external manipulators to weld the commodity parts together and perform other tasks.

To maximize the utility of the IPJR paradigm to the fullest extent, I present algorithms for finding near-optimal assembly sequences and for implementing Simultaneous Localization and Mapping (SLAM) to maintain an estimate of the assembly process through the accumulation of local strut length measurements. I define a model of truss assembly probability and a minimizing metric based on the covariance trace. I show that structure error grows cubically with node count. I present a three-step approach for generating near-optimal assembly sequences; commencing assembly on a central location of the structure, greedily assembling to minimize the covariance trace, and performing a local search on the space of sequences to swap steps until a local minimum is

found. I show that this method consistently generates more precise sequences than any process alone.

I then simulate the SLAM method with four different estimators commonly used in for SLAM; a least linear squares approach, the Extended Kalman Filter, the Unscented Kalman Filter, and the Maximum Likelihood Estimator. I show that when nonlinearity in the assembly process is dominant, the Maximum Likelihood Estimator is better than the other estimators, but for space telescopes with precision requirements, all four are functionally equivalent. I also show that when SLAM is used, the difference in covariance trace between sequences is reduced, reducing the need for finding globally optimal sequences. SLAM also mitigates the growth of structure error.

Finally, I present the results of physical assembly trials on a telescope truss made of aluminum tubes, assembled by three IPJRs using two methods: an open loop approach, and an MLE-SLAM approach. I show that the MLE-SLAM assembly algorithm works even when the physical trials included unmodeled processes such as deformation under gravity, outperforming the open loop algorithm.

Dedication

To my wife Emily Komendera, who, on October 16th, 2009, told me that she wanted to be married to a doctor, and stood behind my detour for four uncertain, challenging, but ultimately rewarding years.

To my grandmother Marion Komendera, who taught me to read, encouraged my curiosity, and reminded me of what was beyond the horizon when it seemed so far away.

And finally, to the star whose supernova created all the matter that formed the Earth, the Sun, and all of life as we know it. Without that supernova, it would have been exceptionally challenging to write this thesis.

Acknowledgements

I thank my NASA mentors John Dorsey and Bill Doggett for years of design and hardware support, and for bringing me onto their team. I thank fellow NASA researchers Bruce King, Dave Mercer, and Rob Hafley for their support in making Prototype 2. I thank my wife Emily for helping me assemble, disassemble, and measure the 3D truss assembly experiments. I thank Michael Otte for suggesting truss sparsity and various other things, Dustin Reishus for helping me develop the prototype idea of Intelligent Scaffolding, Andy McEvoy for discussions and contributions on truss stability, Eric Frew for allowing me to use his Vicon lab, and Radhika Nagpal for ideas leading to the development of Prototype 1. Finally I thank my advisor, Nikolaus Correll, for four years of support, through proofreads, letters of recommendation, delivering a talk on my behalf, and innumerable discussions.

Contents

Chapter

Tables

Table

Figures

Figure

Algorithms

Algorithm

Chapter 1

Introduction

Assembly robots have found practical use on assembly lines for decades, predictably performing the same routine motions in a streamlined process that has made advanced technology cheap and widely available. However, in situations deviating from the predictable and routine, outside of the assembly line, robotic assembly has never advanced past the prototype and toy problem phase. In academia and industry, nearly all of the research focus has been on using mobile robots to connect interlocking (often homogeneous) parts. These efforts have produced impressive technical demonstrations but nothing else.

Robots that can replace humans on construction projects have long been desired, but progress toward that goal has been limited. One notable application is construction in hazardous environments that humans cannot easily access. Making this practical requires the intersection of numerous subfields, including sensor fusion, distributed algorithms, motion planning, error detection and correction, and sequence planning. Construction calls for the ability to handle a wide range of issues, including cutting and shaping parts as needed, addressing assembly mistakes, distributing tasks to specialized agents, and handling unexpected events. Accuracy requirements call for sensors and optimal estimation algorithms, especially when an assembly does not consist of interlocking, self-correcting linkages. Complex and expensive robots that can handle all tasks are particularly risky, as the loss of one can be devastating. Reliance on pre-made, precisely machined parts can also lead to failure if just one unique part breaks. Often, the attachment of a subassembly requires several agents coordinating, for example, to hold up and maneuver large parts.

The aim of my research is to show that space telescope optical bench trusses, with precision requirements on the order of $10\mu m$, can be built by teams of Intelligent Precision Jigging Robots (IPJRs). Most structures do not require this level of precision, so I consider the problems involved in space telescope assembly to be a superset of the problems in the assembly of other kinds of structures. Intelligent Precision Jigging Robots are robots that can precisely hold a pose between disconnected parts (in welding, this is known as **jigging**), enabling an external agent (such as a robotic arm) to permanently bond the parts. IPJRs adjust the distances between the parts using highly precise actuators and precise sensors for distance measurement. IPJRs free the external agents from having a high precision requirement, reducing the complexity of the external agent. A set of IPJRs and external agents performing assembly are individually simpler than an all-purpose assembly robot, thus they can be made more inexpensively, enabling robustness.

IPJRs require both precise hardware and smart assembly algorithms to reach the extreme level of precision needed for space telescope trusses. The "smart assembly algorithms" are the focus of this thesis; both algorithms for identifying suitable assembly sequences, and formulation of the assembly problem as a Simultaneous Localization and Mapping (SLAM) problem. This is the first time anyone has applied SLAM concepts to the problem of assembly, in which the process and manufacturing error on the parts needs to be considered. As such, my work is not an incremental improvement in a saturated field, but a new branch that attempts to solve field assembly in a new way.

1.1 The State of On-Orbit Construction

Assembling structures in space [75] has the potential to overcome payload limitations of earth-based missions, thereby enabling the manufacturing of scalable structures. Space telescopes that could be assembled on orbit are a high priority for NASA [66] and the space industry [26], with proposed diameters of tens of meters up to hundreds of meters. Except for the International Space Station (ISS), all current spacecraft are transported to orbit as an integrated unit using a single launch. This severely constrains the mass and size of the spacecraft system because it

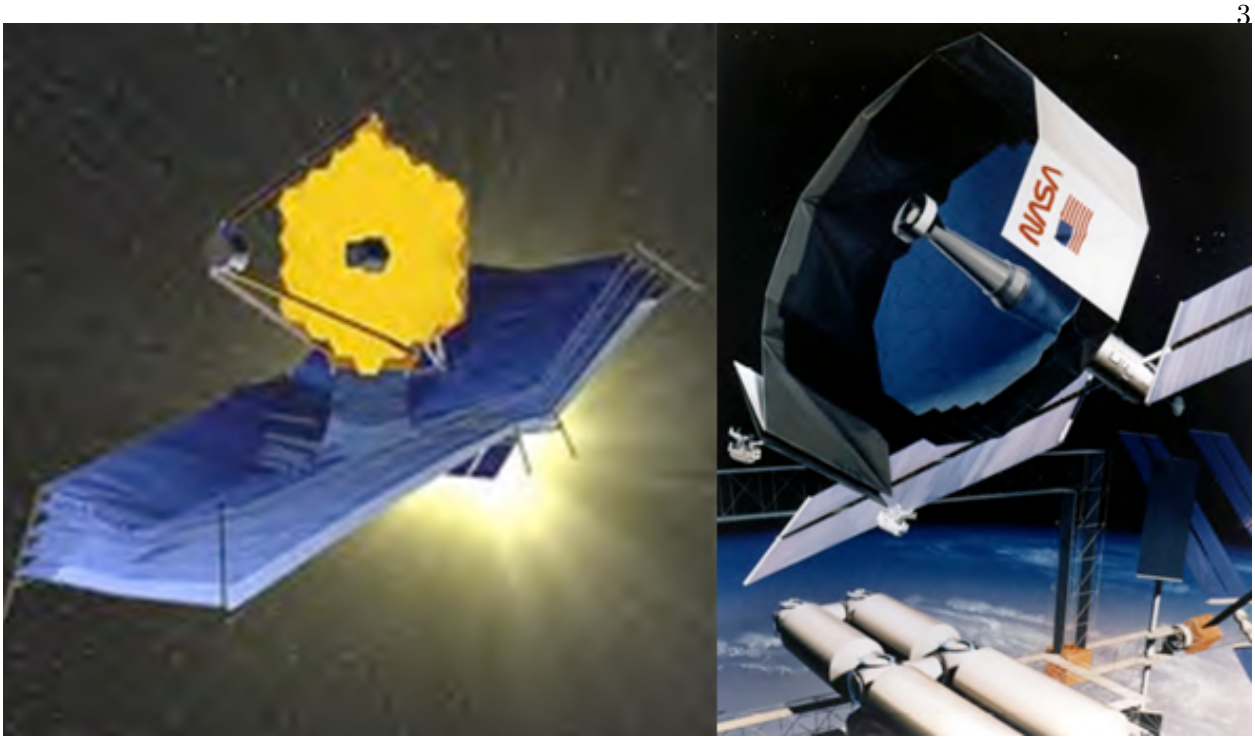


Figure 1.1: Left: the James Webb Space Telescope, a $6.5m$ telescope that will be launched assembled. Right: $20m$ far-infrared telescope concept [38, 44] that requires on-orbit assembly.

must be designed to the mass and volume constraints of the chosen launch vehicle and its payload shroud, as well as the loads imposed by the launch environment. Once on orbit, various systems, such as solar arrays, radiators and antennas are deployed to achieve an operational configuration. The James Webb Space Telescope (JWST), shown in Figure 1.1, with a primary mirror diameter of 6.5 meters, likely represents an upper limit to the size of aperture that can be achieved for a single-launch telescope using deployable structures and mechanisms. This is because the spacecraft complexity rapidly increases with increasing number of deployable mechanisms and systems, as does the potential for deployment failure, resulting in a decrease in spacecraft and mission reliability. Although an on-orbit servicing and repair capability would help to mitigate spacecraft mission risk resulting from deployment and other early system failures, this capability does not currently exist. Furthermore, current spacecraft, including the JWST, are not designed to take advantage of such services (even if they did exist). An alternative, espoused by some, is to build a Heavy Lift Launch

Vehicle (HLLV) that includes a larger payload shroud. The cost of designing and manufacturing such a special-purpose launch system would be very expensive, as would the cost for each launch (because of launch infrequency). Spacecraft designed to launch on a HLLV might be somewhat larger but there would still be a limit on the maximum size of spacecraft that could be launched. Also, lack of servicing and repair capability would still be inherent in the traditional design and operation approach.

One approach that can result in larger space systems and takes advantage of multiple launches, is to incorporate on-orbit assembly, as was used for the ISS. The ISS was assembled from a relatively small number of very large and massive modules or components, with each component requiring its own launch. The components were positioned and berthed robotically on orbit, and then permanent mechanical and utility line connections completed. However, many of the large telescope and exploration vehicle applications include large area or span trusses to provide lightweight, high stiffness and precise support and backbone structures, all of which require assembling a much larger number of small and lightweight truss elements. Previous approaches proposed for assembling truss and telescope structures and systems in space have been perceived as very costly because they require high precision and custom components that rely on mechanical connections, supporting infrastructure that is unique to each application and robust processes for other operations such as mirror-to-truss assembly.

The desire to field large (i.e. greater than 10-meter diameter primary mirror) optical systems in space has been a dream of space scientists for many decades. Many concepts for such telescopes have been developed over the years, with one of the more recent examples being a 30-meter space observatory operating at the ultra violet-optical-near infrared (Hubble-like) wavelengths [52]. This concept is scalable (the primary aperture diameter can be varied) and relies on multiple launches to place the telescope elements in space and in-space robotics to assemble the elements and complete the telescope. Another recent concept for a large space telescope, with a 10-meter diameter primary mirror and operating in the ultraviolet-optical wavelengths, is also conceived to be assembled robotically in space [18]. In the early 1990s NASA completed extensive technology development for

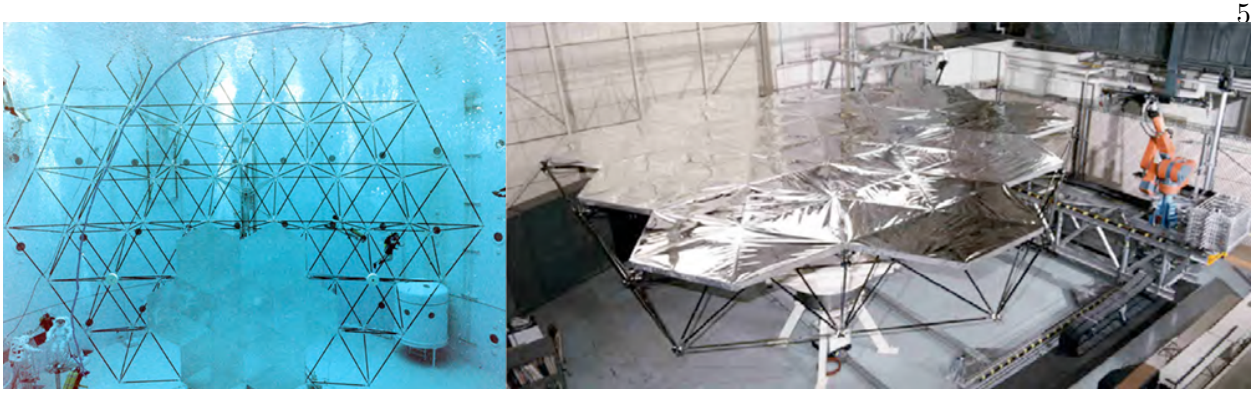


Figure 1.2: Left: the 84-node 14m-diameter telescope truss, which is the motivating example in this dissertation, was assembled by simulated EVA [66]. Right: an autonomous assembly experiment of a 102-strut truss at Langley Research Center [15].

a 20-meter diameter far-infrared space telescope that evolved into the precision segmented reflector [38, 44] as depicted in Figure 1.1.

The structural architectures for the large space telescopes discussed previously have many similar features, such as: segmented primary mirrors, where the mirror size is determined by either manufacturing or launch-vehicle shroud size constraints; large-area stiff and lightweight trusses which support the primary mirror segments; truss beams/towers/masts to support secondary mirrors and/or instrument packages; and large lightweight sun shields. The rationale and benefits for using trusses as the primary structure for many space applications (including telescopes) has been documented [49] and their high degree of structural performance has been treated comprehensively [47]. Based on this rationale, a large amount of design and technology development efforts have focused on lightweight space trusses. Preliminary design approaches for large high precision segmented reflectors are well established [48], forming the foundation for even more refined and detailed treatment of structural concepts and mechanics issues [39]. In what might be considered the culmination of the telescope mirror support truss development efforts, a precision truss structure was designed, fabricated, assembled and tested, ultimately validating the high surface precision, stiffness and strength that had been predicted [8].

As mentioned previously, one of the key impediments to achieving large space telescopes is

the assumption that the telescope must be launched as a complete system, subject to the inherent payload mass and volume limitations imposed by a single launch vehicle. Launch vehicle limitations and practical constraints that limit the likely size of a deployable telescope system (the JWST configuration for example) have been reported [38]. This reference also summarizes the rationale and benefits that accrue when a large space telescope is designed to be assembled on orbit, outlines a concept, estimates the performance and discusses assembling a 25-meter diameter space telescope. In addition to advances in telescope mirror support truss design, fabrication and structural performance, a great deal of progress has been made in assembling these structures both manually, by astronauts in Extra-Vehicular Activity (EVA), and robotically. EVA structural assembly development culminated in an experiment where a 14-meter diameter, doubly curved telescope truss, consisting of 315 individual struts, along with 7 (of the 37 total needed) mockup reflector panels was assembled in neutral buoyancy [66] as shown in Figure 1.2. In parallel research, methods were also developed to enable robotic telescope assembly. The robotic work culminated in the repeated autonomous assembly (and disassembly) of an $8m$ diameter truss structure, composed of 102 individual truss members along with 12 hexagonal panels [15], as shown in Figure 1.2.

1.2 Related Robotic Assembly Research

Robotic field assembly research can be categorized in two ways: research into identifying sequences that are geometrically possible for industrial applications, and research into assembly of structures made mostly of interlocking parts (with a few exceptions for amorphous construction). I did not find any experiments in the literature that used modifiable, stock components, nor did I find any that modeled assembly as a SLAM problem; instead, part errors and noises are abstracted away by self-correcting linkages and by simply ignoring error propagation.

1.2.1 Assembly Algorithms

Algorithms for mechanical assembly planning (via disassembly sequences) for problems in well-known environments such as assembly lines, and with few assembly robots, were explored in

the 80s and 90s, resulting in algorithms for finding fully-ordered sequences [23, 12, 53, 69, 41, 70], or using opportunistic assembly planning [19] when necessary. An overview of the challenges in mechanical assembly planning can be found in [20]. All of these approaches attempted to find a suitable order for assembly that avoids construction deadlock and assumed rigid bodies. In reality, structural forces and minor assembly errors often lead to situations where parts of a structure must be forced open to jam a new component in. Such algorithms do not consider these situations. Algorithms for distributed assembly are often simple and are only applicable to the specific class of structures under consideration, and include modeling the assembly process as a differential equation [3], using raster scanning techniques [42], or distributing the assembly of structures layer-by-layer through Voronoi diagrams [72]. Finding assembly sequences by reversing the disassembly sequences is used in many approaches, since disassembly sequencing with only geometric constraints does not require backtracking [23]. AND/OR graphs [23] assume that subassemblies can be made independently. Complete and correct algorithms cannot escape the worst case $O(n!)$ for enumerating all build sequences [24], rendering these algorithms useful only for structures with tens of parts, and making this problem NP-Hard. Partial ordering [12] may be used reduce the search space for assemblies, but is still exponential in the worst case. A* search with pruning [70] is shown to find optimal paths subject to identifying good heuristics. Non-directional blocking graphs [69] can find assemblies in polynomial time where assembly steps are valid if they can be moved into place without violating constraints. The difficulty of solving this problem made it fall out of fashion, but interest was renewed with the popularity of distributed robotics and stochastic methods, including genetic algorithms [40], and probabilistic roadmaps [60]. A robust distributed algorithm based on opportunistic disassembly sequencing is described in [22], which uses teams of robots to assemble structures while handling exceptions due to a wide range of failures, and relying only on a human operator when failures are beyond the scope of the assembly robots. Taking into account physical constraints such as structural stability and material properties into the build order has been shown in [46].

1.2.2 Robotic Assembly Experiments

More recently, collective robot assembly has seen a revival [67]. Quadrotor assembly is a recent and popular development; cubic truss structures are assembled in [42], a quadrotor assembly agent uses reinforcement learning to learn to handle unmodeled situations in [2], and [27] demonstrates control with a manipulator arm attached. Other approaches include truss climbing and assembling robots [13], termite-inspired swarm assembly robots [68], and a robot team that can build IKEA furniture in cluttered environments [30]. Mobile assemblers [71] used visual feedback to estimate and correct errors in assembly. The equal distribution and parallelization of the truss assembly task is shown and demonstrated in [5, 74]. With some exceptions such as amorphous assembly in [51], most of these methods rely on self-correcting, interlocking mechanisms, which would add extra mass and expense due to machining requirements, and we know of no such experiment that considers welding or cutting. Some recent, distributed large-scale assembly tasks have used common construction materials. Three different robots are shown to successfully dock a part to an assembly [54]. An experiment performed at NASA's Jet Propulsion Laboratory demonstrated the precision assembly of beams by a pair of cooperative robots using highly rigid motions to ensure precision [59, 58]. A major hurdle to overcome is precise assembly by mobile manipulators; peg-in-hole tasks are explored in [21, 14], but do not match a level of precision seen on the assembly line.

Large scale assembly experiments will require robots that can climb and traverse the structure. I envision that IPJRs, welding robots and materials could be transported by climbing robots, which has been extensively studied on trusses [64, 73, 10, 6] and unstructured terrain [7].

Many of these methods focus on the idea of swarm assembly, where practical concerns are mostly ignored in favor of demonstrating large scale swarming behavior and the equal distribution of tasks. The IPJR paradigm implicitly permits large scale swarms through the parallelization approach I describe in this thesis, but this is not the principal contribution of the paradigm. I believe that robotic construction in the field will be better served by attempting to build with typical

construction materials, followed by implementing the high level concepts proposed elsewhere.

1.2.3 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping [61], in which a robot is both mapping the environment and its own location within it, has been extensively researched and improved in recent years. Extended Kalman Filter-SLAM was an early [56], and still often used, variant; its major problems are the inaccuracy of the Extended Kalman Filter's Gaussian assumption and the computational cost of updating the estimates. Research has been motivated by reliably identifying and swiftly estimating the poses of a large number of features which often must first be recognized in images (especially if they were previously seen) and fed into the estimator [62, 50]. Range-only SLAM [29, 4] is very similar to the assembly SLAM methods in this thesis; it uses only range measurements, often between beacons in addition to range to the robot itself, but a problem is that multiple possible localizations are possible with only distance measurements, which is also true for truss structures.

The IPJR SLAM formulation is not a difficult problem requiring an improvement on the state of the art. The combined state space is $O(N)$, the landmarks have known correspondences, and there are not thousands of measurements coming in per second. My implementation of linear least squares and MLE-SLAM are offline SLAM problems in which all poses and measurements are considered, like with GraphSLAM [62], whereas my EKF-SLAM and UKF-SLAM implementations are online planners and maintain and update a covariance. Particle filters are very popular and used in algorithms like FastSLAM and FastSLAM 2.0 [50], but the good performance and time requirements of my other SLAM implementations did not necessitate trying the particle filter, nor did they require tactics to reduce the state dimensionality.

1.3 Intelligent Precision Jigging

In the Intelligent Precision Jigging paradigm, first introduced in [16], the allocation of responsibility for structural precision is transferred from the structural elements to the IPJRs. This is

achieved by allowing a group of IPJRs to precisely form individual cells within a truss, and thereby guide the placement and welding of the permanent struts. The IPJRs must be able to:

- Connect to a node at a location known precisely relative to the node center.
- Set and hold a precise distance between two nodes.
- Set and hold a strut that is provided by an external manipulator.
- Communicate with the other IPJRs in its group to coordinate cell formation.
- Communicate with the auxiliary manipulator in order to commence welding, and to request reconfiguration.
- Measure the distance between nodes with high precision before and after a weld.

I constructed three prototype IPJRs that each embodied the requirements for the paradigm. The results of experiments using the first two prototypes are summarized in Chapter 2, while the third prototype is the primary focus of this dissertation.

- Prototype 1 [36, 37], shown in the upper left of Figure 1.3, was a prebuilt triangle for constructing 2D trusses. The lengths of the edges were changed by Firgelli L-16 actuators, accurate to $0.5mm$. Trusses were constructed of wooden dowels that were cut by the external manipulator (for this experiment, a human) and glued to the wooden nodes.
- Prototype 2 [35], shown in the upper right of Figure 1.3, was another prebuilt triangle for constructing 2D trusses, but this time with a focus on autonomy and high precision materials. The edge lengths were controlled by Ultra-Motion actuators accurate to $8\mu m$, while the length measurements were performed by a Keyence IL-030 laser sensor, precise to $1\mu m$. The external manipulation was performed by the Lightweight Surface Manipulation System (LSMS) [17], which also welded the titanium struts to the titanium nodes.

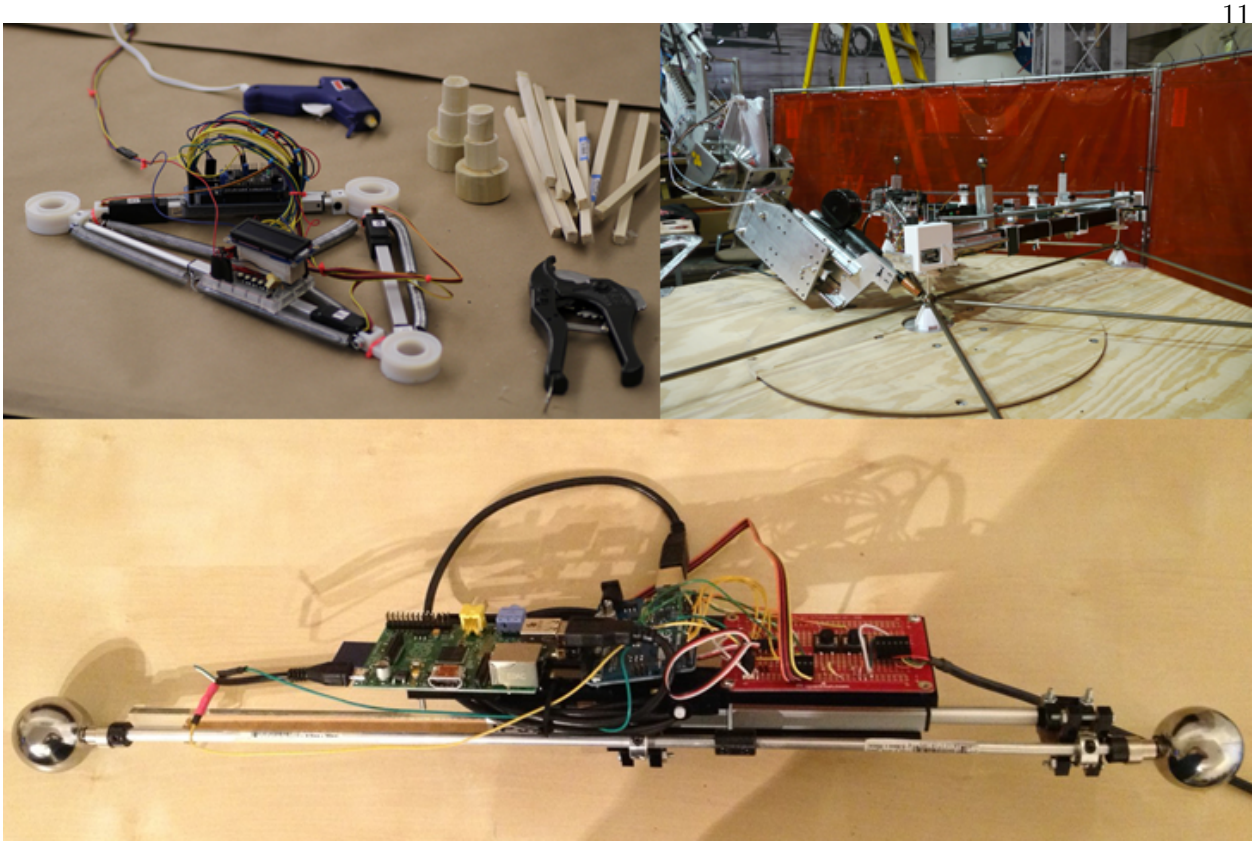


Figure 1.3: The three IPJR prototypes: (top left) assembled 2D wooden trusses, (top right) assembled 2D welded titanium trusses, and (bottom) assembled 3D aluminum trusses.

- Prototype 3 [33, 34], shown in the bottom of Figure 1.3, consists of several one-edge IPJR units, of which three are needed to construct 3D trusses. I used the same Firgelli actuators that were on prototype 1. Trusses were made of telescoping aluminum struts. The external manipulator (a human) attached and detached the IPJRs, and provided measurement feedback by a ruler.

Each prototype demonstrated the complete assembly of a number of truss structures. Prototype 2 came the closest to the vision of space telescope truss assembly; no direct human involvement was used in the experiment (a human did tele-operate the LSMS), the truss was made of stock titanium struts, which were welded to titanium node balls, and high precision actuation and sensing were included.

Neither prototype 1 nor prototype 2 used an error detection and correction loop. Prototype 2 had unmeasured and unmodeled biases which defeated the high precision actuation and sensing. The error of the assembly experiment using prototype 2 was three orders of magnitude greater than that of the actuators themselves. A fundamental problem was bias due to thermal expansion or structural deformation.

Prototype 3 was designed to include a method for detecting and correcting assembly errors. Welding, deformation under forces, and other unmodeled processes all contribute to assembly errors. One approach is to model forces induced by gravity, thermal expansion, and vibrations by using Finite Element Method software. However, I chose a different approach. Since assembly errors are irreversible, the strategy I chose was detect and correct. To do so, I grouped all of these physical processes together as a single process noise, and modeled this problem as a Simultaneous Localization and Mapping (SLAM) problem [61]. Assembly by IPJR is a SLAM problem for a few reasons:

- IPJRs build up a map of their environment using only local measurements (length measurements) of the landmarks (the nodes).
- Additional measurements between a given node and other nodes both early and late in the assembly process are **loop closures** and therefore reduce uncertainty.
- Each IPJR's position is also unknown, and is modeled as the mean of the two nodes it is attached to.
- The measurement update loop incorporates well known estimation functions such as the suite of Kalman Filters and the Maximum Likelihood Estimator.

Using SLAM, error propagation can be mitigated to such an extent that the error of each node is on the order of the process noise for a single strut, and does not grow as the assembly grows. This leads to the thesis statement.

1.4 Thesis Statement

A group of IPJR robots and external manipulators, using local measurements and a SLAM estimator, can collaborate to assemble truss structures made of simple, modifiable, commodity parts to an accuracy and precision necessary for the construction of large space telescopes.

1.5 Thesis Summary

This dissertation describes the theory and algorithms necessary for IPJRs to achieve high levels of precision in a closed loop assembly process, and documents the results of both simulated and physical experiments to validate the theory and algorithms.

Chapter 2 provides a full summary of the IPJR research up to, but not including, the work presented in this dissertation. This includes all of prototypes 1 and 2, and the early research on prototype 3.

Chapter 3 introduces the notation used in the rest of the dissertation, and defines the framework on which the theory is founded.

1.5.1 Assembly Sequences

Chapter 4 covers all of the topological assembly sequence algorithms (i.e. the ones that do not depend on the truss's Euclidean embedding). First, I derive expressions for counting the number of assembly sequences for a truss structure, establish upper bounds, and show that it is exponential. Then I describe an algorithm for finding the possible seed configurations for a structure (known as **starting triangles**), followed by an algorithm for identifying the nodes and struts that can be added to a given substructure. I then describe how to parallelize an assembly sequence, allowing multiple IPJR groups to build a structure in parallel. I then follow this up with an algorithm that describes how to find a random sequence, either any random sequence or a sequence that runs in parallel with the fastest time. Using all of the above, I describe an algorithm that can find central starting triangles. Then, I introduce the concept of an adjacent sequence, a necessary component

of local search algorithms that operate on the graph of complete sequences.

Finally, I discuss and simulate methods for estimating the number of assembly sequences for a given topology. First, I describe how to enumerate every sequence, which only terminates in reasonable time for the smallest structures. Then I describe a method that estimates the sequence count based on branching factor product. Then, for several classes of trusses made of cubic and tetrahedral-octahedral cells, I run the sequence count estimator and show that the algorithm is asymptotically correct for structures with known numbers of assembly sequences, the sequence counts are within the derived bounds, and the counts are exponential in node count.

1.5.2 Probability Model

Chapter 5 introduces the model of probability for truss structures. First, I define a full structure function with only strut lengths as parameters. I derive a model of the probability of a truss structure given that each of its struts has a Gaussian process noise, and no error correction is used. I then linearize the model around the mean. I derive the covariance matrix of the linear model as a function of the Jacobian matrix of the full structure function, and show how the covariance matrix trace is a proper choice for the minimizing metric. I show that the trace has useful properties, including node error additivity, permitting incremental algorithms. I argue that traces are minimized when struts are orthogonal to one another.

I then analyze a simple structure made of regular tetrahedra, which I call the "triple helix", and show that the total trace of each node increases cubically as the node count increases, and show that the mean derivative of a node with respect to a single strut increases quadratically as the node count increases. I follow this with an analysis of the accuracy of the linear covariance matrix model, and show that for precisions that may be used in on-orbit experiments, the error of the linear model is negligible.

Finally, I discuss measurements. Using the canonical form of multivariate Gaussian distributions, I derive a general measurement function and describe how the structure assembly can be estimated near-optimally given any kind of measurement, enabling the future use of global posi-

tioning systems such as the Vicon [1]. Then I discuss strut length measurements in more detail, showing how the covariance trace is reduced as a function of which strut is measured.

1.5.3 Optimization of Sequences

Chapter 6 describes a joint method to find a low-trace, locally minimal assembly sequence using a three-step approach; start from a centrally-located triangle, greedily assemble the structure to minimize the trace, then perform a local search on the space of assembly sequences adjacent to the greedy sequence, taking steps to the neighbor with the lowest trace repeatedly until the sequence is minimized. I test the algorithm and its subcomponents on a 45-node subset of the underwater telescope in Figure 1.1, and show that starting centrally is better than starting from any random triangle, greedy assembly far outperforms random assembly, and local search can reduce the greedy assembly further by taking several hops.

1.5.4 Assembly as SLAM with Estimation

Chapter 7 describes a general algorithm for incorporating SLAM and measurements in an assembly process, whether by simulation or by physical trials. I describe four estimators; the linear least squares algorithm based the canonical Gaussian form, the Extended Kalman Filter, the Unscented Kalman Filter, and the Maximum Likelihood Estimator. I simulate each of these on a 45-node subset of the underwater truss, an 18-node subset of the same truss, and a 20-node triple helix. I show that every one of these filters exceptionally outperforms the open loop process, and for small variances and small structures, they are essentially equivalent. With the triple helix simulations, I show that the Maximum Likelihood Estimator is the best estimator. I also show that, when SLAM is used, the covariance trace difference between two sequences is reduced substantially, eliminating the need for a globally optimal sequence.

1.5.5 Physical Experiments

Chapter 8 presents the results of implementing the MLE-SLAM assembly algorithm on prototype 3, the 3D IPJRs. I first simulate the assembly of a 10-node subset of the underwater truss, comparing the global optimal sequence with the median optimal sequence. I then describe the results of 3 physical trials with open loop assembly, and 3 physical trials with MLE-SLAM assembly, showing that the physical trials match the expected results, and that MLE-SLAM can successfully overcome unmodeled physical biases.

1.5.6 Discussion and Conclusion

Chapters 9 and 10 wraps up the key results from the previous chapters chapters. I also present a case study, returning to the initial problem: can a space telescope truss be made of stock components to $10\mu m$ precision using off-the-shelf hardware, and conclude that the hardware I used in prototype 2 was nearly sufficient if SLAM had been used. I then discuss the future of the IPJR paradigm.

Chapter 2

Early IPJR Experiments

This chapter describes my previous work with the first two prototypes, and discusses the lessons I applied to the research presented in this dissertation.

2.1 Prototype 1

Prototype 1 consisted of a triangle of three actuators and was designed for precise assembly of wooden trusses. Figure 8.1 shows the IPJR prototype, assembly tools, struts, and nodes. Wooden dowels, acting as struts, were roughly cut and bound with glue by a human user. This system permitted the construction of triangles with varying dimensions, and relied on an external agent to handle the struts and provide mobility to the jig.

The IPJR triangle's actuation consisted of three Firgelli L16-140 150:1 -P actuators with built-in distance sensors in the form of potentiometers. This allowed each jig to change its length from roughly 287mm to 428mm. The control hardware was an Arduino Mega 2560 with an Adafruit motor shield, a 2×16 character LCD, and five buttons. Each actuator potentiometer had a stated accuracy of 0.5 mm. Potentiometer readings were converted to a 10 bit signal (1024 discrete values) using the Arduino's analog-digital converter. Both actuator and joints had free play of about 1 mm, which were improved by attaching compressive springs to each edge and across the triangle span. Each joint had a 25.4 mm diameter cylindrical hole to align with a node on the structure. A Keyence IL-030 analog laser sensor and a Keyence IL-1000 amplifier unit were used to characterize the accuracy and repeatability of the actuators. The IL-030 has an accuracy of 1 μm. The actuators

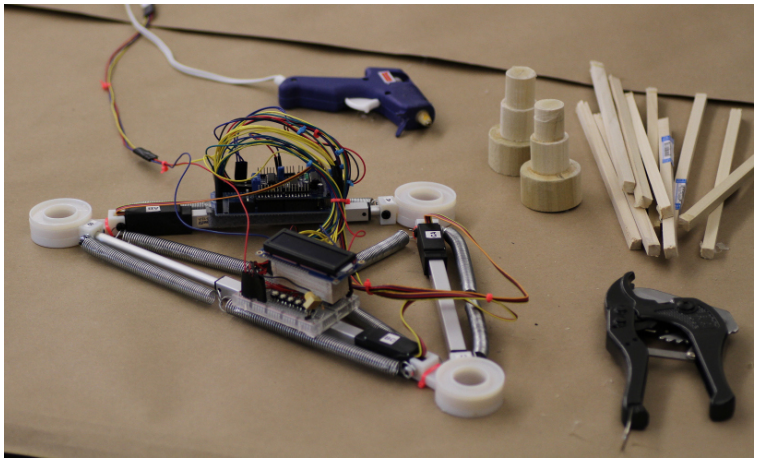


Figure 2.1: Prototype 1 in right triangle configuration. The controller is attached to the top-left edge, the button interface and LCD are attached to the bottom edge. The three joints are printed parts. Also in the picture (clockwise from top): the glue gun used to bond struts to nodes, nodes that form the vertices of the triangular cells, struts that form the edges of the triangular cells, and the low-precision cutting tool used to cut the struts to approximately the correct size. No other measurement, bonding, or cutting tools were used in the construction of the structure.

were set to a specific length L, the sensor was tared, the actuator was a random distance in ± 13 mm, then back to L, and the offset was using the IL-030.

2.1.1 Experiments

Struts were cut from 12.7 mm square dowels and nodes were constructed from 25.4 mm height segments of 50.8 mm, 31.75 mm, and 25.4 mm diameter cylinders respectively, forming a 76.2 mm stepped cone. Struts attach to the bottom step, the IPJR triangle rests on the second step, and distance measurements are based on the center of the top step. The topmost 25.4 mm cylinders fit in the ring-like joints on the IPJR triangle. When connecting struts to nodes, the struts are cut to length and are glued to the tops of the bottom cylinder.

2.1.1.1 Square

The square truss experiment was designed to test the reliable assembly of a large square subdivided into four right triangles. The design has five nodes: one at the center, and four at the

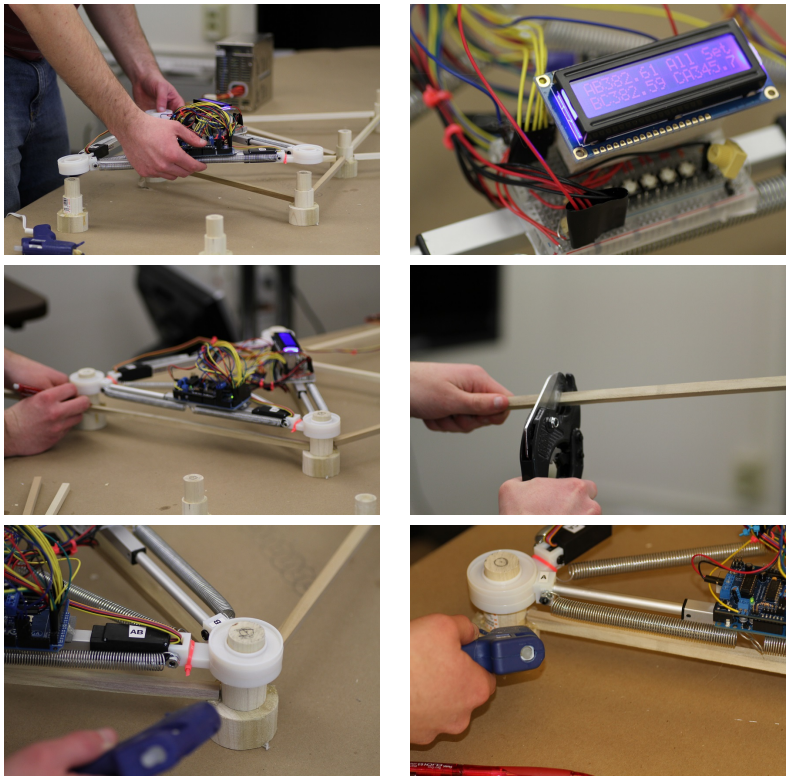


Figure 2.2: Assembly of a cell on the ring truss. Top left: after the IPJR triangle sets its lengths, the agent sets it down on nodes. Top right: the IPJR displays the lengths of the edges to be glued in this step. Center left: the agent marks the location to cut the strut. Center right: the agent cuts the strut. Bottom left and right: the agent glues the strut to both nodes.

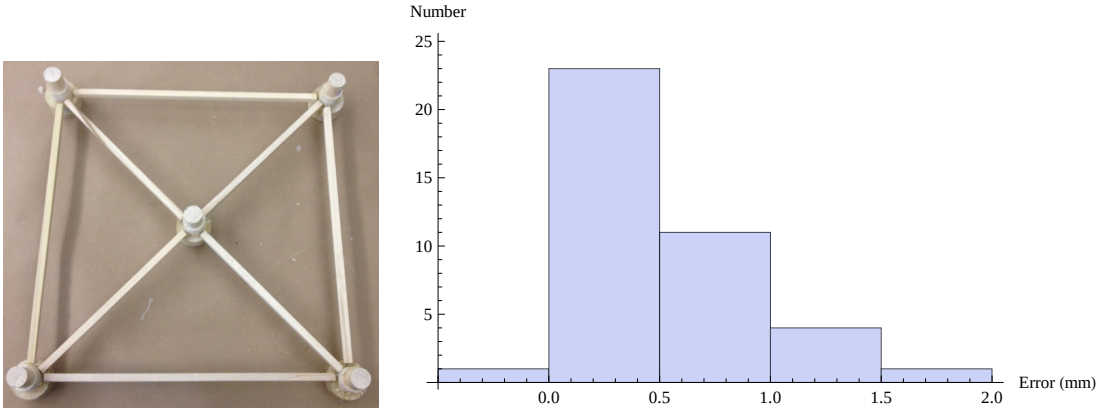


Figure 2.3: Left: image of a completed square truss. Right: histogram of the measured edge length errors for five square truss assembly tests, in which the desired lengths are 295 mm and 417.2 mm. The mean error is 0.416 mm.

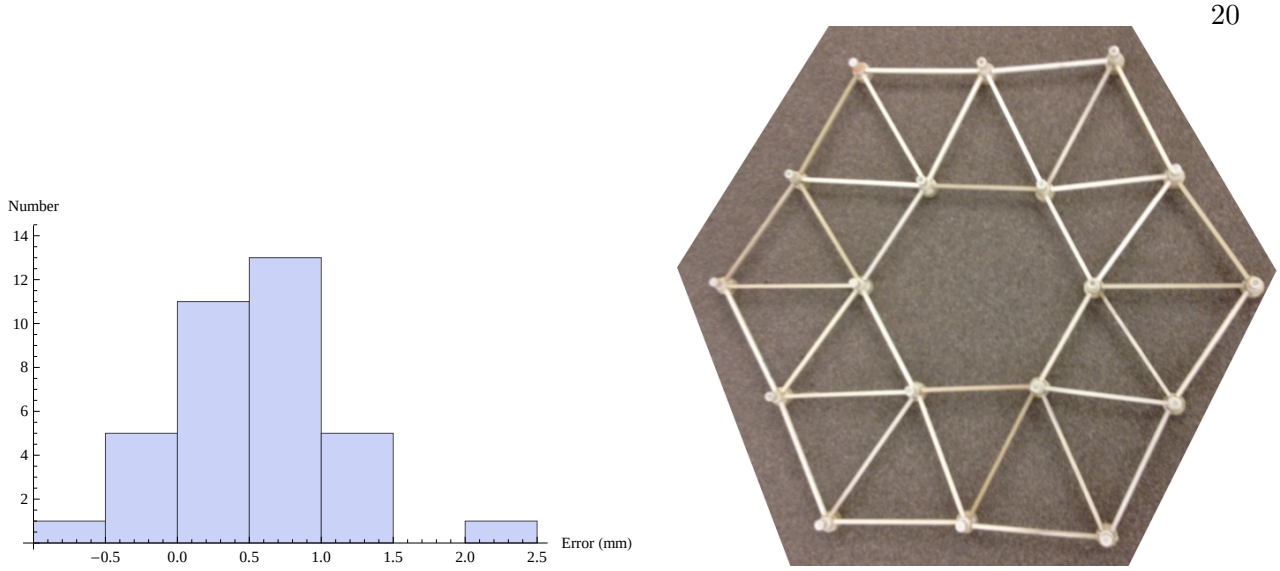


Figure 2.4: Left: Histogram of measured length errors of the physical experiment of the irregular ring truss. The mean error is 0.508 mm. Right: Result of the ring truss assembly experiment.

corners. There are eight struts: four on the outside and four connecting the corners to the middle, as shown in Figure 2.3. The interior node-to-node distances should be 295 mm, and the exterior node-to-node distances should be 417.2 mm.

The measured edge length errors are shown in Figure 2.3. Measurements were recorded in increments of 0.5 mm; therefore, each measurement is ± 0.25 mm. The tests show that the edges are longer than the nominal length. The average error is 0.416 mm, with a standard deviation of 0.459 mm. Most cases were within 0.5 mm desired length: only 8 of the 40, including every edge 41, had a greater error than the accuracy of the actuators' potentiometers. Edge 34 on test 5 is longer than it should be because the cut strut was slightly larger than the gap, and exerted forces on either end after the glue dried.

2.1.1.2 Ring Truss

Space telescope optical benches will not be composed of equilateral triangles; the geometry will be mapped to fit the surface of a parabola. The irregular ring truss test reflects this mapping and demonstrates the versatility of the robot.

The average edge length error was 0.508 mm, as shown in Figure 2.4, and the standard deviation was 0.601 mm. Prior to adding the final two triangles, the three unplaced edges (0,5), (0,17), and (6,17) were measured, to calculate the overall error accumulation of the structure. The resulting gap errors were 0.5 mm, 0.8 mm, and 1.4 mm. Since the largest of these is within the range of other edge errors, no forces were required to reduce the gap to an acceptable level for bonding. Figure 2.4 shows the completed structure.

2.1.2 Analysis of Prototype 1 Experiments

I observed a positive length bias. A positive length bias would result in the first three right angles to be slightly less than 90 degrees, and the final angle having to be larger to close the structure, causing the increase. On the ring truss, the wide range of errors on the other edges indicate its large errors canceled out in the accumulation. The calibration may have been altered by the springs exerting different forces for the right triangle, causing the error. The low precision strut measurement device may have contributed. Errors in calculating node centers could have occurred, which were the basis for determining strut length. Finally, the small number of tests and the free play between nodes and struts could mean that these results are not indicative of the long term behavior, although both the square truss and the ring truss demonstrate the positive error bias. The size of the ring truss required that the experiment be laid out over several laboratory tables, whose surfaces were offset by a few millimeters. The five square truss tests and the one ring truss test were completed without having to induce stress on the trusses to close the final cells. These tests gave valuable insights on how to improve the IPJR. The calibration, performed only on near-equaliteral configurations represented by identical potentiometer voltages, became incorrect by up to 0.3 mm when the IPJR was irregular. This was not due to free play in the actuators, but was due to the potentiometers themselves being affected by internal stress. The potentiometer calibrations are not independent in practice. Length changes on one IPJR edge might induce subtle stresses that slightly change the readings without noticeably changing the actual edge length, leading to different offsets on the potentiometer output as a function of the length of the other actuators. The largest such

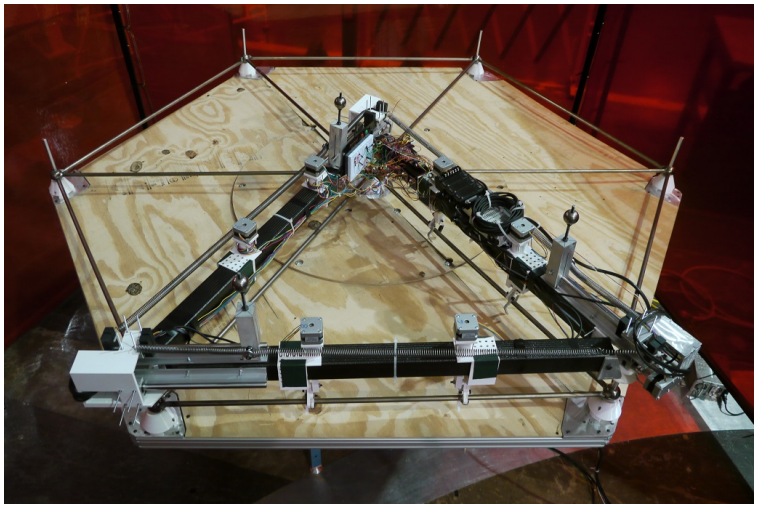


Figure 2.5: The IPJR prototype is shown resting on the finished titanium truss on which the nominal distance between the tops of the node posts is 1.002 m.

error was 0.3 mm, and is likely one of the largest sources of error in the experiments. Although this source of error was not modeled in the simulations, it might be countered by performing a multi-variate calibration that takes the overall state of the system into account.

2.2 Prototype 2

The second prototype constructed a 2D truss made of 7 titanium nodes and 12 titanium struts, arranged in a hexagonal lattice of six cells, representing a simplified optical bench for a space telescope. The cells were nominally equilateral triangles with a node-to-node distance of 1.002 m.

Starting with a prebuilt cell called the "kernel", which is used both as calibration target as well as a reference for the IPJR to start, assembly proceeded in five stages, repeated once for each additional cell:

- The LSMS lifted the IPJR off the truss and positioned it over a canister containing the new struts and nodes
- The IPJR captured the required struts and nodes from the canister

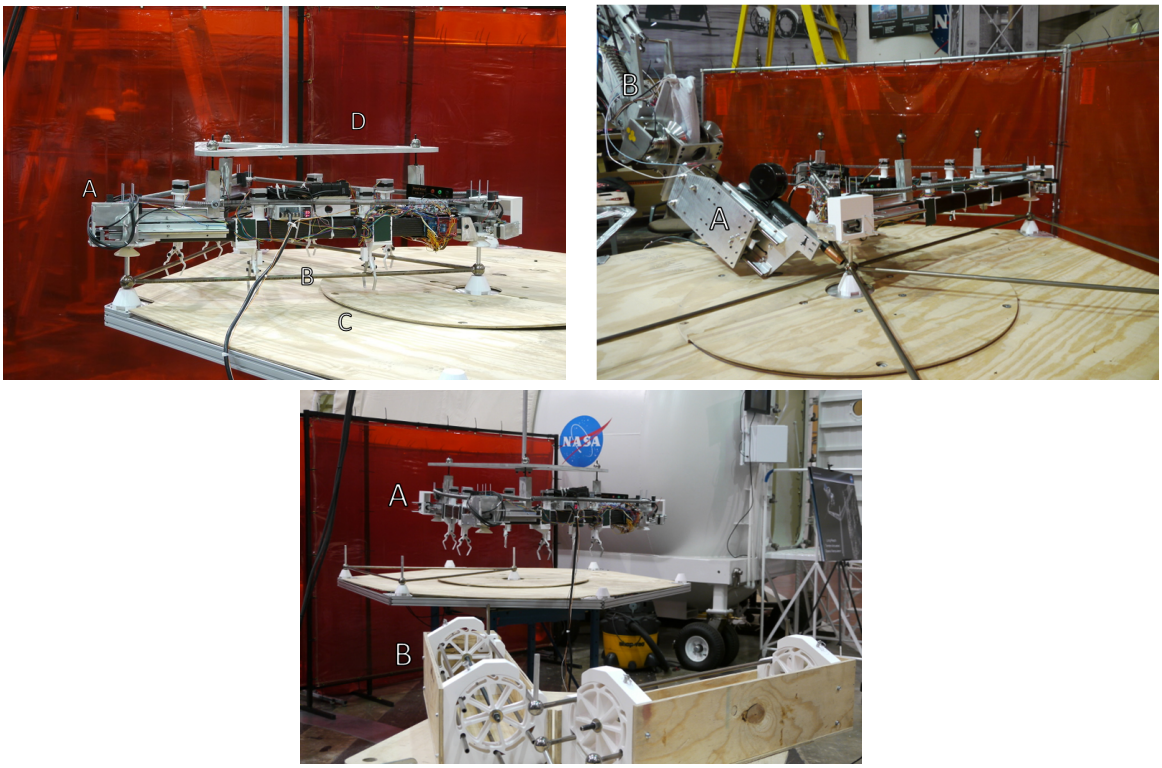


Figure 2.6: Top Left: The IPJR (A) rests on top of the kernel (B) at the start of the experiment, which in turn rests on the turntable (C); the lifting plate (D) is used to lift the IPJR. Top Right: The welding end effector (A), attached to the LSMS forearm (B), is preparing to weld strut 16 to node 1. Bottom: The LSMS is transporting the IPJR (A) to the strut canister (B) in the foreground.

- The LSMS positioned the IPJR on the assembly site
- The IPJR adjusted its lengths to the desired truss dimensions, positioning the new node appropriately
- The LSMS welded the new struts and node to the truss

To accomplish these steps, the IPJRs required the capability to capture the truss components, the LSMS required a welding end effector and a lifting mechanism, and the canister needed to provide the new components in such a way that the IPJR could capture them. Additionally, the IPJR needed the ability to compensate for the discrepancy between the kernel and the nominal 1.002 m edge lengths, thus necessitating actuators to change the IPJR edge lengths. Figure 2.6

shows the tools used to complete the experiment.

The materials for the truss and the IPJR were chosen with attention to mitigating precision errors. Titanium was chosen for its favorable low coefficient of thermal expansion of 8.5 $\frac{\mu m}{m^{\circ}C}$, an important consideration for telescopes on orbit subject to thermal expansion. Struts were 12.7 mm-diameter hollow tubes of 961.9 mm length, which were welded to node balls 38.1 mm in diameter. Affixed to the top of each node ball was a 12.7 mm-diameter, 107.95 mm-tall aluminum post. Each post represented a mounting point for a segment of the reflector. The node post positions were the metric for determining the precision of the truss.

The Lightweight Surface Manipulation System (LSMS) is a long-reach manipulator designed to operate on planetary surfaces, intended to manipulate large objects in preparation for a manned landing [17]. It is 4.25 m tall and has a reach of 8.5 m. By itself, the LSMS provides precision on the order of mm, which is not sufficient to construct trusses at the precision demanded by space telescopes. An off-the-shelf arc welding gun was integrated into an end effector attached to the LSMS wrist. A lifting plate end effector gave the LSMS the ability to lift the IPJR by grasping three attachment points on the IPJR.

The limited maneuverability of the LSMS requires a turntable to provide the final degree of freedom necessary for controlled placement of the IPJR and the welding gun. In this experiment, a human operator rotated the turntable as necessary, but future experiments will utilize a motorized turntable that will be under control of the IPJR.

The canister presented new struts and nodes for the IPJR to capture while the LSMS positioned the IPJR over the canister. It was built and positioned in such a way that the struts and nodes are positioned roughly in the shape of the cell they are supposed to build. This enabled the IPJR to grab all of the required elements at once and carry them over to the assembly site. As with the turntable, this version required a human operator, but future versions will utilize motors controlled remotely by the IPJR.

The Intelligent Precision Jigging Robot in this paper was nominally designed (Figure 2.7) to align the tops of the node posts (where a mirror would be attached) to within 5 μm of an adjustable

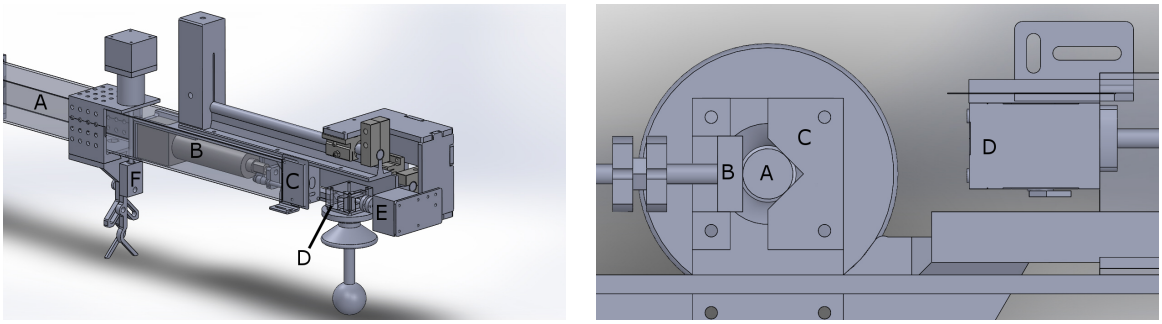


Figure 2.7: Left: The composite tube (A) contains the Ultra Motion actuator (B) and the IL-030 (C); the IL-030 measures its distance from the target plate (D), which also captures node posts using a stepper motor (E); another gripper (F) captures struts. Right: A top-view cutaway shows the node post (A) being pushed by the stepper motor (B) into a right-angled surface on the target plate (C), which is also used for distance sensing by the IL-030 (D).

length in the range of 0.987 to 1.013 meters. To meet the 5 μm requirement, the IPJR uses Ultra Motion D-Series actuators, with a specified 7.9 μm motion per step, enabling any desired length to be within 3.95 μm of a step, and a 50.8 mm stroke length. To test repeatability and to calibrate, each edge had a Keyence IL-030 laser distance sensor to sense the length of the IPJR edge by measuring the extension of the Ultra Motion actuator. The IL-030 has a repeatability of 1 μm, but a small operational range of 20-45 mm, which limited the range of the IPJR.

The IPJR is a triangle consisting of three identical edges, with the exception of auxiliary hardware attached to one of the three edges. Each edge is a mechanical linkage between two components, the main body and the node gripper module. The main body of each edge is a composite tube, a material chosen for its favorable thermal expansion properties over most metals. An Ultra Motion actuator extends the node gripping module away from the main body. Two rails with bearings are used to prevent bending moments on the Ultra Motion actuators, which can cause them to fail under lateral loads of just 13 N.

The node gripping module consists of a funnel used for guiding a node post while the IPJR is being lowered onto the canister, a target plate for capturing node posts precisely, and a stepper motor to push the node post into the target plate. The IL-030, attached to the main body, measures the edge's extension by measuring its distance to the target plate. Attached to the tube are two

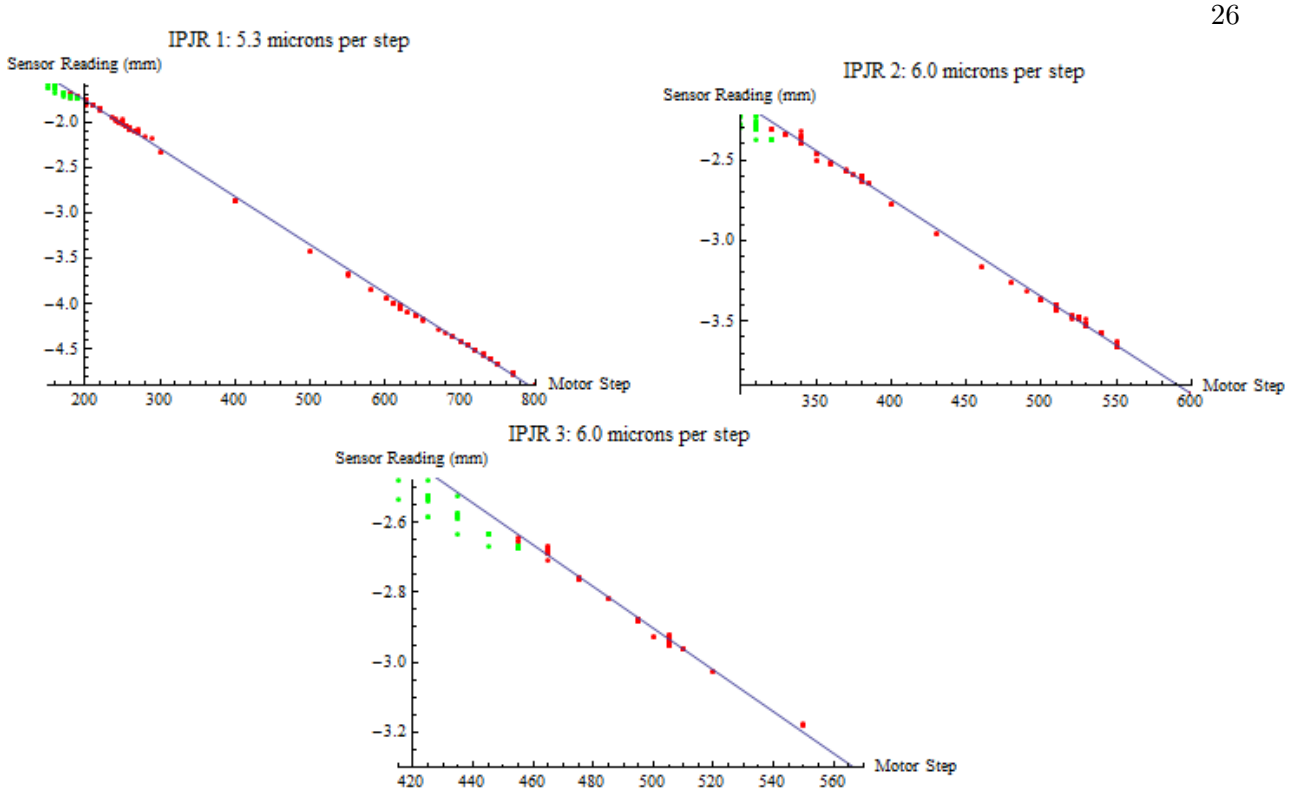


Figure 2.8: Results of the laser calibration, in which the steps from full retraction were mapped to the sensor readings. The sensors were arbitrarily zero-shifted, but only the relevant readings were of interest. The nonlinear region is shown with green points and the quasi-linear region with red points.

grippers for lifting and holding the struts.

To eliminate free play within the entire IPJR, springs are attached between the lifting plate posts, imposing a compressive force on each edge. Thus, when each Ultra Motion is commanded to go to a step, there will be no loose components to interfere with the measurement.

2.2.1 Calibration

To calibrate the IPJR on the kernel, touch sensors were applied to the node posts on the kernel and the target plates. Each edge length had some influence on whether or not contact was made on the others, so the edges could not be calibrated one at a time. This amounted to a 3-dimensional search for a maximum contact-free length for all three actuators such that the slightest increment in

Table 2.1: Node coordinates and distance error after assembly (m).

x	y	z	||error||
0.	0.	0.	0.
0.998072	0.	-0.00004572	0.0039282
0.497226	0.864428	-0.0000127	0.00503288
-0.501045	0.863276	-0.00078486	0.00454969
-0.998114	-0.00196461	0.00079756	0.00442672
-0.498673	-0.866141	0.00156972	0.00323913
0.499244	-0.865118	0.001143	0.00337018

one edge would result in contact on that edge only. The calibration of the linear actuators involved finding a least squares fit on the laser measurements, with results shown in Figure 2.8. However, at shorter lengths, as the IPJR was closing in on the maximum contact-free lengths, the mutual effect each edge had on the other resulted in significant nonlinearity (the green points). The slope was only calculated for lengths sufficiently distant from the nonlinear regions (the red points). The distance per step was found to be 5.3, 6.0, and 6.0 μm respectively. Note that these values differ from the specified Ultra Motion step distance of 7.9 μm. Reasons may include the compressive force from the springs and the node posts when the IPJRs are overextended, thereby contracting the parts of each edge, and the IL-030s being slightly off-axis.

2.2.2 Analysis of Prototype 2 Experiments

The assembly of the truss took place over six hours, with the full truss being completed by the IPJR and the LSMS. The maneuvering and welding by the LSMS contributed most of the duration. To save time, the strut canister was not used for the final 4 cells, since it was shown to work at least once and otherwise did not affect the outcome of the experiment. Upon completion, the experimental setup was disassembled and the truss was measured. The measurements are shown in Table 2.1 and Figure 2.9.

The data show that, compared to the desired 1.002 m separation, all of the nodes are less than 1 m apart. The largest deviation is on node 3, on the kernel, at 5.03 mm, and among the nodes placed by the IPJR, the largest deviation is on node 4 at 4.55 mm. The edge lengths

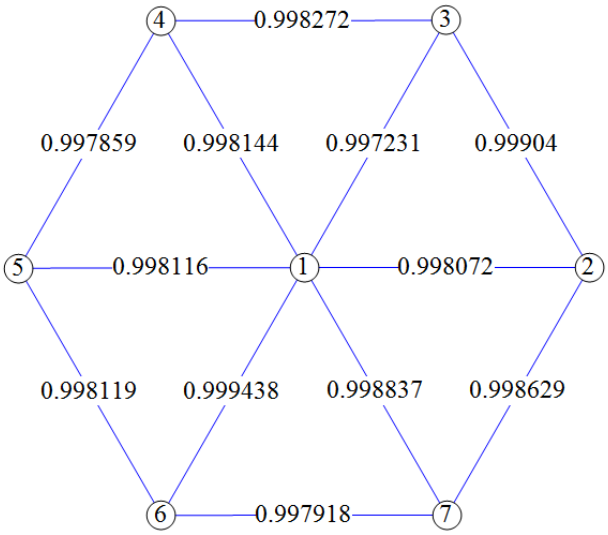


Figure 2.9: The measured distances between the nodes (m) on the final truss structure.

show a range of 2.207 mm, with a standard deviation of 593 μm. The data not only significantly differs from the desired structure, but the kernel substructure differs as well, and the relationship is not simply a factor of scale. Strut bowing was observed, indicating tensions and compressions within the truss, possibly arising from post-weld thermal contraction. The processes leading to these errors could include factors such as temperature gradients (including during and after the welding process), internal forces in the structure induced by the cooling of the truss, an imprecise turntable, imprecise node posts, forces induced by the IPJRs while it was gripping the nodes, a biased calibration, and the node posts' deviation from vertical.

Although the experiment was a successful demonstration of an IPJR working with the LSMS to assemble and weld a truss structure, the approach needs modification to meet the precision requirement. As intended, the experiment revealed several issues which will lead to modifications. The use of high precision actuators is promising, but it will rely on a much more accurate measurement than the IL-030 sensors provided. The sensors measured a small gap between the main body and the node gripper body, which only gave a measurement of the length of the gap, and can only

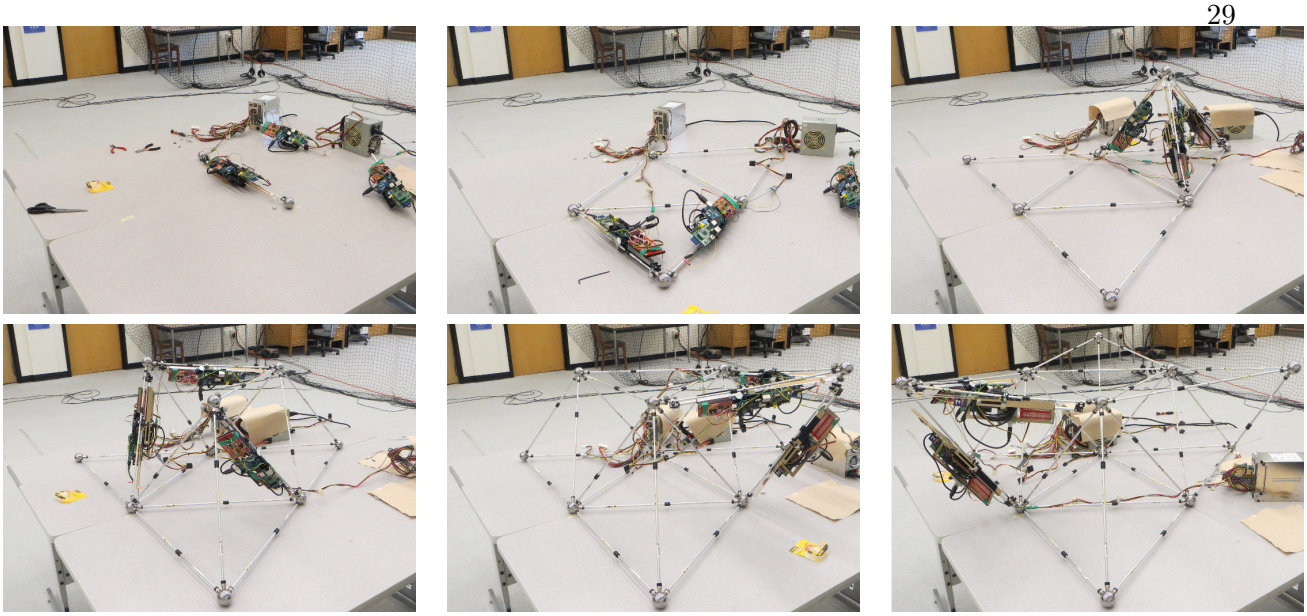


Figure 2.10: In reading order, the first nodes placed requires only one IPJR, the other grounded nodes require 2 IPJRs, and the subsequent nodes require 3 IPJRs to be placed. The final frame shows the finished assembly, prior to the removal of the IPJRs.

scale reliably to the IPJR as a whole assuming the structure components are rigid bodies, and the laser is perfectly aligned with the strut axis. The IPJR had a great deal of compliance, was built imperfectly, and was subject to other nonlinearities in the calibration. The calibration showed that even if the welding process introduced no errors, the 95th percentile radius from the desired node position is 248 μm.

2.3 Prototype 3

My initial experiment involving prototype 3 and SLAM evolved directly into what is presented in this thesis, and is nearly identical. The differences are:

- An assembly sequence was not found *a priori*, but was planned at each step. It was a form of greedy assembly, where instead of picking the node that adds the lowest trace, the choice was to build on the base that has the lowest trace.

- The Extended Kalman Filter was the only estimator tried.

- I allowed triangular cells in addition to the starting triangle, with the assumption that the Z-axis was fixed, $z = 0$.

- Maximum Likelihood Estimation was used only to estimate the final structure.

To test the validity of the EKF algorithm on real hardware, I performed two assemblies of the small truss with the IPJRs and the aluminum-strut, steel-node structure (Figure 2.10), using the minimum trace heuristic. To collect measurements each step, in lieu of an on-board laser distance sensor planned for the future, the human operator measured the distance between node balls. The EKF algorithm corrected for the hidden biases due to the various factors affecting the parameters of the IPJRs: imprecise connections to the aluminum tubes, bent tubes, IPJR extension and contraction hysteresis, and deflection due to gravity.

The results of the two trials, bounded by the 95th percentile accuracy error of the maximum likelihood estimator, are shown in Figure 2.11. The mean errors are $1.7mm$ and $2.7mm$, with a confidence interval of $1.2mm$. Both are an improvement over open loop assembly, even when physical biases are excluded. The second trial has the largest individual node error at $7mm$. The two physical assembly experiments with the EKF assembly algorithm performed far better than the open-loop control cases, even with the conservative MLE confidence interval of $1.2mm$. Considering that the struts and IPJRs were not made equally, this shows that precise assembly can be made possible by lower quality components through error detection and correction. The second physical trial shows a trend of growing error: this is attributed to two unlocked struts not sliding as expected, deforming the bottom layer — violating the $z = 0$ assumption. I believe that the error growth would have stopped had there been more nodes to add. During the experiments, the IPJRs occasionally rebooted, and struts broke, but the trials never failed. One IPJR suffered a voltage error and was incapacitated. Since I had two spare IPJRs, the experiments were not hindered. The accuracy and ability to continue despite hardware failure justifies the use of simple robots when assembly requires precision and accuracy.

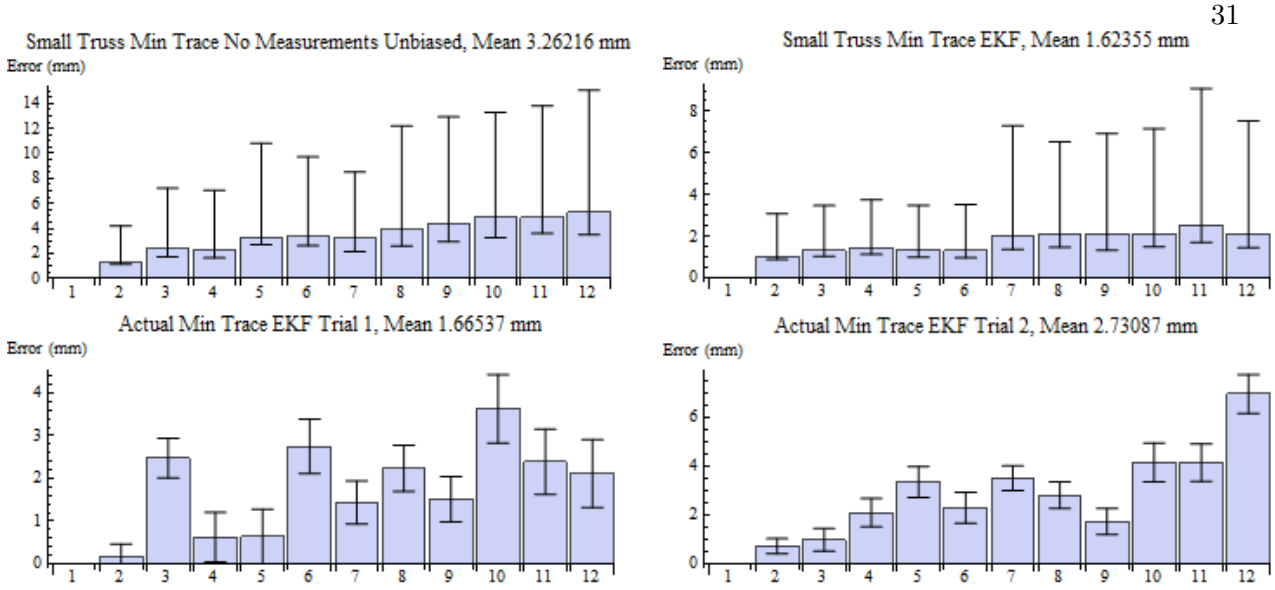


Figure 2.11: Top left: Simulated results of open loop assembly, assuming no hidden biases. Top right: Simulated results of EKF assembly. Bottom row: the estimated errors of the two physical experiments, bounded by the MLE confidence interval.

2.4 Lessons Learned

Prototype 1 was a proof of concept; the purpose of the experiment was to demonstrate incremental assembly of a wooden truss, and nothing more. However, I did learn a few important lessons. One is that free play between parts is a major source of error and must be eliminated. Another is that calibration of the Firgelli actuators' potentiometers is unreliable, and more direct measurements are necessary. While I used the Firgelli actuators again, I did not use their potentiometers in the SLAM algorithm.

Prototype 2 was another proof of concept, this time using precise hardware and fully robotic manipulation. I expected the structure to be precise to within well under $1mm$. The results of this experiment showed that calibration alone could not hope to achieve anywhere near the $10\mu m$ necessary for space telescope trusses, despite the precise hardware used in the IPJR. I determined that the most likely cause of the error was thermal stress on the structure due to the cooling of the spot welds contracting each edge. With error detection and correction, this could have been

noticed and the remainder of the assembly could have been adjusted to compensate for the change. The large errors in the prototype 2 experiment were the major motivation for me to examine error correction routines.

Prototype 3 showed that SLAM overcame hidden biases and enabled assembly to be done more precisely, leading to me trying three additional estimators. However, allowing triangular cells after the starting triangle led to errors when the $z = 0$ assumption was inaccurate. I fixed this problem by requiring that every node after the starting triangle has 3 IPJRs attaching it to the base.

Chapter 3

Introduction and Definitions of Incremental Truss Assembly

A truss structure is an undirected graph $G = (V, E)$ embedded in Euclidean space. Each vertex $i \in V$ has an associated structural **node** X_i with a 3-dimensional position $(x_i, y_i, z_i)^T$. The number of nodes in the structure, and the size of V, is N. As nodes are spherical and struts can be attached anywhere, the orientation of each node is not important. Each edge $E_{ij} = \{i, j\}$, is associated with two vertices and corresponds to a **strut** of the structure. The length of the strut between two nodes i and j is $||X_i - X_j||$.

The **starting triangle** is the name given to the first three nodes assembled. Their positions form the orthonormal basis of the truss structure; the first node is fixed to the origin, the second node varies only on the X-axis, and the third node varies only on the XY-plane. This fixes the frame of reference to the structure itself, and does not require an external frame. A starting triangle is denoted Δ.

3.1 Placement of Nodes by Strut Length Adjustment

IPJRs assemble trusses by attaching new nodes to nodes already on the structure. The new node is the **floating node** to indicate that it is not fixed until welding is performed, and the nodes to which the floating node is attached are the **base triple**, in which each node is a **base node**. In this paper, the floating node is indexed as f, and the base nodes $(i, j, k) = B_f$. In the model, one node is attached at a time by using exactly as many IPJRs as the node has degrees of freedom: 1 IPJR for 1 dimensions (reserved only for the X-axis node) 2 IPJRs for 2 dimensions (the XY-plane

node), and 3 IPJRs for 3 dimensions (all other nodes).

X_f is a found as a function of the base triple it is attached to, and the struts connecting the float to the bases:

$$X_f = \mathcal{F}_f(X_{B_f}, L_{B_f}) \tag{3.1}$$

In the specific case of tetrahedra, the distances between base nodes i, j, k and floating node f are the lengths that the IPJRs must satisfy, and are found as:

$$\begin{aligned} L_{i,f} &= ||X_i - X_f|| \\ L_{j,f} &= ||X_j - X_f|| \\ L_{k,f} &= ||X_k - X_f|| \\ X_f &= \mathcal{F}_f(X_i, X_j, X_k, L_{i,f}, L_{j,f}, L_{k,f}) \end{aligned} \tag{3.2}$$

The function $\mathcal{F}_f()$ can be found by algebraically solving the system for X_f:

$$\begin{aligned} X_f &= \mathcal{F}_f(X_i, X_j, X_k, L_{i,f}, L_{j,f}, L_{k,f}) \\ &= X_i + \frac{X_{j,i}\left(L_{i,f}^2 - L_{j,f}^2 + X_{j,i}.X_{j,i}\right)}{2X_{j,i}.X_{j,i}} \\ &\quad + \left(\frac{X_{j,i} \times X_{k,i}}{\sqrt{X_{j,i} \times X_{k,i}.X_{j,i} \times X_{k,i}}} \times \frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}\right) \\ &\quad \left(\frac{\left(L_{j,f}^2 - L_{i,f}^2\right) X_{j,i}.X_{k,i} + X_{j,i}.X_{j,i}\left(L_{i,f}^2 - L_{k,f}^2 - X_{j,i}.X_{k,i} + X_{k,i}.X_{k,i}\right)}{2X_{j,i}.X_{j,i}\sqrt{X_{k,i}.X_{k,i} - \frac{(X_{j,i}.X_{k,i})^2}{X_{j,i}.X_{j,i}}}}\right) \\ &\quad + \left(\frac{L_{i,f}\sqrt{1 - \frac{\left(L_{i,f}^2 - L_{j,f}^2 + X_{j,i}.X_{j,i}\right)^2}{4L_{i,f}^2 X_{j,i}.X_{j,i}}} X_{j,i} \times X_{k,i}}{\sqrt{X_{j,i} \times X_{k,i}.X_{j,i} \times X_{k,i}}}\right) \\ &\quad \left(\sqrt{1 - \frac{\left(\left(L_{i,f}^2 - L_{j,f}^2\right) X_{j,i}.X_{k,i} + X_{j,i}.X_{j,i}\left(-L_{i,f}^2 + L_{k,f}^2 + X_{j,i}.X_{k,i} - X_{k,i}.X_{k,i}\right)\right)^2}{\left((X_{j,i}.X_{k,i})^2 - X_{j,i}.X_{j,i}X_{k,i}.X_{k,i}\right)\left(-2L_{j,f}^2\left(L_{i,f}^2 + X_{j,i}.X_{j,i}\right) + \left(X_{j,i}.X_{j,i} - L_{i,f}^2\right)^2 + L_{j,f}^4\right)}}\right) \end{aligned} \tag{3.3}$$

While I present $\mathcal{F}$ here as-is, the full derivation of $\mathcal{F}$, and the derivatives of $\mathcal{F}$ with respect to both node positions and strut lengths, can be found in Appendix A.

Equation 3.3 has two solutions based on the ordering of i, j, k, each a mirror image of the other with respect to the base. Determining which is correct is determined by the right hand rule: the path from i to k is counterclockwise, and the position of f is in the direction of the rotation vector defined by the path using the right hand rule. Thus, B_f is an ordered triple.

Using more IPJRs than is required results in an overdefined system, which in the physical world would result in large stresses on the structure unless the extra struts passively adjusted their lengths. All truss structures require $3N - 6$ struts to be stable (the -6 term corresponds to the reduced degrees of freedom of the starting triangle). When a node has more adjacent nodes than the base nodes used for assembly, the extra struts (and the structure itself) are **redundant**. IPJRs also attach to redundant struts, but passively allow the struts to adjust since their lengths are a function of the assembly struts' lengths.

Choosing the best base nodes for each floating node is central to the problem in this article. The edges in E that are actively set by IPJRs are labeled E_A, and the redundant struts are labeled E_R:

$$\begin{aligned} E_A &\subseteq E \\ \|E_A\| &= 3N - 6 \\ E_R &= E \setminus E_A \end{aligned} \tag{3.4}$$

The full structure $\mathbf{X}$ is a vector containing all of the node positions. The length of the vector is the same as the length of the actively assembled struts, $3N - 6$, which is also the total degrees of freedom in the structure:

$$\mathbf{X} = (x_2, x_3, y_3, x_4, y_4, z_4 \ldots x_N, y_N, z_N) \tag{3.5}$$

The full vector of lengths L_E contains the positive, real-valued lengths for the assembly struts

E_A, and is ordered in the assembly order:

$$\begin{aligned} L_E &= (L_{2,1} \dots L_{last}) \\ \|L_E\| &= 3N - 6 \end{aligned} \tag{3.6}$$

Prior to the completion of the structure, a substructure with K nodes is denoted V_K, E_K. E_K also includes passive struts, where $E_K \subset (E_A \cup E_R)$.

3.2 Assembly Sequences

An **assembly sequence** A is an N-tuple consisting of pairs of assembly **steps**; each pair is defined as $\langle f, B_f \rangle$. A sample assembly sequence may be:

$$A = [\langle 1, () \rangle, \langle 2, (1) \rangle, \langle 3, (1,2) \rangle, \langle 4, (1,2,3) \rangle, \langle 5, (2,3,4) \rangle \dots \langle f, (i,j,k) \rangle] \tag{3.7}$$

If two nodes i, j can be added to a structure independently of each other, then their relative ordering is irrelevant. For example, if an assembly sequence produces two branches, nodes added to one branch can be added independently from nodes on the other branch. This allows for parallelization: multiple groups of IPJRs could work on separate branches of structures independently. If one assumes all independent additions to a structure are made simultaneously, and each addition takes one unit of time, a parallel sequence can be defined, in which each floating node f is assigned a value t_f, where t_f is the earliest **time** at which the node f can be assembled, and is also the minimum number of nodes that need to be assembled before it can be assembled (the ancestor count) plus 1:

$$t_f = 1 + \max(t_i : i \in B_f) \tag{3.8}$$

In other words, a floating node can only be assembled after the last node of its base triple has been assembled. The starting triangle is assembled at $t = 1, 2, 3$; the nodes whose base is the

starting triangle can be assembled at at $t = 4$, and so on. A can be partitioned into a **parallel assembly sequence** P by t values, each one corresponding to an assembly layer:

$$P = (A_{t=1}, A_{t=2} \dots) \tag{3.9}$$

Two assembly sequences with different but equivalent permutations can be compared by their parallel orders:

$$A_i = A_j \text{ if } P_i = P_j \tag{3.10}$$

An assembly order A_i is **faster** than A_j if $\|P_i\| < \|P_j\|$. A **fastest assembly order** for a starting triangle Δ is an $A_{fastest}$ such that $\|P_{fastest}\| \leq \|P_{all}\|$ for Δ. A **central triangle** is a triangle $\Delta_{central}$ such that $\|P_{\Delta_{central},fastest}\| \leq \|P_{\Delta_i,fastest}\|$ for Δ_i.

3.3 Measurement model

The IPJR model makes few assumptions about what kinds of measurements are available. While global position sensors, cameras, and other sensors will likely be used in an on-orbit experiment, they are not necessary, and were never available for my experiments. Instead, each IPJR has the capability to measure the length of the strut it is currently attached to (pre- or post-weld). They include measurements of both the assembly struts and the passive struts, as long as the IPJRs are attached. Measurements $M_{i,j}$, between nodes i and j, are taken after welding is complete and X_i, X_j are permanent. I assume the error to be normally distributed with mean $\|X_i - X_j\|$, and **measurement variance** σ_M^2:

$$M_{i,j} \sim \mathcal{N}(\|X_i - X_j\|, \sigma_M^2) \tag{3.11}$$

An IPJR builds up a map of its surroundings with local measurements, including its own position (expressed as the two nodes it is connected to), making this a SLAM problem in which the robot deploys its own landmarks.

When redundant struts are added between a floating node and extra nodes in the structure not in the base triple, IPJRs passively adjust. These IPJRs are assumed to be capable of measuring the passive struts after they have been welded. Thus, the set of measurements M defines the full set of edges E for the substructure of size K:

$$M_K = \{\mathcal{N}(\|X_i - X_j\|, \sigma_M^2) : (i,j) \in E_K\} \tag{3.12}$$

The extra measurements of the redundant edge set E_R enhance the effect of loop closure in the MLE-SLAM algorithm. Without these extra measurements, the MLE algorithm finds the structure that independently maximizes the likelihood of each strut based on only its own measurements; this is because the number of measurements is equal to the number of degrees of freedom. The extra measurements force an overdefined system, where each strut estimate is a function of all of the other struts noises and measurement noises.

3.4 Structure Sparsity

Truss structures commonly have an upper limit on the number of edges emanating from a strut: for example, Pratt trusses, Warren trusses, and the tetrahedral-octahedral honeycomb truss in [66] have nodes that connect to only a limited number of neighbors. Such structures have a number of edges constant in the number of vertices. If $D_v N = E$ is the expression for the number of edges, D_v is the **vertex half-degree**, or half the average number of struts touching a node:

Lemma 1. $\|E\| = O(N) = D_v N$

Proof. Given a physical node of finite size, the surface of the node is big enough for only a finite number of connections to struts, which does not increase as the total number of nodes and struts increases. Thus, the number of edges is $\|E\| = O(N)$, and truss structures are **sparse**. □

Sparse graphs are better implemented as adjacency lists to conserve space and time.

Chapter 4

Assembly Sequences

In this chapter, I describe algorithms that operate on the topological properties of trusses without regard to their Euclidean embedding in $\mathbb{R}^3$. The runtimes herein depend on Lemma 1, which states that all truss structures are sparse graphs, and the number of struts is linear in the number of nodes.

4.1 Number of Assembly Sequences for Non-Redundant Structures

To properly analyze the number of assembly sequences for the general case of redundant structures, I first analyze the simpler case of non-redundant structures.

Lemma 2. *A structure with $3N-6$ nodes has exactly one possible base triple for each node.*

Proof. If a node in such a structure has multiple base sets, the number of struts connecting to the node is the union of the base sets:

$$\|E_f\| = \text{number of struts adjacent to node } f = \left\| \bigcup_{b \in B} b \right\| \geq \|b \in B\| \tag{4.1}$$

The extra edges are redundant; therefore this is a contradiction. □

Lemma 3. *A structure that has exactly $3N-6$ struts has a number of assembly sequences equal to the number of possible starting triangles, which is $O(N)$.*

Proof. Lemma 2 shows that each node has exactly one base configuration; therefore, for a given starting triangle, a non-redundant structure has only one assembly sequence, and that the number of assembly sequences is equal to the number of starting triangles. □

4.2 Number of Assembly Sequences for Redundant Structures

Redundant structures — far more common than non-redundant ones — will have a number of non-redundant substructures not exceeding the number of $3N - 6$ combinations of the E edges:

$$\|E_r\| \leq \binom{\|E\|}{3N-6} \tag{4.2}$$

The inequality is required due to the fact that not all combinations of $3N - 6$ struts are possible to build: for example, such a set might divide a structure into two distinct structures, or one or more nodes may not have enough struts as degrees of freedom.

Recall that D_v is **half** of the average vertex degree (number of struts) of a node in a truss; Equation 4.2 becomes:

$$\|E_r\| \leq \binom{D_v N}{3N-6} \tag{4.3}$$

The number of subsets of E satisfying $\|E_{r,n}\| = 3N - 6$ is $O((D_v N)!)$ from the following expression for the number of combinations:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{4.4}$$

This can be immediately reduced to $O(2^{D_v N})$ when noting that:

$$2^n = \sum_{k=0}^{n} \binom{n}{k} \tag{4.5}$$

However, even that is too large given that it considers all values of k, not just one. The bounds can be tightened further with the following inequality described in [57]:

$$\frac{\left(\frac{(r+1)^{r+1}}{r^r}\right)^m}{4mr} < \binom{m(r+1)}{m} < \left(\frac{(r+1)^{r+1}}{r^r}\right)^m \tag{4.6}$$

Lemma 4. *The number of substructures of length* $3N-6$ *is* $O\left(\left(\frac{1}{27}\left(D_v-3\right)^{3-D_v}D_v^{D_v}\right)^N\right)$.

Proof. Setting $m = 3N-6$ and $m(r+1) = D_vN$:

$$r = \frac{D_vN}{3N-6} - 1 \tag{4.7}$$

The right side of Inequality 4.6 then becomes:

$$\left(\left(\frac{ND_v}{3N-6}\right)^{\frac{ND_v}{3N-6}}\left(\frac{ND_v}{3N-6}-1\right)^{\frac{ND_v}{6-3N}+1}\right)^{3(N-2)} \tag{4.8}$$

As n increases, $\frac{D_vN}{3N-6}$ asymptotically approaches $\frac{D_v}{3}$ and $3N-6$ approaches $3N$, reducing this to:

$$\left(\frac{1}{3}\left(D_v-3\right)^{1-\frac{D_v}{3}}D_v^{\frac{D_v}{3}}\right)^{3N} = \left(\frac{1}{27}\left(D_v-3\right)^{3-D_v}D_v^{D_v}\right)^N \tag{4.9}$$

The left side is then:

$$\frac{1}{4(3N)\left(\frac{D_v}{3}\right)}\left(\frac{1}{27}\left(D_v-3\right)^{3-D_v}D_v^{D_v}\right)^N = \frac{1}{4D_vN}\left(\frac{1}{27}\left(D_v-3\right)^{3-D_v}D_v^{D_v}\right)^N \tag{4.10}$$

Let $K = \frac{1}{27}\left(D_v-3\right)^{3-D_v}D_v^{D_v}$. Inequality 4.6 can be stated as:

$$\Omega\left(\frac{1}{N}K^N\right) = \|E_r\| = O(K^N) \tag{4.11}$$

□

The final step is to show that, when considering all starting triangles, the time complexity is asymptotically unchanged.

Theorem 1. *The number of assembly sequences for a structure is* $O((K+\epsilon)^N)$ *where* ϵ *is an arbitrarily small positive real number.*

Proof. With $O(N)$ starting triangles, the number of assembly sequences for every starting triangle is $O(NK^N)$. However, I use the same method in the previous proof to show that $N^cK^N = O((K+\epsilon)^N)$ for any positive c. The limit of $\frac{N^cK^N}{(K+\epsilon)^N}$ must approach 0 as $N \to \infty$. Taking the logarithm results in:

$$c\log(N) + N\log(K) - N\log(K+\epsilon) \tag{4.12}$$

The term $c\log(N)$ vanishes, resulting in a linear equation which goes to $-\infty$ if $\epsilon > 0$. When converting back to the exponential form, the limit becomes 0. □

I have shown that the number of assembly sequences is exponential. The remainder of this dissertation is devoted to finding ways to search for and approximate optimal solutions, thereby overcoming the exponential nature of assembly. The following chapter delves further into this topic.

4.3 Identifying Starting Triangles

For a structure in which a starting triangle has not been defined, the possible starting triangles must be found. For a set of edges E, this is equivalent to solving the 3-clique problem; the best algorithms are $\Omega(E^{3/2})$ in the worst case [9]. However, for sparse structures, the number of triangles is reduced since every node has a constant number of neighbors.

Lemma 5. ***AllTriangles*** *is $O(N)$, returning $O(N)$ triangles.*

Proof. The algorithm iterates over all vertices, automatically giving it N steps. On the inside of the loop, all pairs of neighbor nodes are considered, which is $\binom{2D_v}{2} = 4D_v^2 - 2D_v$ pairs. For each of the neighboring nodes n_j, its own list is searched twice, once for each of the starting node i and the other neighbor n_k. Since this is an adjacency list, each of the four searches take $O(2D_v)$ in the worst case. The total number of operations is $32D_v^3 - 16D_v^2$. However, since D_v is constant in N, this is $O(1)$, thus $O(N)O(1) = O(N)$. □

Algorithm 1 An algorithm that searches for all starting triangles by looking at all of the pairs of adjacent nodes to each node, and seeing if the other nodes are also adjacent. A brute force algorithm, $O(\|V\|^3)$, but for sparse graphs (such as truss structures) is $O(\|V\|)$. The $n_1 > i, n_2 > n_1$ ensures that duplicates are not added.

AllTriangles(V, E)

Require: V, E in adjacency list format

$\Delta \leftarrow \emptyset$
for $i \in V$ **do**
  for $n_1, n_2 \in E_i : n_1 > i, n_2 > n_1$ **do**
    if $n_2 \in E_{n_1}$ **then**
      $\Delta \leftarrow \Delta \cup \{\{i, n_1, n_2\}\}$
return Δ

Algorithm 2 The possible next nodes algorithm looks at all the nodes that are adjacent to the set of currently made nodes.

PossibleNextNodes$(V_{made}, V_{candidates}, E)$

$V_{possible} \leftarrow \{j : \{i, j\} \in E, i \in V_{made}, j \in V_{candidates}\}$
$S_{next} \leftarrow \emptyset$
for $f \in V_{possible}$ **do**
  $V_{f,allbases} = \{b : \{f, b\} \in E, b \in V_{made}\}$
  $S_{next} \leftarrow S_{next} \cup \{(f, B) : B \subseteq V_{f,allbases}, \|B\| = 3\}$
return S_{next}

4.4 Possible Next Nodes

A partially built structure has a number of possible next additions to make, but determining which ones are valid requires finding all the possible tetrahedral apexes that can be made given the nodes that already exist. Algorithm 2 does not permit nodes that have more or fewer than 3 base nodes, but can be trivially modified to do so. It is important to note that a tetrahedron apex does not need to rest on a base whose nodes are all connected to one another: both this algorithm and Equation 3.3 do not distinguish between connected and disconnected base nodes.

Theorem 2. ***PossibleNextNodes*** *takes $O(N)$ time and returns $O(N)$ float-base pairs.*

Proof. A good implementation will use adjacency lists, in order to optimally collect the possible edges and nodes, which is $O(D_v V_{made})$ due to going through each D_v-length list for each node. Collecting all base nodes for a given float node takes $O(D_v) = O(1)$ time, and iterating all 3-subsets takes $O(D_V^3) = O(1)$ time. When multiplied for all $O(V_{possible})$ nodes and added to the

time for possible edges and nodes results in $O(V_{made} + V_{possible})$ time and $O(V_{made})$ possible nodes. On average, $\|V_{made}\| = \|V_{possible}\| = \frac{N}{2}$, making this algorithm $O(N)$ in time and in next node count. □

4.5 Parallelization

Finding the parallellized assembly sequence P is an $O(N)$ operation made possible by casting the problem as a topological sorting of a directed acyclic graph. Algorithm 3, **ParallelizeAssembly**, is such an implementation.

Theorem 3. *Finding the parallel ordering P of A takes $O(N)$ time.*

Proof. Partitioning an assembly sequence into P, Algorithm 3, can be performed by considering a directed acyclic graph formed by edges such that $i \to j$ if i is a base of j. Then, after topologically sorting the DAG, which is $O(V + E) = O(N)$, the longest path can be found with weights equal to 1, which is also $O(N)$ [55]. The longest path solves for Equation 3.8. Then, the nodes can be iterated over and added to the t-slot in P equal to $i_{distance}$, which takes $O(N)$ time. All told, finding a parallel partition and determining the parallel degree D_p is $O(N)$.

□

Having a parallel assembly order P is important because of the equivalency of any of the numerous permutations of A that have the same float base pairs. An algorithm to enumerate over all assembly sequences would do better to consider such a sequence only once.

4.6 Single Random Sequence

Given a set of nodes and struts, a single random sequence can be found by determining which float-base pairs can be added each step, and randomly choosing one. The parameter Q can be set to find either any random sequence, or a random sequence that is as long as the fastest parallel sequence for that starting triangle. When looking for a fastest sequence, each node's assembly time t_f will be constant; the only random component is which base is chosen.

Algorithm 3 An $O(N)$ algorithm for partitioning an assembly sequence A (which is not necessarily sorted) into an ordered list in which the order's index is a time t that, for each assembly step in that set, is the length of the sequence to get to step.

ParallelizeAssembly(A)

$P \leftarrow ()$
$O_{sort} \leftarrow ()$
$V_{dag}, E_{dag} \leftarrow \{(i \rightarrow f) : \forall (f, B_f) \in A, \forall i \in B_f\}$
for all $i \in V_{dag}$ **do**
  $i_{discovered} \leftarrow False, i_{processed} \leftarrow False, i_{distance} \leftarrow 0$
for all $i \in V_{dag}$ **do**
  DagVisit($i, V_{dag}, E_{dag}, O_{sort}$)
for all $i \in O_{sort}$ **do**
  for all $j \in E_i$ **do**
    if $i_{distance} + 1 > j_{distance}$ **then**
      $j_{distance} \leftarrow i_{distance} + 1$
$P_t = \{(f, B) : (f, B) \in A, f_{distance} = t\}$
return P

DagVisit($n, V_{dag}, E_{dag}, O_{sort}$)

if $n_{discovered} = True$ **then**
  Not a DAG
if $n_{processed} = False$ **then**
  $n_{discovered} \leftarrow True$
  for all $i \in E_n$ **do**
    DagVisit($i, V_{dag}, E_{dag}, O_{sort}$)
  $n_{processed} \leftarrow True$
  $n_{discovered} \leftarrow False$
  Push n **to head of** O_{sort}

Algorithm 4 To find a random sequence for a structure V, E, start by choosing a triangle, then, while the structure is incomplete, select a float-base pair from the set of possible next nodes. If at any point there are no float-base pairs for an incomplete structure, a sequence does not exist.

RandomSequence(V, E, Δ_{start}, Q)

$P_{partial} \leftarrow (\Delta_{start})$
while $\sum_t \|P_{partial,t}\| < \|V\|$ **do**
  $V_{made} \leftarrow \{f : f \in P_{partial}\}$
  $S_{next} \leftarrow$ **PossibleNextNodes**($V_{made}, V \setminus V_{made}, E$)
  if $S_{next} = \emptyset$ **then**
    return No Assembly Sequence
  if $Q =$Fastest **then**
    $L_{next} \leftarrow \{(i, B_{random}) : \forall i \in S_{next}, (i, B_{random}) \in \{(i, B) : i \in S_{next}\}\}$
  else if $Q =$Any **then**
    $L_{next} \leftarrow \{\text{RandomChoice}(S_{next})\}$
  $P_{partial} \leftarrow P_{partial} \cup (L_{next})$
return $P_{partial}$

Theorem 4. ***RandomSequence*** *terminates with a valid sequence, or determines that no sequence exists, in $O(N^2)$ time, and can be optimized to run in $O(N)$ time.*

Proof. The **RandomSequence** algorithm, for a starting triangle, must take $O(N)$ assembly steps, each time calling **PossibleNextNodes**, which is $O(N)$. Likewise, for Q =Fastest, the L_{next} list of additions to the layer is constructed from an iteration through S_{next}, which is $O(N)$ in length (if Q =Any, a single random choice is constant time). Therefore, finding a random sequence is $O(N^2)$.

A simple tweak to how **PossibleNextNodes** is used can reduce this to $O(N)$ by searching for neighboring nodes of only the latest nodes added, and adding these to S_{next}, which would be maintained from step to step. Since no steps would be found more than once, S_{next} grows to $O(N)$ size over the entirety of the while loop, amortizing it to $O(1)$ per step.

Likewise, L_{next} would only iterate through the newest steps, making the iteration also $O(1)$ when amortized. Since creating an assembly sequence requires N steps at an absolute minimum, the tweaked **RandomSequence** is optimal at $O(N)$. □

4.7 All Central Triangles

Starting centrally has an intuitive explanation. For example, assembling a truss in a zig-zag fashion results in very large errors due to the long chain of error propagation for each node. Starting from the center and proceeding outward in concentric circles is likely to be better. We encode this heuristic by finding a starting triangle that minimizes the number of steps needed to assemble the most distant node. This is analogous to finding a **central vertex** in a graph. The distinction is that a node cannot be reached on the assembly trajectory unless at least three of its adjacent nodes are in place. To find the set of central triangles, first all triangles must be enumerated, then a fastest sequence must be generated for each triangle to determining how many time steps are needed.

Each starting triangle will have a number of fastest sequences, but how fast they are (that is, the length of $P_{fastest}$ for that starting triangle) will vary: a centralized triangle will have a

Algorithm 5 Central starting triangles are identified by finding out which starting triangles produce the **fastest** among fastest sequences, and is $O(N^2)$ with an optimal implementation of **RandomSequences** and **PossibleNextNodes**

AllCentralTriangles(V, E)

```
1: Δ_all ←3-Cliques(V, E)
2: P_fastest ← ∅, Δ_fastest ← ∅
3: for all Δ_i ∈ Δ_all do
4:    P_i ← RandomSequence(V, E, Δ_i, Fastest)
5:    if P_fastest = ∅ or ‖P_i‖ < ‖P_fastest‖ then
6:       P_fastest ← P_i, Δ_fastest ← {Δ_i}
7:    else if ‖P_i‖ = ‖P_fastest‖ then
8:       Δ_fastest ← Δ_fastest ∪ {Δ_i}
9: return Δ_fastest
```

shorter $P_{fastest}$. The length of $P_{fastest}$ can be found by randomly generating a sequence using **RandomSequence**, with the argument Q =Fastest. The algorithm **AllCentralTriangles**, shown in Algorithm 5, requires **RandomSequence** to be called for all triangles to determine which ones produce the shortest parallel sequences.

Theorem 5. ***AllCentralTriangles*** *runs in $O(N^2)$ time, returning $O(N)$ triangles.*

Proof. The first call is to **AllTriangles**, which returns $O(N)$ triangles in $O(N)$ time for sparse truss structures. Each of the $O(N)$ triangles calls **RandomSequence**, which is $O(N^2)$. However, using the same tweak in Theorem 4 to optimize **RandomSequence**, it is $O(N)$, and **AllCentralTriangles** is $O(N^2)$. □

4.8 Adjacent Sequences

The neighboring sequences for a given sequence is generated by **AdjacentSequences**, shown in Algorithm 6. Two sequences are adjacent if one of the following holds:

- A_i and A_j have the same assembly edge set E_A, but different starting triangles.
- A_i and A_j differ by a single assembly step $\langle f, (i, j, k)\rangle$.

The set of sequences with identical E_A and different starting triangles is found first, and saved as A_Δ. For the reduced set E_A, or any set E with exactly $3N - 6$ edges, only one assembly

Algorithm 6 The set of adjacent sequences to A is found by looking at all the sequences generated by the other starting triangles with the same edges, and the other orders found by changing a single float base pair. For the latter, any choice must be checked for validity by constructing the parallel version.

AdjacentSequences(A, V, E)

1: $A_{adj} \leftarrow \emptyset$
2: $E_A \leftarrow \{(b, f) : (f, B) \in A, b \in B\}$
3: $\Delta_{all} \leftarrow$ **AllTriangles**(V, E_A)
4: $A_\Delta \leftarrow \{$**RandomSequence**$(V, E_A, \Delta_i, \text{Fastest}) : \Delta_i \in \Delta_{all}\}$
5: **for all** $(f, B) \in A$ **do**
6: $\quad V_{independent} \leftarrow \{i : i$ does not depend on $f\}$
7: $\quad S_{f,others} \leftarrow$ **PossibleNextNodes**$(V_{independent}, \{f\}, E) \setminus \{(f, B)\}$
8: $\quad$ **for all** $(f, B_{other}) \in S_{f,others}$ **do**
9: $\quad\quad A_{f,other} \leftarrow ((i, B_i) \in A : i \neq f) \cup (f, B_{other})$
10: $\quad\quad A_{adj} \leftarrow A_{adj} \cup \{A_{f,other}\}$
11: $A_{adj} \leftarrow A_{adj} \cup A_\Delta$
12: **return** A_{adj}

sequence exists, so **RandomSequence** will only return that sequence. Not all triangles will have a sequence; when **RandomSequence** returns no assembly sequence, nothing is added to A_Δ for that triangle.

The set of sequences with one altered step is found next. For every step (f, B_f) in A, all other possible bases B_{other} are considered that do not violate the assembly sequence. The set $V_{independent}$ is the set of all vertices that can be assembled when (f, B) is removed from A, and is the set from which all base triples will be drawn. The full set of base pairs for f, $S_{f,others}$, is found by calling **PossibleNextNodes** with $V_{independent}$ and $\{f\}$, and removing the current step (f, B). The set A_{adj} is the full set of assembly sequences found by replacing (f, B) with each member of $S_{f,others}$. The union of A_{adj} and A_Δ is then returned.

Theorem 6. ***AdjacentSequences*** *runs in $O(N^2)$ time and returns $O(N)$ sequences.*

Proof. **AdjacentSequences** can only return $O(N)$ sequences because there are only $O(N)$ possible changes. The sequences with different starting triangles number only $O(N)$, and the swapped sequences are $O(N)$ for a similar reason: each node is a part of only $O(1)$ triangles, resulting in $O(1)$ swapped steps per node. Each call of **PossibleNextNodes** is only $O(1)$ because only the node in question is examined. However, **AdjacentSequences** runs in $O(N^2)$ time, since it calls

Algorithm 7 This algorithm finds all assembly orders (parallelized to eliminate duplicates) by keeping a queue of previously discovered partial orders, popping an order, adding each possible node to the order, and pushing all the new partial orders onto the queue. Since multiple assembly orders may have the same parallel order, each new order is checked against the queue to determine if it is already there before it is added.

AllSequences(V, E)

Require: V, E in adjacency list format

$\Delta \leftarrow$ **AllTriangles**(V, E)
$P_{all} \leftarrow \emptyset$
for $\Delta_i \in \Delta$ **do**
  $P_{partial} \leftarrow (\Delta_i)$
  $Q_{partial} \leftarrow (P_{partial})$
  while $\|Q_{partial}\| > 0$ **do**
    $P_q \leftarrow Q_{partial}.pop()$
    $A_q \leftarrow \{(f, B_f) : P_t \in P_q, (f, B_f) \in P_t\}$
    $V_{made} \leftarrow \{f : (f, B_f) \in P_q\}$
    if $V_{made} = V$ **then**
      if $P_q \notin P_{all}$ **then**
        $P_{all} \leftarrow P_{all} \cup \{P_q\}$
      Continue
    $V_{notmade} \leftarrow V \setminus V_{made}$
    $S_{next} \leftarrow$ **PossibleNextNodes**$(V_{made}, V_{notmade}, E)$
    if $S_{next} = \emptyset$ **then**
      return No Assembly Sequence
    for $(f, B_f) \in S_{next}$ **do**
      $A_n \leftarrow A_q \| ((f, B_f))$
      $P_n \leftarrow$ **ParallelizeAssembly**(A_n)
      if $P_n \notin Q_{partial}$ **then**
        $Q_{partial}.push(P_n)$
return P_{all}

RandomSequence N times. □

4.9 All Sequences

The algorithm for finding all sequences is described here, but is of cursory interest because the number of assembly sequences is $O(K^N)$, and is therefore only useful to enumerate the number of sequences for very small structures. Still, its existence is useful in verifying that the sequence count estimation algorithm is correct for small structures.

I have thought of two ways to enumerate all sequences, both of which produce the same results through slightly different means. The method I present here does not require backtracking, but

suffers in that it has to enumerate over assembly sequences that have the same parallel sequence. Algorithm 7 works as follows: the starting triangles are found, and a set of all parallelized assemblies is initialized. Then, for each starting triangle, a queue (i.e. breadth-first search) is started with a single element: the partial parallelized order of the starting triangle. Then, as long as the queue has partial orders, a partial order is popped and flattened to a normal assembly order. The nodes in the assembly order are found. If the assembly order is finished, the parallel version is added to the set of orders, and the while loop continues. Otherwise, the set of next nodes possible nodes is found; if none is found, then no assembly sequence exists. For each of the next possible nodes, the normal assembly order is appended by the next node, the parallel version of the order is found, then added to the queue if the parallel version is not already there.

4.10 Sequence Count Estimation

To estimate the assembly sequence count, I implemented a method first described by Donald Knuth [31] to estimate the number of terminal nodes in a search tree: take random walks, and for each walk, count the number of branches available at each step, and at the terminal node, return the product of the number of branches at each step. The full estimation algorithm, **Estimate-SequenceCount** is shown in Algorithm 8. Each random walk is generated by **BranchProduct**, shown in Algorithm 9. Over many trials, this number converges to the number of sequences. While the simplest method to generate random sequences is to add one step at a time from the set of steps that are available, this can result in convergence to the same structure graph from different paths, making the search space a directed acyclic graph instead of a tree. A better algorithm proceeds by adding to the structure a random subset from the set of possible float base pairs. To make this a search tree, the set of nodes that can be added is limited to nodes that have at least one base in the set of nodes that were added in the previous iteration. That is, a possible float base pair cannot be added now if it could have been added at an earlier time. But this leads to possible dead ends: if the algorithm does not add a node when it had the opportunity, it may not be possible to add in future iterations. For this reason, the algorithm does not always return a sequence.

Since some of the searches will result in dead ends, the tally of dead ends is kept. The mean of the set of branch products, one for each trial, is then multiplied by the ratio of successful trials to total trials to produce an estimated sequence count. Since **BranchProduct** starts with a completed triangle, the estimated sequence count is per triangle: the overall total is found by multiplying by the number of starting triangles.

BranchProduct starts with a random triangle, a parallel sequence generated from the triangle, and initializes the branch product at 1. Then, until the random sequence is finished: it determines the set S_{next} of next float base pairs for the parallel sequence. It then prunes the set by including only float base pairs whose base contains at least one node that was added in the previous level. If no such nodes remain, a dead end was reached, and the function is terminated. Otherwise, the leftover set, S_{latest}, is further partitioned in the following way:

- Create a new set for each vertex: $S_{i,latest}$.
- If there is only one float node in the full set S_{latest}, it must be placed.
- Otherwise, if there are multiple float nodes in S_{latest}, check to see if adjacent nodes to i are not yet made, and if so, add $\emptyset$ to $S_{i,latest}$. This allows the node to be skipped over this step and attached in a later level. If all neighboring nodes have been made, the node must be added in this level. The purpose behind this rule is to reduce the number of dead ends, but does not fully eliminate them.

Once S_{latest} is modified, the number of branches is the product of the number of ways the nodes can be added to the current level. Since at least one node is needed, the empty set (i.e., adding nothing) is subtracted from the product, but only if every node can be added later as determined previously. Finally, a random selection from each $S_{i,latest}$ is chosen and appended to the partial order (if an empty set is chosen for some node i, it is skipped), the running product is multiplied by the branch number for this step, and the loop continues. When the partial order is completed, the branch product is returned.

Algorithm 8 This algorithm estimates the sequence count by doing a number of random assemblies and averaging the product of the number of branches at each assembly step, following an algorithm described in [31].

EstimateSequenceCount(V, E, N_{trials})

$\Pi_{trials} \leftarrow \emptyset$
$\Sigma_{failures} \leftarrow 0$
$n \leftarrow 1$
$\Delta \leftarrow$ **AllTriangles**(V, E)
while $n \leq N_{trials}$ **do**
  $\Delta_{random} \leftarrow$ random choice from Δ
  $\Pi_n \leftarrow$ **BranchProduct**(V, E, Δ_{random})
  if $\Pi_n =$ **Dead End then**
    $\Sigma_{failures} \leftarrow \Sigma_{failures} + 1$
  else
    $\Pi_{trials} \leftarrow \Pi_{trials} \cup \{\Pi_n\}$
    $n \leftarrow n + 1$
return $\frac{N_{trials}}{N_{trials}+\Sigma_{failures}}\text{Mean}(\Pi_{trials})$

Algorithm 9 This algorithm randomly assembles a structure by adding a subset of the possible next steps. It estimates the branch product by multiplying the number of subsets of possible additions at each step.

BranchProduct(V, E, Δ_{random})

$P_{partial} \leftarrow (\Delta_{random})$
$\Pi_{branches} = 1$
while $\sum_t \|P_{partial,t}\| < \|V\|$ **do**
  $S_{next} \leftarrow$ all float base pairs that can be attached to the structure
  $S_{latest} \leftarrow$ prune S_{next} by requiring the base set includes a node made in the previous step
  if $S_{latest} = \emptyset$ **then**
    return Dead End
  for $S_{i,latest} \in S_{latest}$ where $S_{i,latest}$ is the set of float base pairs with float node i **do**
    if $\|S_{latest}\| > 1$ and i has at least one neighboring node not yet made **then**
      $S_{i,latest} \leftarrow S_{i,latest} \cup \{\emptyset\}$
  $\Pi_{next} \leftarrow \prod_i \|S_{i,latest}\|$
  if $\forall i, \emptyset \in S_{i,latest}$ **then**
    $\Pi_{next} \leftarrow \Pi_{next} - 1$
  $\Pi_{branches} \leftarrow \Pi_{branches}\Pi_{next}$
  $L_{next} \leftarrow$ random selection from each $S_{i,latest}$.
  $P_{partial} \leftarrow P_{partial}\|(L_{next})$
return $\Pi_{branches}$

4.11 Numerical Estimation of Sequence Count for Various Truss Styles

I tested Algorithm 8 on a variety of truss structures resembling real-world examples, and verified the sequence count algorithm on small structures on which it was feasible to iterate all

sequences. I then compared these to the theoretical upper bound $O(K^N)$. The types of trusses are:

- Cubic trusses, subdivided into towers, planes, and full 3D structures.
- Tetrahedral-octahedral honeycombs, also subdvided into towers, planes, and full 3D structures.
- Telescope trusses, a slice of the tetrahedral-octahedral honeycomb containing two layers.

Before estimating the count for large truss structures, I tested the algorithm on a few tractable examples based on the cubic tower and telescope trusses. The single cube has 96, the double cube has 2448, and the telescope has 12708 assembly orders when all starting triangles are considered. Algorithm 8 was run 10000 times for each structure, and averaged to 95.15, 2459.47, and 12669 sequences, empirically validating the algorithm. A table showing the validation results is shown in Figure 4.1.

Figure 4.2 shows the K exponent based on each structure's asymptotic D_v compared to the observed exponent for each structure. The exponent is derived from a linear fit of the logarithmized estimated sequence counts. The following subsections describe the results in more detail, but one trend can be observed immediately: the number of sequences is exponential, but grows more slowly than the number of combinations estimated by K. This is expected. In a few cases, such as with the telescope and the full cube trusses, the exponent is significantly smaller. In all cases, the exponent difference shows that it is impractical to randomly sample a $3N - 6$ set of edges expecting to find a valid sequence: the likelihood of finding one becomes almost 0 for even modest structures.

4.11.1 Cubic Trusses

Cubic trusses are ubiquitous. They are most recognizable as Pratt trusses and Howe trusses, which can be found on a large number of bridges around the world. However, trusses such as these are not necessarily composed of tetrahedra, so additional struts must be added to the cubic cells to permit them to be built as a series of tetrahedra. This can be achieved by considering the extra struts as temporary struts.

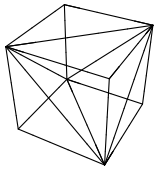
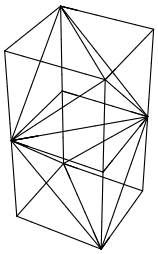
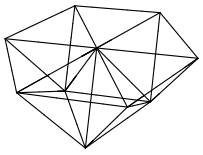
	N	E	Δ	# Sequences	# Sequences per Triangle	Est # Sequences	Est # Sequences per Triangle
	8	24	96	96	1	95.15	0.991146
	12	31	180	2448	13.6	2459.47	13.6637
	10	26	150	12 708	84.72	12 669.	84.4602

Figure 4.1: For three small structures, Algorithm 8 is compared to the true sequence count. Each column, from right to left, shows the structure, the number of nodes, the number of struts, the number of starting triangles, the actual number of sequences, the average actual number per triangle, the estimated number of sequences, and the estimated number per triangle.

A cube can be subdivided into five tetrahedra as shown in Figure 4.1, in which 4 non-adjacent corners are given diagonals. This results in a lattice in which vertices alternate between having diagonals and not. For these experiments, each vertex is at an integer $\{i, j, k\} \in \mathbb{Z}$. Each vertex has an edge to all vertices that are at 1 unit distance, and the vertices whose coordinate sum $i+j+k$ is even have an edge to all the vertices that are at $\sqrt{2}$ units distance. I tested three kinds of trusses: a cube tower of various heights, which may also be thought of as a Pratt bridge, a cubic plane with various horizontal dimensions, and full cubes, with 3D square lattices. Cube towers are lattices with 2 vertices along the X and Y axes, with the Z-axis varying from 2 to 10 vertices. Cube planes have 2 vertices on the Z axis, with between 2 and 10 vertices on the X and Y axes. Full cubes vary

Cube	D_v	K	Fit Exponent
Tower	3.25	2.41419	1.91
Plane	4.5	17.537	6.35
Full	6	64.	8.47
Tet Oct	D_v	K	Fit Exponent
Tower	3.88889	8.08856	4.87
Plane	5.	28.9352	10.74
Full	7.	119.147	11.5
Telescope	5.	28.9352	8.78

Figure 4.2: The half-vertex degree, D_v (where $\|E\| = D_v N$) for arbitrarily large structures of various styles is shown, along with the upper bound exponent K, and the experimentally derived exponent, which for all cases is smaller than K, showing that the number of assembly sequences is far smaller than the number of $3N - 6$ combinations.

from 2 to 10 vertices on all axes.

Figure 4.3 shows the results. The cube tower has a D_v value approaching 3.25 as height increases, the plane 4.5, and the full cube 6. For each structure of each size , the estimated sequence count is shown. 10 sets of 1000 trials each were performed (to show the variance in the mean calculation) as well as its fit, along with the number of combinations $\binom{E}{3N-6}$ in the structure, and the plot of $O(K^N)$. In all cases, the number of structures is $O(K^N)$ as predicted, and is in fact significantly better.

4.11.2 Tetrahedral-Octahedral Honeycombs, Including Telescopes

The tetrahedral-octahedral honeycomb is a space-filling tiling consisting of octahedra surrounded by tetrahedra. This kind of truss is also ubiquitous, even in reduced strut count. Space trusses, for example, consist of half-octahedra and the tetrahedra between them. Slicing the honeycomb on a different plane produces the set of space telescope trusses [66].

As with the cube trusses, the honeycomb must be divided into tetrahedra to allow the IPJRs

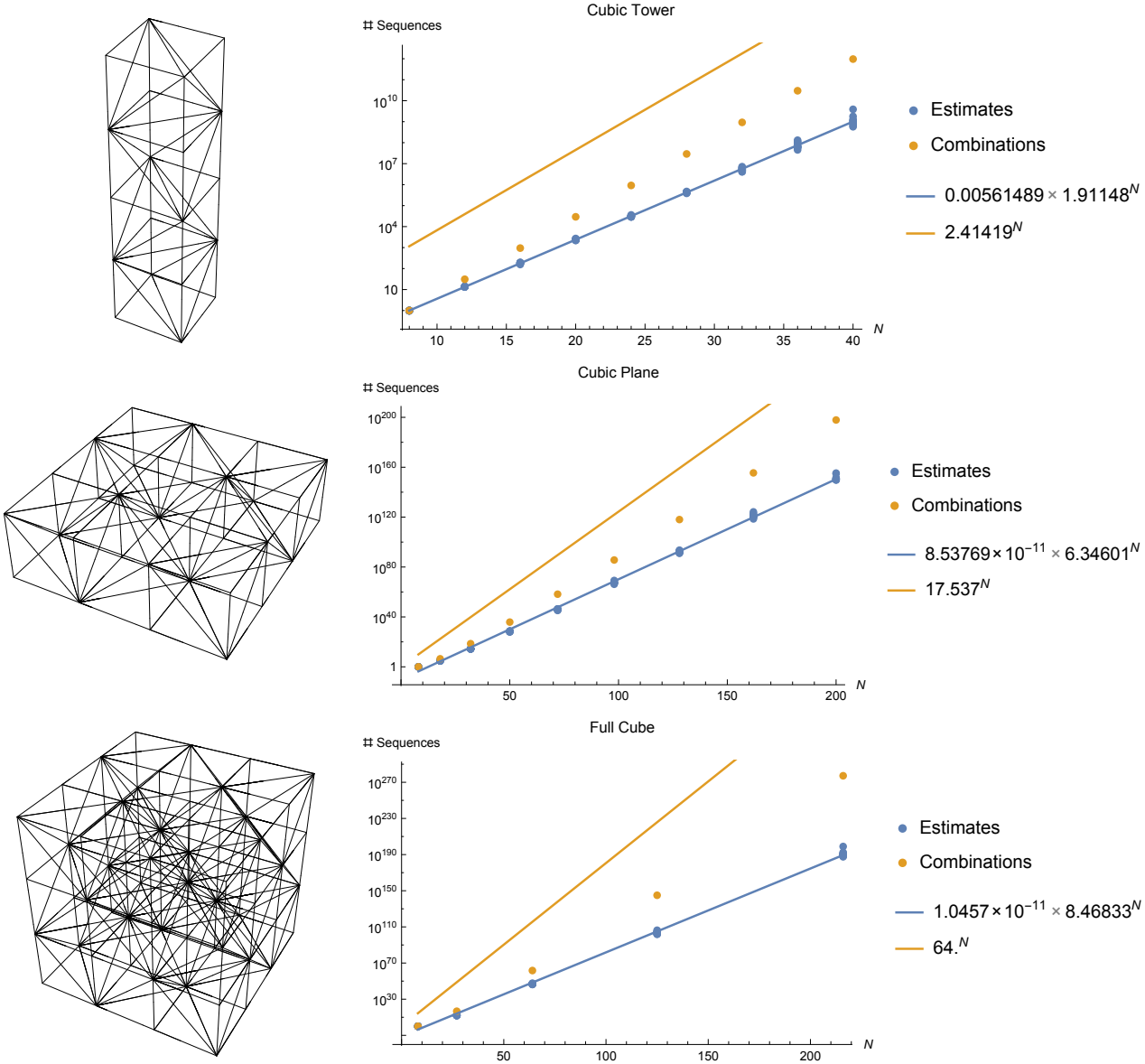


Figure 4.3: For cubic towers/bridges (top), planes (middle), and full cubes (bottom), the plots of estimated sequence count (blue), number of $3N-6$ combinations (yellow points), and K^N (yellow line) are shown with respect to node count N.

to construct using tetrahedral cells. These can be considered temporary to the final structure, but are needed to ensure a rigid placement for the floating nodes. An octahedron can be cut into four tetrahedra by inserting a single edge creating a diagonal on one of the squares, therefore making the honeycomb possible to build with IPJRs without adding a large number of extra struts. A

tetrahedral-octahedral honeycomb can be defined by the set of vertices $\{i, j, k\} \in \mathbb{Z}$ such that $i + j + k$ is even. Every vertex has edges to all other vertices that have $\sqrt{2}$ units distance. The octahedron-dividing diagonal adds edges to the truss in the following way: an edge exists between V_1, V_2 if $i_1 - i_2 = 2$ and $\|V_1 - V_2\| = 2$, which says that each vertex is connected to the nearest neighbors along the X-axis. As with the cube truss, I tested a tower, a plane based on the space truss, and a fully 3D truss. Additionally, I tested the telescope planes. The tower has a maximum dimension of 3 on the X and Y axes, and ranges from 2 to 7 on the Z-axis. The plane truss has 2 vertices on the Z-axis and ranches from 3 to 10 on the X and Y axes. The full truss ranges from 3 to 7 vertices along all axes.

The telescope is defined differently: each example produces a hexagon on the top surface where each side of the hexagon has between 2 and 7 vertices. If the vertex count per side is S_v, then the vertices are in the set $V_{telescope} = \{\{i, j, k\} : i, j, k \in [1, 3S_v - 2], (i + j + k = 3S_v - 2 \wedge Max(i, j, k) < 2S_v - 1) \vee (i + j + k = 3S_v \wedge Max(i, j, k) < 2S_v)\}$. The former condition produces the bottom layer, the latter the top.

The half-vertex degree D_v is $3.\bar{8}$ for the tower, 5 for the plane and telescope, and 7 for the full scope. As with the cubes, the number of assembly sequences is exponential, but less so than the number of combinations and the upper bound $O(K^N)$.

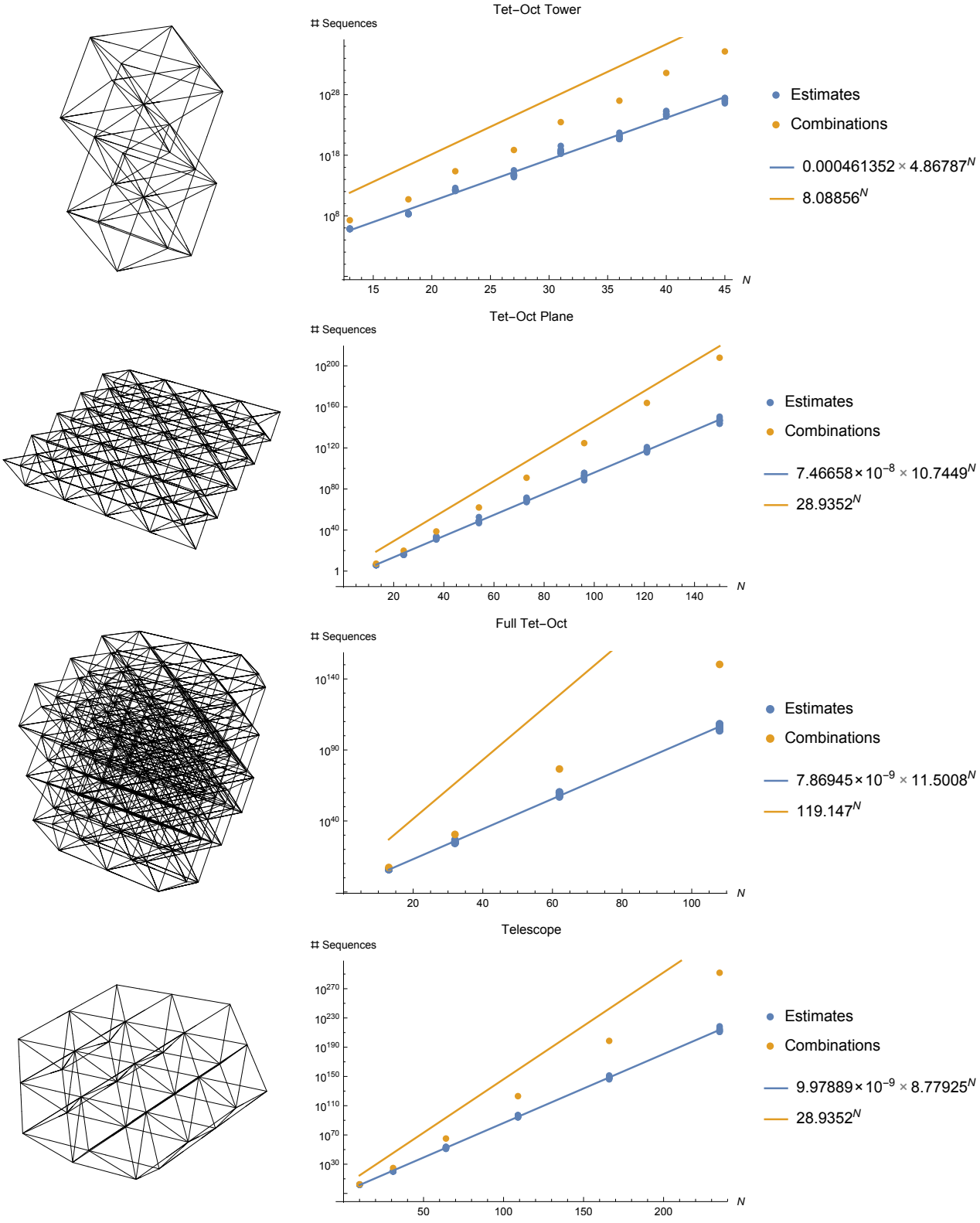


Figure 4.4: For tetrahedral-octahedral honeycomb towers/bridges (top), planes (top middle), full honeycombs (bottom middle) and telescopes (bottom), the plots of estimated sequence count (blue), number of $3N - 6$ combinations (yellow points), and K^N (yellow line) are shown with respect to N.

Chapter 5

Assembly Probability Model

The previous chapter described a method to construct triangular and tetrahedral cells in a truss structure by adjusting the lengths of the struts between the base nodes and the floating node. This chapter builds on the previous chapter by describing the probability model for assembly when strut length adjustment has uncertainty. First, I describe the complete, nonlinear probability model for an assembled structure. I follow this with a linearized model, appropriate for high precision and for determining an optimal assembly sequence. Linearization leads to convenient methods to determine both the overall and the node-wise marginal probabilities, enabling exact analysis of assembly sequences. Finally, I compare the nonlinear and linear models to estimate an upper bound on allowable uncertainty for the linear model.

The model presented in this chapter does not assume forces, internal or external. It is representative of an on-orbit structure without internal stresses, and not subject to thermodynamic expansion and other environmental factors.

5.1 Probability of a structure state X with respect to lengths L_E

Errors in the truss structure arise from errors in extending and contracting the IPJRs. These errors build up over the assembly; distant nodes with long chains of ancestors are far more likely to have larger covariances. The truss probability model is based on expected positional errors of the nodes, assuming that no on-board error detection and correction is used when assembling. This model is referred to as the **open loop** model. In this section, I describe the probability model and

the reasoning for the using the covariance trace as the error metric.

To derive an expression for the covariance, the following assumptions are made: IPJR errors are small relative to the structure, and the length of each strut is a Gaussian noise variable $L_{i,j} \sim \mathcal{N}(\bar{L}_{i,j}, \sigma_L^2)$, where $\bar{L}_{i,j}$ is the nominal length plus noise with **process variance** σ_L^2. Recall the definition of the node assembly function in Equation 3.2.

The recursive nature of Equation 3.2 is readily apparent; for each X_i, replace it with $\mathcal{F}_i$, and repeat down to the base cases: $X_1 = 0, X_2 \sim \mathcal{N}(\bar{L}_{1,2}, \sigma_L^2)$. Let the function returning the full structure state $\mathbf{X}$ as a function of the full set of struts in E_A be $\mathcal{F}_V$:

$$\mathbf{X} = \mathcal{F}_V(L_E) \tag{5.1}$$

Note that while the error model for each individual strut $\mathcal{N}(\bar{L}_{i,j}, \sigma_L^2)$ is independent, error accumulates through node assembly. The probability distribution of the full structure state $\mathbf{X}$ is therefore the product of the probability distributions of each of the lengths $p_{L_{i,j}}$, where each is the aforementioned distribution and the variable is the length between nodes:

$$p(\mathbf{X}) = \prod_{i,j \in E_A} p(\|X_i - X_j\| - \bar{L}_{i,j}) \tag{5.2}$$

Despite the apparent simplicity of the function definition, solving for the covariance and finding marginal probabilities requires calculating integrals over large dimensional spaces. Approaches for solving this problem have been the subject of ongoing research for centuries, and advances in computational hardware cannot fully solve this problem. It is of particular importance in the field of robotics, where fast methods for finding statistics such as mean and covariance of robot and environment states given noisy measurements are a research priority [61]. This dissertation explores a few such approaches. One is the linearization of the model around a fixed point, which I describe in the following sections.

5.2 Linearized probability of a structure state X with respect to lengths L_E

Linearizing nonlinear functions has been a popular and generally useful method for dealing with tough control and probability problems. Let J be the Jacobian matrix of the full structure state with respect to L_E with rows $L_{i,j}$, evaluated at the nominal lengths $\bar{L}_E$:

$$J = \left.\frac{d\mathcal{F}_V(L_E)}{dL_E}\right|_{\bar{L}_E} \tag{5.3}$$

Then the linearized version of $\mathcal{F}_V$, called $\bar{\mathcal{F}}_V$, is defined around the desired complete node state $\bar{\mathbf{X}}$:

$$\bar{\mathcal{F}}_V(L_E) = \bar{\mathbf{X}} + J(L_E - \bar{L}_E) \tag{5.4}$$

The probability model in Equation 5.2 can be approximated as a linear model around the nominal structure $\bar{\mathbf{X}}$ for very small noise, a valid assumption for space telescope trusses with standard deviations on the order of $10^{-6}m$. As described in [11], the approximate covariance of a nonlinear function of Gaussian noises can be found as follows. Because the length noises are independent of each other (i.e. zero-mean Gaussian noise added to a nominal length), the Σ_L term is a diagonal matrix with entries σ_L^2, allowing for simplification:

$$\begin{aligned} p(\mathbf{X}) &\approx \mathcal{N}(\bar{\mathbf{X}}, \Sigma_X) \\ \Sigma_X &= J\Sigma_L J^T = \sigma_L^2 J J^T \end{aligned} \tag{5.5}$$

In this form, determining the overall and marginal distributions of the assembly function is tractable, enabling an algorithm to quickly evaluate any assembly sequence.

5.2.1 Covariance Trace as the Metric

Given several assembly sequences A_a, the covariance matrices $\Sigma_{\mathbf{X_a}}$ can be calculated as shown in Equation 5.5. But how does one compare two covariance matrices to determine which

one is "better?" The metric I chose was the trace of the covariance matrix, which is defined as $Tr(\Sigma) = Tr($ eigenvalues $\lambda) =$ sum of the diagonals. I define the optimal assembly order as the one that minimizes the trace:

$$A_{optimal} = \underset{A_i}{\operatorname{argmin}}(\mathrm{Tr}(\Sigma_{X,A_i})) \tag{5.6}$$

The trace is the sum of the expected squared errors of each variable, $Tr(\Sigma_X) = \sum_i E[(X_i - \bar{X}_i)^2]$, making the trace the expected squared error of the entire structure vector $\mathbf{X}$.

The trace of a covariance matrix can also be expressed easily in geometrical terms. The exponent in a multivariate normal distribution, $(X-\mu)^T\Sigma^{-1}(X-\mu)$, is the square of the Mahalanobis distance [43]:

$$D_M(X) = \sqrt{(X-\mu)^T\Sigma^{-1}(X-\mu)} \tag{5.7}$$

The Mahalanobis distance is a multivariate analogue of the standard deviation: all points satisfying $D_M(X) <= 1$ are said to be within one standard deviation of the mean. The set of points at which the Mahalanobis distance is constant are hyperellipsoids for that constant. This is obvious when the diagonalized form of the covariance matrix is used:

$$D_M(X)^2 = 1 = \frac{(x_1-\mu_1)^2}{\lambda_1} + \cdots \tag{5.8}$$

The semi-major and semi-minor axes of the hyperellipsoid are the square roots of the eigenvalues, $\sqrt{\lambda_1}$. Due to the eigenvectors being orthonormal, the eigenvectors scaled to the square roots of the eigenvalues can be considered to be the half-lengths of a bounding box surrounding the ellipsoid for $D_M(X) = 1$. The half-length of the diagonal of the bounding box is the square root of the sum of the eigenvalues, which makes it the square root of the trace, and thus the expected squared distance of a point to the mean μ, as shown in Figure 5.1.

When expanded out using the definitions of $\mathbf{X}$ and L_E in Equations 3.5 and 3.6, Equation 5.5 becomes:

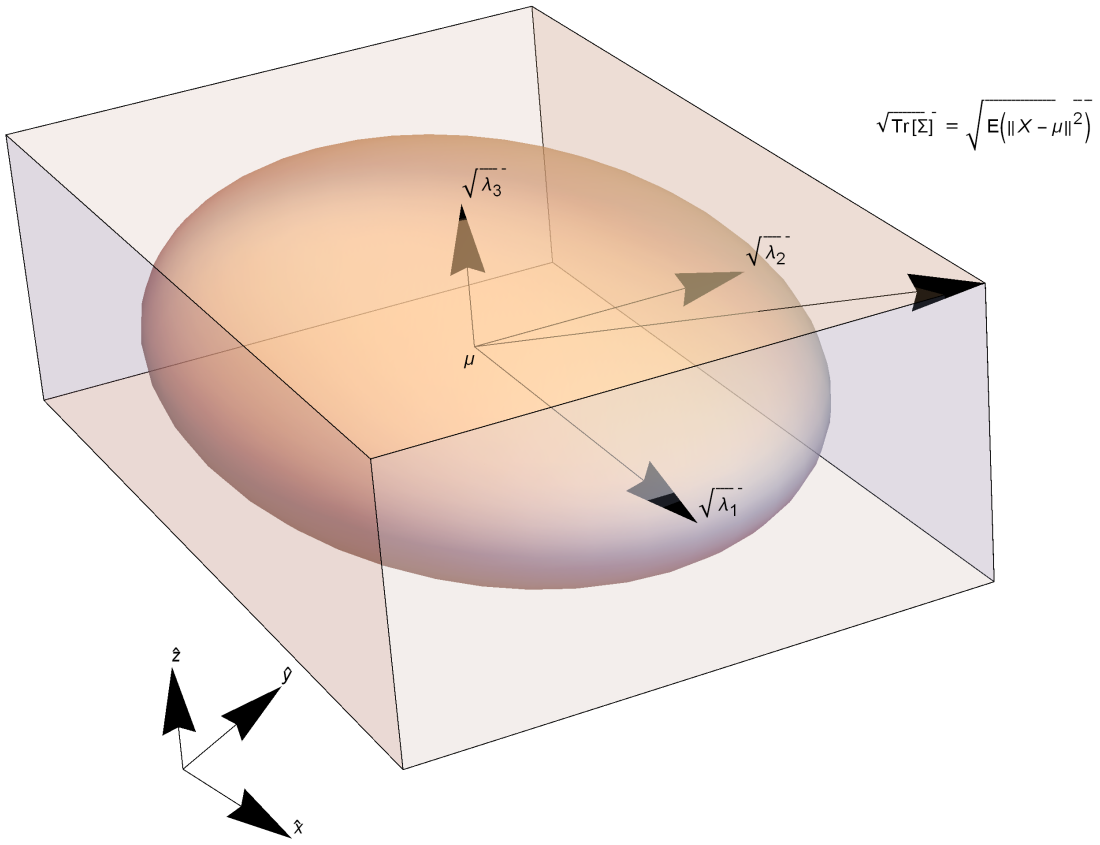


Figure 5.1: A covariance matrix Σ around mean μ is decomposed, where the eigenvalues are $\lambda_1, \lambda_2, \lambda_3$, and the eigenvectors are shown emanating from the mean μ. The Mahalanobis ellipsoid for $D_M(X) = 1$ is shown, along with the bounding box of the ellipsoid whose half-diagonal is equal to the square root of the trace of Σ and the square root of the expected squared distance $E[\{X - \mu\|]$.

$$
\begin{aligned}
\Sigma_X &= \begin{pmatrix} \frac{dx_2}{\mathrm{dL}_{2,1}} & \cdots & \frac{dx_2}{\mathrm{dL}_{last}} \\ \vdots & \ddots & \vdots \\ \frac{dz_N}{\mathrm{dL}_{2,1}} & \cdots & \frac{dz_N}{\mathrm{dL}_{last}} \end{pmatrix} \cdot \begin{pmatrix} \sigma_L^2 & 0 & \cdots \\ 0 & \sigma_L^2 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \cdot \begin{pmatrix} \frac{dx_2}{\mathrm{dL}_{2,1}} & \cdots & \frac{dz_N}{\mathrm{dL}_{2,1}} \\ \vdots & \ddots & \vdots \\ \frac{dx_2}{\mathrm{dL}_{last}} & \cdots & \frac{dz_N}{\mathrm{dL}_{last}} \end{pmatrix} \\
&= \begin{pmatrix} \sum_{i,j \in E} \sigma_L^2 \frac{dx_2}{dL_{i,j}}^2 & \cdots & \cdots \\ \cdots & \ddots & \cdots \\ \cdots & \cdots & \sum_{i,j \in E} \sigma_L^2 \frac{dz_N}{dL_{i,j}}^2 \end{pmatrix}
\end{aligned} \tag{5.9}
$$

The trace of Σ_X, and the expected squared error of $\mathbf{X}$, is simply the length variance σ_L^2

multiplied by the sum of the derivatives of each node variable with respect to each length variable — the sum of the squared elements of the Jacobian:

$$\mathrm{Tr}(\Sigma_X) = \sigma_L^2 \sum_{i=1}^{N} \sum_{j,k \in E_A} \frac{dX_i}{dL_{j,k}}^2 \tag{5.10}$$

Additionally, since the nodes are defined with respect to only the lengths, subsets of the rows in J can be used to find the marginal covariance for that subset of variables:

$$\Sigma_{subset} = J_{subset}(\sigma_L^2 I) J_{subset}^T \tag{5.11}$$

This leads to another important fact:

$$\mathrm{Tr}(\Sigma_X) = \sum_{i=1}^{N} \mathrm{Tr}(\Sigma_{X_i}) \tag{5.12}$$

Finally, a node does not change with respect to struts that are placed **after** the node:

$$\frac{dx_i}{dL_{j,k}} = 0 \text{ when either or both of } j,k \text{ are placed after } i \tag{5.13}$$

Equations 5.12 and 5.13 show that the accumulation of expected squared errors is additive with nodes. When choosing the next step in an assembly sequence, one could find the Jacobian of each of the possible nodes with respect to all of the lengths leading up to it. The marginals of the nodes already placed are not affected. This enables incremental algorithms instead of calculating the full trace after the fact. However, these rules apply only to the trace itself: the covariance, in general, is not incrementally additive, but it can be found by adding new rows to J and recalculating by Equation 5.3.

The trace of a subset of node variables does not describe the individual eigenvalues λ, unless that subset is just one row of J. It is more useful to know the marginal covariances of entire nodes, which produce 3D ellipsoids for all but the first three nodes, and give a clear idea of the variability of each node. The trace will give the bounding box half-diagonal, but to find the semi axis lengths and vectors themselves requires eigendecomposition of $\Sigma_{\mathbf{X}}$ into $\Omega_i \Lambda_i \Omega_i^T$, with the columns of Ω_i

being the vectors with lengths equal to the square roots of the diagonal eigenvalue matrix Λ_i. Numerous methods exist for eigendecomposition and will not be reproduced here [25].

5.2.2 Numerical Calculation of Jacobian matrix J

The closed form derivative of $\mathcal{F}_V(L_E)$ is impractical to calculate due to the number of terms and the embedded square root functions, so I chose to approximate it instead. The central difference formula for calculating a derivative of a function approximates it with a small step size h, and has $O(h^2)$ error [45]:

$$\frac{df(x)}{dx} = \frac{f(x+h) - f(x-h)}{2h} + O(h^2) \tag{5.14}$$

To find the derivatives of all node positions $\mathbf{X}$ with respect to a single length $L_{i,j}$, simulate two assemblies on a structure with all lengths unchanged except for $L_{i,j}$, to which h is either added or subtracted, then follow the formula.

$$\frac{df_V(L_E)}{dL_{i,j}} = \frac{f_V(L_E|_{L_{i,j}=\bar{L}_{i,j}+h}) - f_V(L_E|_{L_{i,j}=\bar{L}_{i,j}-h})}{2h} + O(h^2) \tag{5.15}$$

The value I used for h is $h = 5 \times 10^{-7}$, making the Jacobian approximation error $O(10^{-13}m)$.

5.2.3 Covariance Trace and Cell Geometry

The covariance trace of a node with respect to only the struts connecting it to its base varies with the orthogonality of the struts. The closer to orthogonal the set of float-base struts is, the smaller the covariance trace of the float with respect to the base. I conjecture that the aforementioned trace is minimized when the three struts are all orthogonal to one another (that is, the apex could be a corner of a cube).

Conjecture 1.

$$\textit{If } (X_f - X_i).(X_f - X_j) = 0, \forall i,j \in B_f, \textit{ then } \left(\frac{d\mathcal{F}_f(L_{B_f})}{dL_{B_f}}\right)^2 \textit{ is minimal.} \tag{5.16}$$

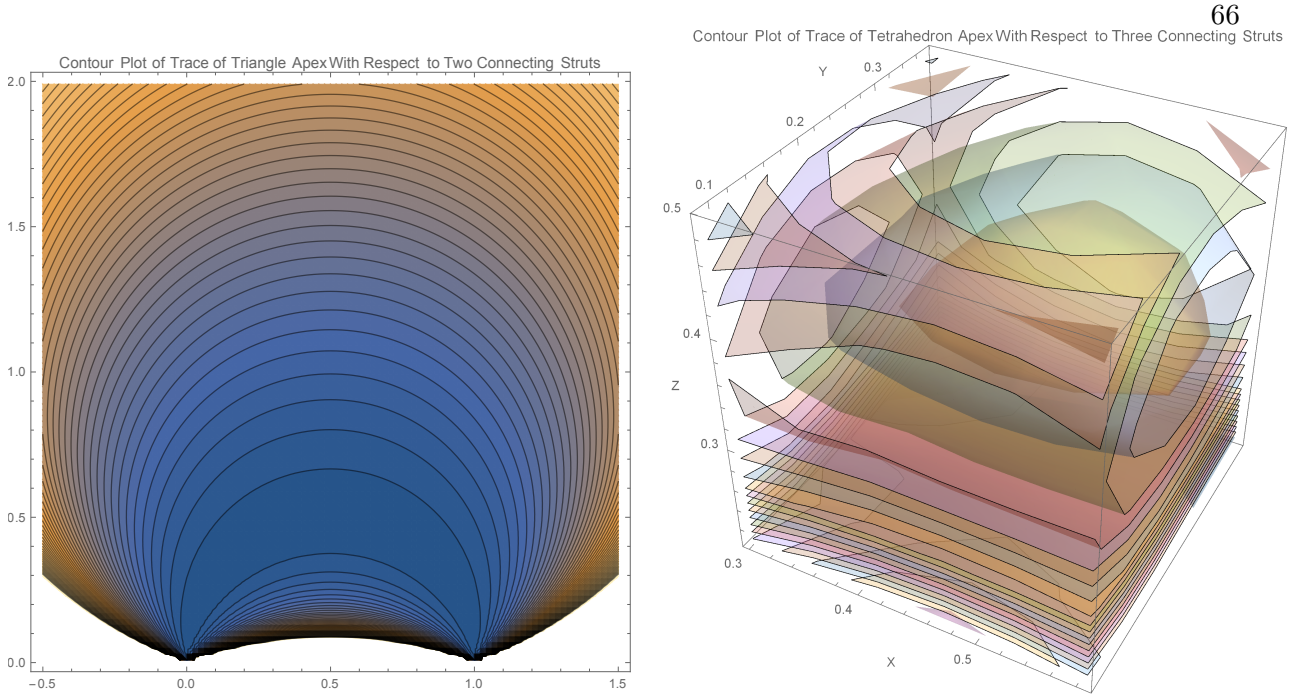


Figure 5.2: Contour plots of marginal covariance traces of a triangle apex with respect to its connecting struts (left) and a tetrahedron apex with respect to its connecting struts (right).

Figure 5.2 shows the marginal covariance traces of both the third node in the starting triangle, and the apex of a generic tetrahedron. For tetrahedral cells, there is a unique point at which the three struts are orthogonal to one another. The set of points with orthogonal struts is a curve for a triangle, as there are infinitely many ways a right triangle can be formed with a fixed hypotenuse.

Strut orthogonality is an important consideration for both assembly sequencing and structure design. For sequencing, the trace of float base pairs will increase the farther from orthogonal the struts are. In extreme cases of nearly degenerate tetrahedra, small deviations can lead to excessive errors, possibly violating the triangle inequality, which in the physical world would mean that one or more IPJRs would detach from the float node or the base. The assembly algorithms presented in this thesis avoid degenerate tetrahedra. However, when randomly choosing a sequence, such tetrahedra blow up the trace, resulting in physically meaningless traces.

By finding the gradient of the trace with respect to the node positions, a gradient descent algorithm (or something more advanced) can be used to modify the positions of the nodes to

minimize the overall trace:

$$\frac{d}{dX_f}\left(\frac{d\mathcal{F}_f(L_{B_f})}{dL_{B_f}}\right)^2 \tag{5.17}$$

I do not fully analyze the consequences of allowing some nodes to vary in this fashion (and is instead future work), however, my experimentation with such a gradient descent algorithm shows that care must be taken when deciding which nodes to allow to move and which must remain fixed: the gradient descent will favor orthogonalizing as many struts as it can, and this may result in nodes merging, forming degenerate cells.

5.2.4 Analysis of Triple Helix Truss

The example structure I use in this section is a tower made of regular tetrahedra in which each new node is attached to the base made of the three previously added nodes. It is the simplest 3D structure I could come up with, uses only one length, and has $3N - 6$ struts, making it non-redundant. I call this structure, shown in Figure 5.3, the "Triple Helix":

$$\begin{aligned} A_{th} &= [\langle 1, () \rangle, \langle 2, (1) \rangle, \langle 3, (1,2) \rangle, \cdots \langle i, (i-3, i-2, i-1) \rangle] \\ L_{i,j} &= 1 \end{aligned} \tag{5.18}$$

In this section, I examine the growth of the covariance trace as a function of assembly step: how much does node placement vary as it gets farther from the origin? Intuitively, distant nodes in the triple helix should should have a higher variance with respect to a strut the closer the strut is to the origin. Figure 5.4 shows a triple helix example with 10 nodes, including two views of the structure and the covariance ellipsoids (shown with $\sigma_L^2 = 0.0025$), and the table showing the values of J with the fixed variables on the starting triangle omitted. The individual J values for later nodes do not appear to differ substantially from those of earlier nodes, which may seem to suggest that the size of a node's ellipsoid is due only to the large number of non-zero entries in

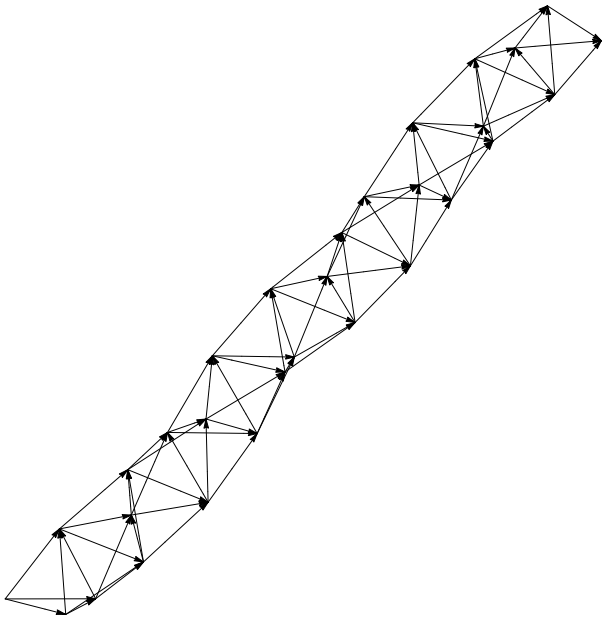
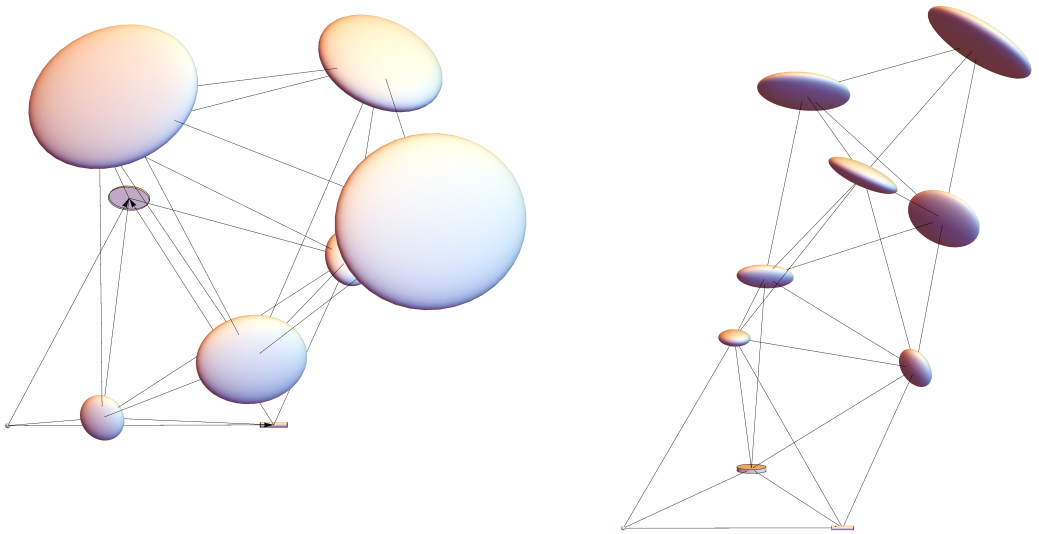


Figure 5.3: A simple structure, the Triple Helix, used for examples in this section. This example is shown with 30 nodes and 84 struts. It is composed of regular tetrahedra with edge length 1, each one built on top of the most recently added nodes, and is non-redundant.

J (that is, the node's ancestor struts). However, this is incorrect: in the long run, the average J value per node does increase, but very slowly: the two plots in Figure 5.4 show how the covariance trace per node increases per node number, and how the mean squared J value increases as well (excluding lengths that the node does not depend on). When scaling the structure up to 200 nodes, a polynomial growth rate in the covariance trace is observed. Nodal covariance trace scales with the cube of the node number. When looking at individual elements in the J matrix, the squared values (which sum to the trace), scale with the square of the node number.

I then tried a modified version of the triple helix: instead of each length being exactly 1, I allowed each length to be randomly chosen from $[0.8, 1.2]$ to see if altered geometry altered the covariance growth trends. The results are shown in Figure 5.5: while the results no longer follow a smooth curve, they do not deviate far from the fit functions, showing that cell geometry is not as



	$L_{2,1}$	$L_{3,1}$	$L_{3,2}$	$L_{4,1}$	$L_{4,2}$	$L_{4,3}$	$L_{5,2}$	$L_{5,3}$	$L_{5,4}$	$L_{6,3}$	$L_{6,4}$	$L_{6,5}$	$L_{7,4}$	$L_{7,5}$	$L_{7,6}$	$L_{8,5}$	$L_{8,6}$	$L_{8,7}$	$L_{9,6}$	$L_{9,7}$	$L_{9,8}$	$L_{10,7}$	$L_{10,8}$	$L_{10,9}$
x_2	1.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
x_3	0.5	1.	−1.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
y_3	−0.289	0.577	0.577	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
x_4	0.5	0.	0.	1.	−1.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
y_4	−0.481	0.385	0.385	0.577	0.577	−1.15	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
z_4	−0.136	−0.136	−0.136	0.408	0.408	0.408	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
x_5	0.333	0.667	−0.444	0.667	−0.444	−0.444	−0.333	0.667	0.667	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
y_5	0.0642	−0.513	0.77	0.385	−0.257	−0.257	0.962	−0.77	0.385	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
z_5	0.318	0.318	−0.816	−0.953	0.635	0.635	0.68	0.68	−0.953	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
x_6	−0.111	1.11	−0.296	1.11	−0.296	−1.04	−1.22	0.815	0.815	0.444	0.444	−1.22	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
y_6	−0.47	0.299	0.385	0.642	0.599	−1.11	0.0642	−0.0428	−0.0428	−0.128	1.03	0.0642	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
z_6	0.121	0.121	0.0454	−0.363	−0.257	0.575	0.0454	−0.0302	−0.0302	1.13	−0.499	0.0454	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
x_7	−0.0185	0.185	0.506	1.85	−1.16	−0.543	−1.04	−0.235	1.31	0.63	−0.42	−0.42	0.741	−0.926	0.63	0.	0.	0.	0.	0.	0.	0.	0.	0.
y_7	−0.303	−0.463	0.449	1.07	0.613	−0.656	0.684	−0.478	−0.442	−1.05	0.699	0.699	0.941	−0.0214	−1.05	0.	0.	0.	0.	0.	0.	0.	0.	0.
z_7	0.202	0.202	−0.605	−0.605	0.524	0.58	0.484	0.479	−0.857	0.0756	−0.0504	−0.0504	0.257	0.801	0.0756	0.	0.	0.	0.	0.	0.	0.	0.	0.
x_8	−0.364	1.31	−0.156	1.42	−0.267	−1.35	−1.73	0.831	0.934	0.716	0.646	−1.23	0.0494	−0.0329	−0.0329	−0.432	1.12	0.0494	0.	0.	0.	0.	0.	0.
y_8	0.00832	−0.836	0.684	0.82	0.0594	−0.195	1.14	−0.86	−0.0879	−0.784	0.102	0.803	0.756	−0.504	−0.504	0.606	−0.421	0.756	0.	0.	0.	0.	0.	0.
z_8	0.563	0.563	−0.781	−1.69	0.193	0.785	0.806	0.753	−0.425	0.806	−0.291	−0.702	−0.963	0.642	0.642	0.973	0.247	−0.963	0.	0.	0.	0.	0.	0.
x_9	−0.663	1.07	0.48	2.26	−0.705	−1.51	−2.33	0.274	1.37	1.19	0.447	−0.789	0.527	−0.59	−0.192	−1.03	0.686	0.686	0.835	−0.239	−1.03	0.	0.	0.
y_9	−0.574	−0.154	0.242	1.3	1.1	−1.05	0.297	−0.15	−0.767	−1.31	1.22	0.623	1.13	0.405	−1.53	−0.537	0.358	0.358	−0.0166	1.16	−0.537	0.	0.	0.
z_9	0.273	0.273	−0.0773	−0.818	−0.328	0.657	0.21	0.12	0.0778	1.34	−0.56	−0.225	−0.47	−0.00056	0.523	0.391	−0.261	−0.261	0.895	−0.314	0.391	0.	0.	0.
x_{10}	−0.586	0.598	0.85	2.57	−1.13	−1.23	−2.17	−0.234	1.59	1.25	0.00457	−0.401	0.878	−1.03	0.0311	−0.974	0.177	0.964	0.989	−0.659	−0.659	0.318	−0.472	0.989
y_{10}	−0.109	−1.27	0.532	1.49	0.586	−0.156	1.35	−0.949	−0.822	−1.96	0.319	1.35	1.88	−0.0797	−0.893	0.0459	−0.579	0.335	−0.617	0.411	0.411	1.15	−0.548	−0.617
z_{10}	0.57	0.57	−1.02	−1.71	0.517	0.773	0.955	0.931	−0.772	0.35	−0.102	−0.697	−0.784	0.962	0.513	0.909	0.382	−1.27	−0.376	0.251	0.251	0.283	0.988	−0.376

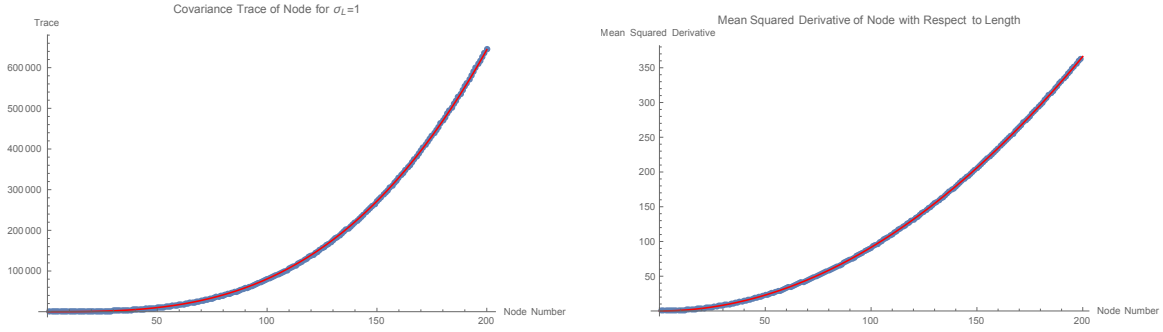


Figure 5.4: Upper left and right: two views of the covariance ellipsoids for the triple helix structure with 10 nodes, scaled to $\sigma_L^2 = 0.0025$. Center: the Jacobian matrix J for each node with respect to each length for the 10 node structure. Lower left: the covariance trace per node for a triple helix with 200 nodes, with $\sigma_L = 1$, shown with a fit function $0.0805N^3$, which shows that the covariance trace increases with the cube of the node number. Lower right: the mean squared derivative of each node with respect to each length (that is, the trace of the node divided by the number of nonzero elements of J_N) shown with a fit function $0.00915N^2$, which shows that the mean squared derivative of node with respect to length varies with the square of the node number.

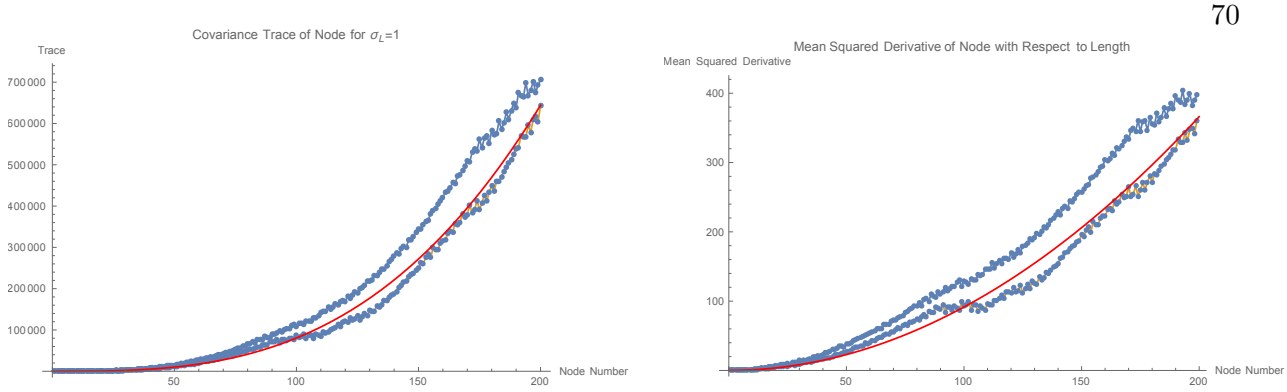


Figure 5.5: When allowing the lengths in the triple helix to vary randomly from 0.8 to 1.2, the covariance trace and mean squared derivative no longer smoothly follow the fit functions from Figure 5.4, but still grow as functions of N^3 and N^2 respectively.

important as distance and ancestor count for determining a covariance trace.

Two lessons are learned form this example: to minimize covariance trace in a structure, in general, one must minimize the distance from the origin to the nodes, and minimize the number of ancestors to any given node. A structure in which the origin is close to the center, and the assembly sequence grows out from the center, should be close to optimal. The next chapter explores this idea more fully.

5.2.5 Numerical Determination of Divergence of Linear $\bar{\mathcal{F}}$ From Nonlinear $\mathcal{F}$

The linearized assembly Equation 5.4 is only a suitable replacement for the actual, nonlinear assembly Equation 3.2 if the results produced by the former equation are reasonably close. "Reasonably close" here means an upper limit on the Euclidean distance between the structure vector or a single node as calculated by both functions. In this section, I will analyze the variations between the triple helix as assembled randomly by both the nonlinear and linear assembly functions for various σ_L and determine when the linear function is no longer useful.

For each length in the assembly order, I randomly choose from the distribution:

$$L_E \sim \mathcal{N}(\bar{L}_E, \sigma_L^2 I) \tag{5.19}$$

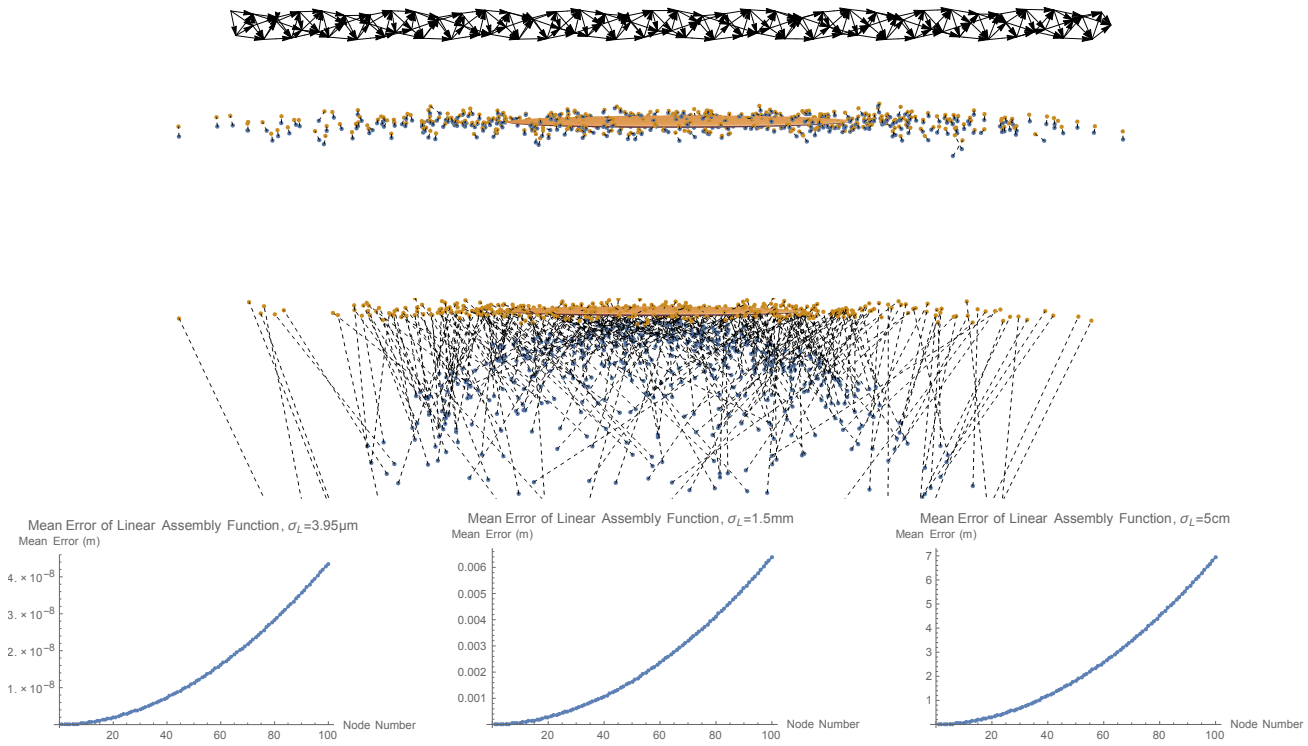


Figure 5.6: Top: The 100 node triple helix. Top Middle: Sample of 500 nodes calculated both by linear $\bar{f}$ (orange) and nonlinear f (blue), with dotted lines spanning the error gap, for the $\sigma_L = 1.5mm$ case. Bottom Middle: Same as above, but with $\sigma_L = 5cm$, showing the increase in variation as the samples move away from the mean. Bottom Left: mean error of $\bar{f}$ as a function of node number for $\sigma_L = 3.95\mu m$. Bottom Center: same, but with $\sigma_L = 1.5mm$. Bottom Right: same, but with $\sigma_L = 5cm$.

Then I calculated the true $\mathcal{F}_V$ and the linearized approximation $\bar{\mathcal{F}}_V$ and compared the results. I do this for 1000 trials of the 100-node triple helix structure, where each strut has a nominal length $\bar{L}_{i,j} = 1m$, and $\sigma_L = 3.95\mu m, 1.5mm, 5cm$. The first σ_L value, $3.95\mu m$, represents a potential linear actuator variance for an on-orbit experiment, $1.5mm$ is on the order of the 3D IPJR prototype, and $5cm$ is an arbitrary large standard deviation for comparison. The results are shown in Figure 5.6. For a structure as large as the 100-node triple helix, the smallest σ_L value results in nearly identical assembly function outputs, with the largest mean error on the order of $50nm$, which is small enough that the nonlinear function can be safely ignored. For the medium range σ_L, the mean error for the final node is about $7mm$, which, for a structure with 100 nodes attached in a

linear fashion, is roughly on the order of σ_L. For the $5cm$ example, the error is significant. For all nodes in all cases, the linear error can be viewed as a tangent to a ball, where the ball is the set of actual node positions given the random lengths — the farther away from the mean, the farther the tangent is from the ball.

For the micron level of precision required on the struts for space telescopes, the linear function $\bar{\mathcal{F}}$ is accurate to the order of nanometers, making it possible to build with precision without ever using or knowing the nonlinear version.

5.3 Measurements

Up to now, this chapter has dealt exclusively with strut length probabilities, with no consideration of measurements. While the mean of a measurement-free assembly distribution will always be the nominal position, with measurements this will change, and covariances will shrink. The first subsection will discuss the Gaussian canonical form, which simplifies operations such as marginalization, reduction, the combination (product) of covariances, and linear conditionals. The following subsections will consider measurements on both edge lengths and node positions, and their linearizations.

5.3.1 Multivariate Normal Canonical Form

Before describing how measurements are applied and structure estimates are made, some background material on how to combine measurements is needed. Multivariate normal distributions have a "canonical form" which makes it easy to find products of distributions, marginal distributions, reductions (such as measurement values), and linear conditionals. Linearized filters such as the Kalman filter are derived from these concepts. I present the canonical form as described in [32].

Let μ, Σ be the mean and covariance of some distribution $\mathcal{N}(\mu, \Sigma)$, and n is the length of μ. The canonical form is a triple k, h, g, which are defined:

$$\begin{aligned} k &= \Sigma^{-1} \\ h &= k\mu \\ g &= -\frac{1}{2}\mu^T k\mu - \log(2\pi)^{n/2}\sqrt{\mathrm{Det}(\Sigma)} \end{aligned} \tag{5.20}$$

The inverse operation is:

$$\begin{aligned} \Sigma &= k^{-1} \\ \mu &= \Sigma h \end{aligned} \tag{5.21}$$

Since g is not required to find Σ, μ, and does not factor into any calculations of k, h for the remainder of this section, I will leave it out for the sake of brevity.

To find a product of two distributions over two sets of variables $p(X_1)p(X_2)$ when $X_1 \sim \mathcal{N}(\mu_1, \Sigma_1), X_2 \sim \mathcal{N}(\mu_2, \Sigma_2)$, convert both to canonical form, then sum them:

$$k_{1,2}, h_{1,2} = k_1 + k_2, h_1 + h_2 \tag{5.22}$$

Since the variable sets X_1, X_2 may have an arbitrary degree of overlap, the matrices k_1, k_2 and h_1, h_2 must be expanded with zeros, and the combined variable set must be consistent between both matrices for the operations to be correct:

$$\text{If the combined variables are } (X_1, X_2), \text{then } k_1 = \begin{pmatrix} k_1 & 0 \\ 0 & 0 \end{pmatrix}, h = \begin{pmatrix} h_1 \\ 0 \end{pmatrix} \tag{5.23}$$

To find the marginalization $p(X)$ when $p(X, Y)$ is known — that is, the integral $\int_{-\infty}^{\infty} p(X, Y)dy$, first, rearrange the columns of k and h to align with the variable set (X, Y):

$$\text{For variables } (X, Y), k = \begin{pmatrix} k_{\mathrm{XX}} & k_{\mathrm{XY}} \\ k_{\mathrm{YX}} & k_{\mathrm{YY}} \end{pmatrix}, h = \begin{pmatrix} h_X \\ h_Y \end{pmatrix} \tag{5.24}$$

Then to find the marginalized k_m, h_m, g_m:

$$\begin{aligned} k_m &= k_{XX} - k_{XY} k_{YY}^{-1} k_{YX} \\ h_m &= h_X - k_{XY} k_{YY}^{-1} h_Y \end{aligned} \tag{5.25}$$

To find the reduction $p(X, Y = y)$ where y is a measurement:

$$\begin{aligned} k_r &= k_{XX} \\ h_r &= h_X - k_{XY} y \end{aligned} \tag{5.26}$$

Finally, the linear conditional form is defined, $p(X|Y) = \mathcal{N}(\mu_x + J(Y - \mu_Y)), \Sigma_X)$, in which the mean of Y is a linear system of X, and variable order (X, Y):

$$\begin{aligned} k_{lc} &= \begin{pmatrix} \Sigma_x^{-1} & -\Sigma_x^{-1} J \\ -J^T \Sigma_x^{-1} & J^T \Sigma_x^{-1} J \end{pmatrix} \\ h_{lc} &= \begin{pmatrix} \Sigma_x^{-1}(\mu_X - J\mu_Y) \\ -J^T \Sigma_x^{-1}(\mu_X - J\mu_Y) \end{pmatrix} \end{aligned} \tag{5.27}$$

5.3.2 General Method for Linearizing Measurements and Calculating Estimates

I recall the definition of the structure covariance matrix in Equation 5.5:

$$\Sigma_X = J \Sigma_{L_E} J^T = \sigma_L^2 J J^T \tag{5.28}$$

Let some generic measurement function $M = f(\mathbf{X})$ be linearized to the following form: $\bar{M} \sim \mathcal{N}(\mu_M + J_M(\mathbf{X} - \bar{\mathbf{X}}))$, and let m be a known measurement of $\bar{M}$. To get the optimal estimate $\hat{\mathbf{X}}$ of $\mathbf{X}$:

- Calculate k_X, h_X from $\mathbf{X}, \Sigma_X$.

- Find the linear conditional canonical form $k_{M|X}, h_{M|X}$ from Equation 5.27.
- Find the product $p(X, M) = p(X)p(M|X)$ as $k_{M,X}, h_{M,X}$ from Equation 5.22.
- Marginalize $k_{X,M}, h_{X,M}$ over $M = m$ using Equation 5.25.
- Convert back to μ, Σ form to get the estimated mean and covariance $\hat{\mathbf{X}}, \hat{\Sigma}_X$.

5.3.3 Position Measurements

The simplest measurement is that of a single node position, X_i. Global positioning systems like the Vicon [1] will associate a positioning error with the node measurement. Assuming the measurement of each axis is independent, one possible measurement is:

$$M_{X_i} = \mathcal{N}(X_i, \sigma_P^2 I) \tag{5.29}$$

Where σ_P is the per-axis standard deviation. The Jacobian can be calculated trivially as $J = I$, since the measurement mean changes exactly as the position does.

Position measurements may also be nonlinear functions of the position, and may depend on the position of the sensor, which itself may have a probability distribution. In this case, it is a SLAM problem, and the full state X must be augmented to include the position of the sensor, and the measurement function $M = f(X)$ will linearize to include both the nodes and the sensor's position. The most obvious such sensor is a camera; while cameras will no doubt play a very large role in future iterations of the IPJR, they are considered future work in this dissertation.

5.3.4 Length Measurements

Of chief interest in this dissertation are length measurements, which are local measurements only, and are taken by the IPJRs that are spanning the gap between two nodes. While each measurement only provides a scalar containing information on six variables, a there are at least as many measurements as there are unknown variables, a structure estimate can be made.

The length measurement function is $M_{L_{i,j}}(X_i, X_j) = \|X_i - X_j\| + \mathcal{N}(0, \sigma_M^2)$. To simplify the representation, I present the derivatives as a function of v_i, v_j as in Chapter A.2.4, where v stands in for any of the set x, y, z. The derivative with respect to the six node variables is:

$$\begin{aligned} \frac{d}{dv_i}(M_{L_{i,j}}) &= \frac{v_i - v_j}{\|X_i - X_j\|} \\ \frac{d}{dv_j}(M_{L_{i,j}}) &= \frac{v_j - v_i}{\|X_i - X_j\|} \end{aligned} \tag{5.30}$$

Using the notation in Equation 5.27, the J matrix is $\frac{d}{dX_{i,j}}(M_{L_{i,j}})$ evaluated at $\bar{\mathbf{X}}$, and the measurement mean $\mu_X = \mu_M$ is $\bar{X}_i - \bar{X}_j$.

5.3.5 Reduction of Structure Covariance Based on Which Strut is Measured

This section examines the effects of taking strut-length measurements on a structure, and discusses how one can maximize the utility of individual length measurements.

Let $p(M|\mathbf{X}) = \mathcal{N}(\mu_M + J_M(\mathbf{X} - \bar{\mathbf{X}}), \Sigma_M)$ be the linear conditional probability function for a set of measurements measurement M, and let $\bar{m}$ be an instance of such a measurement. Following the steps laid out at the end of Section 5.3.1, after measurements, the mean and covariance of the structure $\mathbf{X}$ after measurements are taken are:

$$\begin{aligned} \Sigma_{\mathbf{X}|M} &= (\Sigma_{\mathbf{X}}^{-1} + J_M^T \Sigma_M^{-1} J_M)^{-1} \\ \mu_{\mathbf{X}|M} &= \Sigma_{\mathbf{X}|M}(\Sigma_{\mathbf{X}}^{-1}\bar{\mathbf{X}} - J_M^T \Sigma_M^{-1}(\mu_M - J_M\bar{\mathbf{X}}) \\ &\quad -(-J_M^T \Sigma_M^{-1})\bar{m}) \end{aligned} \tag{5.31}$$

Again, I use the triple helix as the motivating example. I look at the effects of taking just one strut length measurement, varying which strut is measured. In particular, I wanted to know the quantity of improvement in the covariance trace as a function of which strut was measured. Is it better to measure in one area versus another, and why? I also considered measuring lengths between nodes to which a strut is not attached.

Figure 5.7 shows the results of individual strut measurements (top) and all node pair measurements (bottom). Generally, measurements on the earlier-placed struts have a better overall effect, due to the larger number of nodes that depend on such struts, which leads to a cumulative effect. The best strut to measure is the $(1, 4)$. As it turns out, the six best struts to measure follow the $(i, i + 3)$ format.

However, if I allow length measurements between nodes that are not spanned by struts (which, at $O(N^2)$, is the majority of them), a sizable portion of them are better than $(1, 4)$, the best being nearly 3 times better. I believe there are three reasons for this: there is a larger set of measurements to choose from, measurements between non-strut node pairs are not already implicitly included in the covariance matrix, and many of these span large gaps that are otherwise spanned by multiple, propagated strut errors. It is therefore very beneficial to measure lengths between any pairs of nodes possible, not just the struts, to enhance the overall result.

The methods in the following chapters do make use of some measurements that are not of assembly struts. Recall that, for redundant structures, in many cases the base triple is not the complete set of available struts to the new node. The remainder form the passive struts. IPJRs passively adjust their lengths to accommodate the assembly IPJRs, not contributing to the process noise. They do measure the post-weld lengths, however, giving the estimate additional loop closures.

The reduction in covariance trace is an example of loop closure in SLAM. Measurements on nodes with numerous ancestors have a greater impact on the accuracy of the estimate, and measurements on unique node pairs not found in the assembly process are better overall. However, the measurable node pairs will be limited by which ones have struts and/or which ones are close enough to be spanned by an IPJR, depending on the hardware implementation

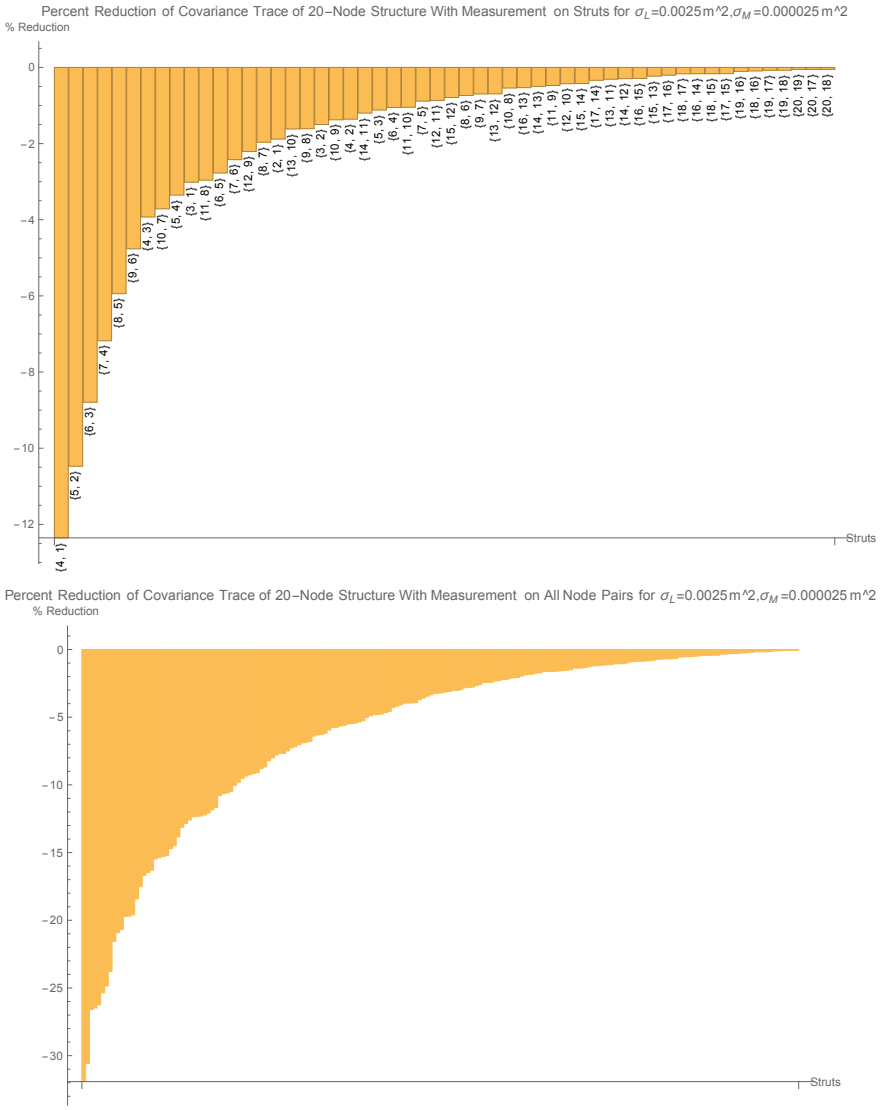


Figure 5.7: Top: the reduction in covariance trace over the 20-node triple helix given a single measurement on the strut listed (not cumulative), showing that, in general, earlier-placed struts have a better impact on the covariance when measured. Bottom: the reduction in covariance trace when all node pairs are included, which amounts to 180 measurements, too many to label. Even without labels, it is clear that many non-strut measurements result in a much larger reduction in covariance trace than the best of the strut measurements.

Chapter 6

Optimizing Assembly Sequences

With the covariance trace established as the metric, the next step is describe an algorithm that can find good assembly sequences. The set of assembly sequences is **exponential** in size, and the problem is NP-Hard, so the optimal sequence cannot be found except through brute force search on very small structures. However, a few heuristics may be employed to find very good sequences:

- A centralized starting point, building incrementally outward, will lead to better results, because a node is likely to be more precise if the chain of steps leading to it is smaller. The algorithm **AllCentralTriangles** was introduced in Section 4.7.
- Choosing the assembly step that minimizes the trace leads to a near-optimal result.
- Better sequences can be found by modifying one step at a time, eventually finding a minimum.

Implementing all of the above heuristics results in an algorithm that can reliably produce assembly sequences with smaller traces than any heuristic alone. This algorithm is **FindLocally-OptimalSequence**, which first picks a central starting triangle, generates an initial sequence by greedy choice, then swaps steps until a minimum trace is found.

Algorithm 10 This algorithm constructs an assembly sequence given a starting triangle by choosing the step among all available steps that minimizes the addition to the structure's overall trace.

GreedyAssembly(V, E, Δ, σ_L)

$A_{made} \leftarrow \Delta$
while $\|A_{made}\| \neq V$ **do**
  $V_{made} \leftarrow \{f : (f, B) \in A_{made}\}$
  $S_{next} \leftarrow$ **PossibleNextNodes**$(V_{made}, V \setminus V_{made}, E)$
  $A_{best} \leftarrow \emptyset, T_{best} \leftarrow \infty$
  for all $(f, B) \in S_{next}$ **do**
    $A_{candidate} \leftarrow A_{made} \cup (f, B)$
    $T_{candidate} \leftarrow \sigma_L^2 \left.\frac{df_V(L_{E,candidate})}{dL_{E,candidate}}\right|^2_{\bar{L}_{E,candidate}}$
    if $T_{candidate} < T_{best}$ **then**
      $A_{best} \leftarrow A_{candidate}, T_{best} \leftarrow T_{candidate}$
  $A_{made} \leftarrow A_{candidate}$
return A_{made}

6.1 Assembly by Greedily Minimizing Trace

The greedy choice algorithm, shown in Algorithm 10, assembles a structure incrementally by choosing the assembly step that minimizes the overall trace. While this will not always produce the best option, it does not require backtracking, and returns good sequences. In addition, it can be used as a starting point for a local search to find a better sequence. It is conceptually simple: beginning from the starting triangle, look at each possible next node, and choose the float-base pair that minimizes structure covariance. Calling **GreedyAssembly** for a central starting triangle has a better chance of finding a lower covariance than starting from other triangles, as I show in Section 6.4.

6.2 Descent by Local Search Around Complete Sequences

Instead of starting from a starting triangle (or nothing at all) and incrementally assembling a structure, one can find assembly sequences by tweaking existing sequences. **AssemblyLocalSearch**, Algorithm 11, examines every single neighboring sequence of a given sequence and moves to the sequence whose trace is the smallest. The process is repeated for the new sequence until no neighboring sequence has a better trace. This is a local minimum in the space of assembly

Algorithm 11 Local search is analogous to steepest descent in the graph of assembly sequences, in which the traces of all neighbors to the current sequence are calculated. If the minimum trace is lower than that of the current sequence, the minimum becomes the new current, and the process repeats until the local minimum is found.

AssemblyLocalSearch$(A_{start}, V, E, \sigma_L)$

$A_{best} \leftarrow A_{start}$
$T_{best} \leftarrow \sigma_L^2 \left.\frac{df_V(L_{E,best})}{dL_{E,best}}\right|^2_{\bar{L}_{E,best}}$
$Min \leftarrow False$
while $Min = False$ **do**
  $Min \leftarrow True$
  $A_{adjacents} \leftarrow$ **AdjacentSequences**(A_{best}, V, E)
  for all $A_{candidate} \in A_{adjacents}$ **do**
    $T_{candidate} \leftarrow \sigma_L^2 \left.\frac{df_V(L_{E,candidate})}{dL_{E,candidate}}\right|^2_{\bar{L}_{E,candidate}}$
    if $T_{candidate} < T_{best}$ **then**
      $Min \leftarrow False$
      $A_{best} \leftarrow A_{candidate}, T_{best} \leftarrow T_{candidate}$
return A_{best}

Algorithm 12 To find a suitable assembly sequence that is locally optimal: choose a central starting triangle, find an assembly sequence that greedily assembles to minimize trace, then perform a local search until the sequence is optimal among its neighbors.

FindLocallyOptimalSequence$(\bar{\mathbf{X}}, V, E, \sigma_L, \sigma_M, h)$

$\Delta_{central} \leftarrow$ RandomChoice(**AllCentralTriangles**(V, E))
$A_{greedy} \leftarrow$ **GreedyAssembly**$(V, E, \Delta_{greedy}, \sigma_L)$
$A_{best} \leftarrow$ **AssemblyLocalSearch**$(A_{greedy}, V, E, \sigma_L)$
return A_{best}

sequences. The neighboring sequences for a given sequence is generated by **AdjacentSequences**.

6.3 Runtime

Lemma 6. ***GreedyAssembly*** *is* $O(N^3)$*, and can be optimized to* $O(N^2)$*.*

Proof. **GreedyAssembly** makes N calls to the assembly loop until assembly is finished. Using the same tweak to optimize the runtime of **RandomSequence**, looking only for the next steps attached to the node previously attached, the **PossibleNextNodes** step can be considered $O(1)$, but even when unoptimized at $O(N)$, it does not contribute significantly to the overall runtime.

Finding the covariance trace is $O(N^2)$; for each of the $3N - 6$ lengths, $\mathcal{F}_V$ is called twice

to generate two modified structures $\mathbf{X}_{\pm h}$, each an $O(N)$ call. This can be reduced to $O(N)$ time using the additive properties of trace established in Equations 5.12 and 5.13. By saving $\mathbf{X}_{\pm h}$ for all lengths each step, and adding the new node to each, the number of calculations is reduced to the number of saved modified lengths plus the new modified lengths, which is $2(3N-6)+6$. All told, **GreedyAssembly** is $O(N^3)$, or $O(N^2)$ when the covariance trace optimization is used. □

Lemma 7. ***AssemblyLocalSearch*** *is $O(S(N)N^3)$, where $S(N)$ is the number of swaps required to find the local minimum.*

Proof. Before the while loop, the covariance trace requires $O(N^2)$ time. Inside the loop, **AdjacentSequences** is called once, and is $O(N^2)$ The inner for loop in **AssemblyLocalSearch** runs $O(N)$ times for the adjacent sequences, calculating the covariance trace each time, making the interior of the while loop $O(N^3)$.

It is possible to reduce the covariance trace calculation time by saving and reusing the traces of unaffected nodes, as done before. However, this tactic is not as powerful for **AssemblyLocalSearch**; the entire covariance needs to be recalculated for structures with different starting triangles, and for swapped steps, on average it will be a step in the middle of the sequence, meaning that the derivatives of half the nodes need to be calculated for all $3N-6$ lengths, which is still $O(N^2)$.

The last step is to determine how many times the while loop is executed before the local minimum is found. The simulations in the following sections suggest that the length is roughly linear. However, greedily choosing neighboring sequences may not take a linear path to the local minimum, so I say that the number of swaps is $S(N)$, and **FindLocallyOptimalSequence** is $O(S(N)N^3)$. I conjecture that $S(N) = O(N)$, but have yet to prove it. □

Theorem 7. ***FindLocallyOptimalSequence*** *is $O(S(N)N^3)$, where $S(N)$ is the number of swaps required to find the local minimum in* ***AssemblyLocalSearch****.*

Proof. The first step is to call **AllCentralTriangles**. As shown in Theorem 5, finding all central triangles takes $O(N^2)$ time. **GreedyAssembly** is $O(N^2)$ when optimized, as shown in Lemma 6.

□

6.4 Assembly Sequencing Results

To estimate the σ_L process noise, I measured the pre- and post-fix lengths of 75 aluminum struts, giving us a process noise of $\sigma_L^2 = 3.57 \times 10^{-7} m^2 = 0.357 mm^2$. The ruler variance σ_M^2 was based on visual estimation of length consistently being within $0.5mm$ of the actual length; if $0.5mm$ is considered the two-sigma length, $\sigma_M^2 = 6.25 \times 10^{-8} m^2 = 0.0625 mm^2$.

The assembly sequencing simulations were performed on a 45-node subset of the 84-node telescope, chosen for the faster simulation runtimes while showing trends. The 45-node telescope has 1260 possible starting triangles, 72 of which are central based on the minimum parallel order length. I used **FindLocallyOptimalSequence** and compared the resulting orders at each step. Additionally, I tweaked the algorithm to allow any triangle to be chosen and compared this to the central triangle approach. The results shown are derived from:

- Any random sequence
- Any random sequence from a central starting triangle
- Greedy assembly from the starting triangles used in any random sequence
- Greedy assembly from the central starting triangles
- Local search from greedy assembly from any starting triangle
- Local search from greedy assembly from central starting triangles

The results are shown in Figure 6.1. The top row shows the distribution of traces of 50 trials with random sequences starting from any starting triangle (left) or from a central starting triangle (right). Both histograms show the **trace logarithms** due to the very large range of trace variances. Random sequences for the telescope almost always include nearly degenerate tetrahedra, which contribute enormous errors to the overall trace; while human assembly planners would naturally

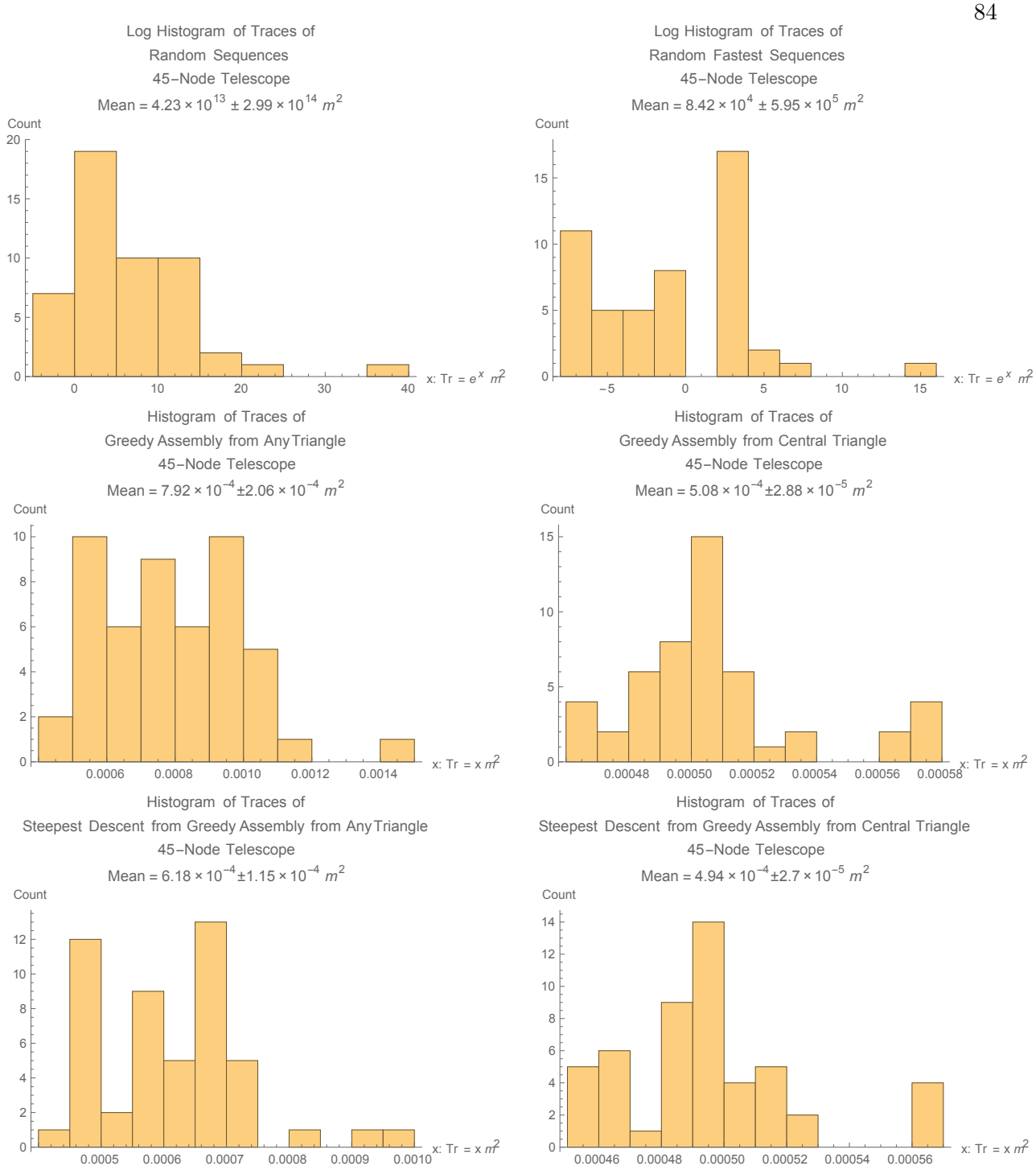


Figure 6.1: Left column: random sequences starting from any starting triangle. Right column: random sequences starting from a central starting triangle. Top row: Random assembly, shown as a log trace histogram due to the large range of values. Middle row: Greedy assembly. Bottom row: Steepest descent starting from greedy assembly.

avoid the degenerate steps, the topology permits their existence as possible assembly steps. The mean traces of these random sequences blew up as a result, making their values meaningless. Therefore, random sequences are not suitable for precision assembly.

Using the same starting triangles as the random sequence experiment, the greedy assembly algorithm is an enormous improvement, and avoids degenerate tetrahera. These results are shown in the top row of Figure 6.1, starting from any starting triangle (left) or from a central starting triangle (right). The means and standard deviations are $7.92 \times 10^{-4} \pm 2.06 \times 10^{-4} m^2$ for any starting triangle, and $5.08 \times 10^{-4} \pm 2.88 \times 10^{-4} m^2$ for central triangles. On average, greedy assembly on central starting triangles is 1.56 times more precise than assembly from any triangle, further justifying building from the center outward.

The last optimization, local search, was performed with the greedy sequences. The results are in the bottom row of Figure 6.1, again starting from any starting triangle (left) or from a central starting triangle (right). The trajectories taken from the greedy to the local minimum are shown in Figure 6.2. The means and standard deviations are $6.18 \times 10^{-4} \pm 1.15 \times 10^{-4} m^2$ for any starting triangle, and $4.94 \times 10^{-4} \pm 2.70 \times 10^{-5} m^2$ for central triangles. For the cases starting from any triangle, locating a local minimum results in an average improvement of a factor of 1.25, requiring an average of 14.28 step changes before finding the minimum. The improvement for central triangle steepest descent is smaller: a factor of 1.03, making an average of 6.96 steps. The reduced number of steps is due to the central triangle greedy sequence starting closer to a minimum. After steepest descent, starting from a central triangle is on average 1.25 times more precise than starting from anywhere. The central triangle steepest descent sequences never found a better starting triangle than a central one, while 20% of the descents starting from any triangle changed their starting triangles to one of the central triangles.

In all of the trace histograms, there are clusters with larger variance than the others due to some of the assembly sequences having a skewed tetrahedral cell contributing a large variance. While these are not so skewed as to blow up the trace calculation, the fact that such cells were found by greedy assembly and subsequently unchanged by the local search shows that multiple

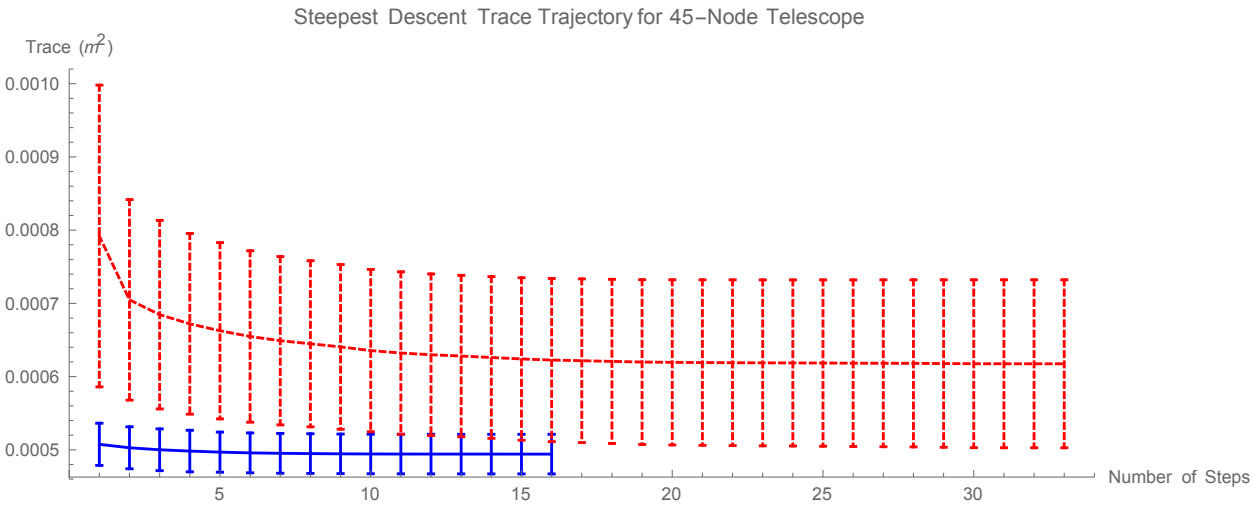


Figure 6.2: The trajectories taken by the local search algorithm, starting a greedy assembly on any starting triangle (orange) and a central starting triangle (blue), averaged and extended in step count to the longest such trajectory. Descending from a greedy sequence generated from a central starting triangle finds better local minima in fewer steps.

random restarts should be used to find sequences without this problem.

Chapter 7

Assembly with Simultaneous Localization and Mapping

The previous two chapters introduced a linear model of the assembly process and the measurement process, enabling me to derive closed forms for the propagation of assembly error, and using that information to derive algorithms for finding optimal assembly sequences. I showed that the linear models work quite well for the hardware I present in this thesis given that assembly lengths are decided in advance. To control assembly by adjusting the lengths as the assembly proceeds to offset the estimated structure, I chose to explore this using nonlinear estimation techniques. This chapter describes the use of mid-assembly measurements, and compares four estimation models: a Linear Least Squares model based on the canonical form, the Extended Kalman Filter, the Unscented Kalman Filter, and the Maximum Likelihood Estimator.

Let $\hat{\mathbf{X}}$ be the estimate of the made nodes of the truss structure, where the estimation function is arbitrary (linear or otherwise). Continuing to build with the predetermined lengths given by $\bar{\mathbf{X}}$ would be inappropriate. My strategy is to use the estimated base nodes as the "correct" values when calculating the lengths needed to reach the goal, so that the expected value of the base and the expected value of the lengths combine to put the expected value of the float node at desired spot.

The tetrahedron function in Equation 3.3 is modified to use the estimates of the base positions $\hat{X}_{ijk}$, and choosing the lengths $\hat{L}_{ijk}$ so that X_f is nominally where it should be:

$$\begin{aligned} X_f &= \mathcal{F}_f(\hat{X}_i, \hat{X}_j, \hat{X}_k, \hat{L}_{i,f}, \hat{L}_{j,f}, \hat{L}_{k,f}) \\ \text{where } \hat{L}_{i,f} &= \|X_f - \hat{X}_i\| \end{aligned} \tag{7.1}$$

With measurements applied at every step, the resultant estimation should be an improvement over measurement-free case, but the adjustment of desired strut lengths at every step based on all of the measurements before it complicates the estimation process since the measurements feed back into the state function. No longer can the covariance matrix be represented in the convenient form $J(\sigma_L^2 I)J^T$.

The following sections compare the Linear Least Squares, the Extended Kalman Filter, the Unscented Kalman Filter, and the Maximum Likelihood Estimator. In following with the capabilities of the IPJRs, this chapter will only focus on length measurements, and only on struts that were recently added, but the methods presented can accept other forms of measurement of any nodes without modification. I also compared two variations of which measurements are chosen: the measurements only of the assembly struts themselves, or the measurements of the assembly struts and the passive struts.

7.1 Linear Least Squares

The linear least squares utilizes the canonical forms first described in Section 5.3.1 and is shown in Algorithm 13. It is called "linear least squares" due to the model being equivalent to the least squares derivation of the Kalman Filter [11]. This filter is almost identical to the Kalman Filter, the only distinction being that at each step in the assembly process, the complete structure canonical form is reduced by the total set of measurements instead of an iterative succession of measurements and actions. There is no propagation through the nonlinear model as there is in the Extended and Unscented Kalman Filters, so I expected this filter to be the least accurate. If $\mathcal{F}$ was a linear function, this would be an optimal estimate, but for the nonlinear case it is not optimal.

The inputs are L_K, the length used during the assembly process, including the lengths that

Algorithm 13 This algorithm returns a linear least squares estimate of a structure with the given lengths L_K (not necessarily the nominal lengths but the ones used during assembly), the expected value of measurements and the measurements themselves, the variances, and a step size for finding the assembly covariance.

LinearLeastSquaresEstimation$(L_K, M_K, A, \sigma_L, \sigma_M, h)$

1: $\bar{\mathbf{X}}_L \leftarrow \mathcal{F}_K(L_K)$
2: $\Sigma_{\bar{\mathbf{X}}_L} \leftarrow$ Apply Equation 5.5 to L_K at $\bar{\mathbf{X}}_L$
3: $\mu_{M_K} \leftarrow \|\bar{X}_i - \bar{X}_j\| \forall (i,j) \in M_K$
4: $k_\Sigma, h_\Sigma \leftarrow$ Apply Equation 5.20 to $\bar{\mathbf{X}}_L, \Sigma_{\bar{\mathbf{X}}_L}$
5: $J_M \leftarrow \frac{d(\|\bar{X}_i - \bar{X}_j\|)}{dX_{i,j,predicted}}\Big|_{\mu_{M_K}} \forall (i,j) \in M_K$ (Equation 5.30)
6: $k_{M|X}, h_{M|X} \leftarrow$ Equation 5.27 with $\mathcal{N}(\mu_{M_K} + J_M(X_{all} - \bar{\mathbf{X}}_L), J_M(\sigma_M^2 I)J_M^T)$
7: $k_{X,M}, h_{X,M} \leftarrow k_\Sigma + k_{M|X}, h_\Sigma + h_{M|X}$
8: $k_{all,M=M_K}, h_{all,M=M_K} \leftarrow$ Equation 5.26 with measurements M_K
9: $\hat{\mathbf{X}}, \Sigma_{\hat{\mathbf{X}}} \leftarrow$ Equation 5.21
10: **return** $\hat{\mathbf{X}}, \Sigma_{\hat{\mathbf{X}}}$

are modified from the desired lengths due to previous estimates. These lengths, and the assembly sequence A and step size h, are used to generate predicted structure (called "predicted" to distinguish from "nominal"), which gives the mean and covariance matrix of that structure: $\bar{\mathbf{X}}_L, \Sigma_{\bar{\mathbf{X}}_L}$. M_K is the set of measured values. The process noise σ_L and measurement noise σ_M are also required.

First, the predicted structure mean and covariance are found, as are the measurement means μ_{M_K}. These are converted into the canonical form. Then, the Jacobian of all of the measurement functions (norms between nodes) with respect to the nodes is found and evaluated at the mean. The Jacobian J_M is then used to calculate the measurement covariance matrix by $J_M(\sigma_M^2 I)J_M^T$, and the linear system $\mu_{M_K} + J_M(X_{all} - \bar{\mathbf{X}}_L)$, both of which are used to find the linear conditional canonical forms. The structure and measurement canonical forms are multiplied using the summation in Equation 5.22, then the canonical reduction is performed by setting the measurement variables to M_K. The result is transformed back into the estimated mean and covariance.

7.2 Extended Kalman Filter

The Extended Kalman Filter was quickly developed after the introduction of the Kalman filters in the 1960s to allow nonlinear systems to be modeled. While in the decades since, improvements have been made (Particle Filters, Unscented Kalman Filters, GraphSLAM and FastSLAM to name a few), EKF remains popular for use in SLAM and estimation due to its relatively simple implementation, and its usefulness in near-linear problems. The most helpful source on the implementation of EKF is in a paper arguing for why it is obsolete [63]. Unlike Algorithm 13, the EKF in Algorithm 14 takes as input prior estimated means and covariances and returns updated means and covariances. Also, unlike Algorithm 13, instead of a complete structure covariance being calculated as a function of only the lengths, the covariances of each added node are linear conditionals depending on the variances of the struts attached to that node, and the covariances of the base nodes (See Appendix B for discussion).

When **ExtendedKalmanFilter** is called for the first time, the prior mean and covariance can be initialized to 0, otherwise it is called with the results of the previous call. The parameter $\bar{\mathbf{X}}_f$ is the desired node position for the current added node f, $L_{f,B}$ are the lengths to its base, $M_{f,neighbors}$ are the measurements between f and any or all of its neighbors, **including** nodes not part of the base but which share a redundant strut with f. The process and measurement noises σ_L, σ_M are passed in as well. The notation $f, neighbors$ indicates that the set may include not only the assembly struts, but also the passive struts.

The first step calculates the predicted structure $\hat{\mathbf{X}}_{t,t-1}$ when the previous estimate is passed through the strut/triangle/tetrahedron functions without added noise (and since $\hat{\mathbf{X}}_{t,t-1}$ is the full structure vector, the other nodes remain unchanged and are set to what they were before). The derivative of nodes with respect to nodes F is then calculated as 1 on the diagonal for fixed nodes to reflect their unchanging values, and F_f is the derivative of the node function $\mathcal{F}_f$ from Equation 3.2. Matrix G is calculated as 0 for fixed nodes, and the derivative of the node function with respect to strut lengths for f. These two matrices are then used to predict the next covariance matrix,

Algorithm 14 The Extended Kalman Filter, applied to structures. This makes heavy use of the functions described in Appendix A and **not** the complete node function as a function of lengths $\mathcal{F}_V(L_E)$. Because the addition of a new node affects only itself and not the other nodes, the fixed nodes remain unchanged, and vary only with respect to themselves.

ExtendedKalmanFilter$(\hat{\mathbf{X}}_{t-1,t-1}, \Sigma_{\hat{\mathbf{X}}_{t-1,t-1}}, \bar{\mathbf{X}}_f, L_{f,B}, M_{f,neighbors}, \sigma_L, \sigma_M)$

$X_{f,t,t-1} \leftarrow \mathcal{F}_f(X_{B,t-1,t-1}, L_{f,B})$, using Equation 3.2
$X_{i,t,t-1} \leftarrow X_{i,t-1,t-1} \forall i \neq f$
$\hat{\mathbf{X}}_{t,t-1} \leftarrow X_{i,t,t-1} \forall i$
$F \leftarrow I$, square matrix of dimension $\|X\|$
$F_f \leftarrow \frac{dX_f}{dX_B}\Big|_{\hat{\mathbf{X}}_{t-1,t-1}, L_{f,B}}$ using Equations A.36-A.40
$G \leftarrow 0$
$G_f \leftarrow \frac{dX_f}{dL_{f,B}}\Big|_{\hat{\mathbf{X}}_{t-1,t-1}, L_{f,B}}$ using Equations A.28-A.33
$\Sigma_{\hat{\mathbf{X}}_{t,t-1}} \leftarrow F\Sigma_{\hat{\mathbf{X}}_{t-1,t-1}}F^T + G(\sigma_L^2 I)G^T$
$M_{t,t-1} \leftarrow \|X_f - X_i\||_{\hat{\mathbf{X}}_{t,t-1}} \forall i \in neighbors$
$H \leftarrow \frac{d\|X_f - X_i\|}{dX_{f,i}}\Big|_{\hat{\mathbf{X}}_{t,t-1}} \forall i \in neighbors$
$U \leftarrow \frac{d(\|X_f - X_i\| + v)}{dv}\Big|_{\hat{\mathbf{X}}_{t,t-1}} \forall i \in neighbors$, where v is a measurement noise variable
$K \leftarrow \Sigma_{\hat{\mathbf{X}}_{t,t-1}} H^T (H\Sigma_{\hat{\mathbf{X}}_{t,t-1}} H^T + U(\sigma_M^2 I)U^T)^{-1}$
$\hat{\mathbf{X}}_{t,t} \leftarrow \hat{\mathbf{X}}_{t,t-1} + K(M_{f,neighbors} - M_{t,t-1})$
$\Sigma_{\hat{\mathbf{X}}_{t,t}} \leftarrow (I - KH)\Sigma_{\hat{\mathbf{X}}_{t,t-1}}$
return $\hat{\mathbf{X}}_{t,t}, \Sigma_{\hat{\mathbf{X}}_{t,t}}$

$\Sigma_{\hat{\mathbf{X}}_{t,t-1}}$. Then the measurements are accounted for. The predicted measurement $M_{t,t-1}$ is made on the predicted structure with no noises. Matrix H is the derivative of the length measurement function with respect to node positions evaluated at the predicted structure, and U is the derivative of the length measurement function with respect to added noise on the measurement functions, also evaluated at the predicted structure. These can be used to find the Kalman gain K, which then provides a corrected structure and covariance, $\hat{\mathbf{X}}_{t,t}, \Sigma_{\hat{\mathbf{X}}_{t,t}}$ when used on the actual measurements $M_{f,neighbors}$.

7.3 Unscented Kalman Filter

The Unscented Kalman Filter [65, 28] is a recent development, improving EKF's first order approximation to a second order approximation. This made it appealing to implement, since the

EKF can perform poorly when nonlinearities add up. The implementation shares some core fundamentals with the EKF, such as the updating of the state estimate and covariance matrices. Instead of relying on Jacobians, the covariance is approximated by a set of "sigma points" surrounding the mean, which are propagated through the state and measurement functions to provide a better covariance estimate. In this regard, the UKF is similar to the particle filter, but uses only $O(N)$ sigma points, discards the sigma points after use, and only predicts unimodal distributions. As with the EKF, the UKF predicts the state, covariance, and measurements, and corrects these with the actual measurements.

The UKF takes as inputs the prior structure estimate and covariance matrices, the set lengths for the recent addition (f, B_f), the measurements for f and its neighbors, the length and measurement variances. The parameters $\alpha, \beta, \kappa, \lambda$ are tuning factors set to the values recommended in [65].

Each time the function is run, the prior state and covariance matrices are **augmented** with zeros and diagonal covariances representing the total number of assembly struts and the total set of measurements measurements. L is the total length of the augmented state. The square root of the augmented prior covariance, $\Sigma_{\surd}$, scaled by $(L + \lambda)$, forms the basis for calculating the sigma points. Each row of the square root represents a vector that is added to the mean to produce a sigma point. [1] .

The list of sigma points is $\mathbf{X}_{augmented,t-1,t-1}$, indexed by i. The sigma point set is initialized with the mean itself. Then each **row** of $\Sigma_{\surd}$ is both added to and subtraced from the mean, resulting in $2L + 1$ points in $\mathbf{X}_{augmented,t-1,t-1}$. Associated with each sigma point is a mean weight and a covariance weight, which are grouped under W_x and W_σ.

The sigma points are then propagated through the node function Equation 3.2 if the node is the just-added floating node, or are left unchanged from the prior if it is a fixed node; the

[1] While there are multiple ways to do this ([65] suggests Cholesky decomposition), the way I chose was to use Mathematica's **MatrixPower** function, which returns matrices A such that $A^T A = B$. Because the covariance values for unplaced nodes is 0, the covariance is not positive definite (only semi-definite) and Cholesky decomposition fails. Additionally, when this problem was fixed by adding dummy variance values, Cholesky would still occasionally fail, prompting me to switch to **MatrixPower**.

Algorithm 15 The Unscented Kalman Filter predicts and corrects the structure state and covariance by propagating a set of sigma points (which combine node variances and length variances) through the node function, and then taking measurements (using measurement variances) to establish a predicted state and covariance, which is then corrected by using the actual measurements.

UnscentedKalmanFilter$(\hat{\mathbf{X}}_{t-1,t-1}, \Sigma_{\hat{\mathbf{X}}_{t-1,t-1}}, \bar{\mathbf{X}}_f, L_{f,B}, M_{f,neighbors}, \sigma_L, \sigma_M)$

$\alpha \leftarrow 0.001, \beta \leftarrow 2, \kappa \leftarrow 0$

$L \leftarrow \|\hat{\mathbf{X}}_{t-1,t-1}\| + 3N - 6 + \|M_E\|$, for the total number assembly struts and measurements

$\lambda \leftarrow \alpha^2(L+\kappa) - L$

$\mathbf{X}_{augmented,t-1,t-1} \leftarrow (\hat{\mathbf{X}}_{t-1,t-1}, 0_{3N-6}, 0_{\|M_E\|})$

$\Sigma_{\mathbf{X}_{augmented,t-1,t-1}} \leftarrow \begin{pmatrix} \Sigma_{\hat{\mathbf{X}}_{t-1,t-1}} & 0 & 0 \\ 0 & \sigma_L^2 I_{3N-6} & 0 \\ 0 & 0 & \sigma_M^2 I_{\|M_E\|} \end{pmatrix}$

$\Sigma_{\surd} \leftarrow \sqrt{(L+\lambda)\Sigma_{\mathbf{X}_{augmented,t-1,t-1}}}$, where the matrix square root of B is $A^T A = B$

$\mathcal{X}_{augmented,t-1,t-1} \leftarrow \{\mathbf{X}_{augmented,t-1,t-1}\}$

$W_x \leftarrow \{\frac{\lambda}{L+\lambda}\}$

$W_\Sigma \leftarrow \{\frac{\lambda}{L+\lambda} + (1 - \alpha^2 + \beta)\}$

for all rows $X_{i,\Sigma_{\surd}} \in \Sigma_{\surd}$ **do**

  $\mathcal{X}_{augmented,t-1,t-1} \leftarrow \mathcal{X}_{augmented,t-1,t-1} \cup \{\mathbf{X}_{augmented,t-1,t-1} + X_{i,\Sigma_{\surd}}, \mathbf{X}_{augmented,t-1,t-1} - X_{i,\Sigma_{\surd}}\}$

  $W_x \leftarrow W_x \cup \{\frac{1}{2(L+\lambda)}, \frac{1}{2(L+\lambda)}\}$,$W_\Sigma \leftarrow W_\Sigma \cup \{\frac{1}{2(L+\lambda)}, \frac{1}{2(L+\lambda)}\}$

$\mathcal{X}_{t,t-1} \leftarrow \emptyset$

for all Sigma points $i \in \mathcal{X}_{augmented,t-1,t-1}$ **do**

  $(\mathbf{X}^i_{nodes,t-1,t-1}, \mathbf{X}^i_{lengths,t-1,t-1}) \leftarrow \mathcal{X}^i_{augmented,t-1,t-1}$

  $\mathbf{X}^i_{nodes,t,t-1} \leftarrow \mathcal{F}_f(X^i_{B,t-1,t-1}, L_{f,B})$ for f, and $X^i_{j,t-1,t-1}$ if $j \neq f$, using Equation 3.2

  $\mathcal{X}_{t,t-1} \leftarrow \mathcal{X}_{t,t-1} \cup \{\mathbf{X}^i_{nodes,t,t-1}\}$

$\hat{\mathbf{X}}_{t,t-1} \leftarrow \sum_i W^i_x \mathcal{X}^i_{t,t-1}$

$\Sigma_{\hat{\mathbf{X}}_{t,t-1}} \leftarrow \sum_i W^i_\Sigma (\mathcal{X}^i_{t,t-1} - \hat{\mathbf{X}}_{t,t-1})(\mathcal{X}^i_{t,t-1} - \hat{\mathbf{X}}_{t,t-1})^T$

$\mathcal{M}_{t,t-1} \leftarrow \emptyset$

for all Sigma points i, $\mathbf{X}^i_{nodes,t,t-1} \in \mathcal{X}_{t,t-1}, X^i_{M,t-1,t-1} \in \mathcal{X}_{augmented,t-1,t-1}$ **do**

  $M^i_{t,t-1} \leftarrow \|X^i_{f,t,t-1} - X^i_{b,t,t-1}\| + X^i_{M_{f,b},t-1,t-1} \forall b \in neighbors$

  $\mathcal{M}_{t,t-1} \leftarrow \mathcal{M}_{t,t-1} \cup \{M^i_{t,t-1}\}$

$M_{t,t-1} \leftarrow \sum_i W^i_x \mathcal{M}^i_{t,t-1}$

$\Sigma_{MM} \leftarrow \sum_i W^i_\Sigma (\mathcal{M}^i_{t,t-1} - M_{t,t-1})(\mathcal{M}^i_{t,t-1} - M_{t,t-1})^T$

$\Sigma_{XM} \leftarrow \sum_i W^i_\Sigma (\mathcal{X}^i_{t,t-1} - \hat{\mathbf{X}}_{t,t-1})(\mathcal{M}^i_{t,t-1} - M_{t,t-1})^T$

$K \leftarrow \Sigma_{XM} \Sigma^{-1}_{MM}$

$\hat{\mathbf{X}}_{t,t} \leftarrow \hat{\mathbf{X}}_{t,t-1} + K(M_{f,neighbors} - M_{t,t-1})$

$\Sigma_{\hat{\mathbf{X}}_{t,t}} \leftarrow \Sigma_{\hat{\mathbf{X}}_{t,t-1}} - K \Sigma_{MM} K^T$

return $\hat{\mathbf{X}}_{t,t}, \Sigma_{\hat{\mathbf{X}}_{t,t}}$

propagated sigma points are $\mathcal{X}_{t,t-1}$. Then the predicted state $\hat{\mathbf{X}}_{t,t-1}$ can be found by multiplying

each propagated sigma point by its weight. Likewise, the predicted covariance $\Sigma_{\hat{\mathbf{X}}_{t,t-1}}$ can be found by calculating the sample covariance, with each sample weighted.

A similar process occurs for measurements: the predicted sigma points $\mathcal{X}_{t,t-1}$ and the measurement offsets $X^i_{M,t-1,t-1}$ are combined to find predicted sigma point measurements $\mathcal{M}_{t,t-1}$. The mean of the measurements is found in the same way as for the state mean, along with the covariance of the measurements Σ_{MM} and the covariance of the measurements and the state Σ_{XM}. From here, the calculations are analogous to the EKF; the Kalman gain K is calculated, which then is used to modify the predicted state based on the actual set of measurements, and the updated $\hat{\mathbf{X}}_{t,t}, \Sigma_{\hat{\mathbf{X}}_{t,t}}$ are returned.

7.4 Maximum Likelihood Estimator

The Maximum Likelihood Estimator is conceptually simple, but depends on a gradient ascent of a joint probability distribution function (also called the likelihood function), which typically is in a large dimensional space, and is approximately 0 nearly everywhere except near the mean. The standard method to mitigate this problem is to find maximum of the logarithm of the likelihood function. Since the number of variables in the joint PDF is $3N - 6 + M$ (in these experiments, on the order of hundreds of variables), the state space is not too large to rule out log likelihood gradient ascent. The log likelihood of a normal distribution function is:

$$\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}\right) = \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{(x-\mu)^2}{2\sigma^2} \tag{7.2}$$

Since the first term on the right does not vary with $\mathbf{X}$, this can be ignored. Recall Equation 5.2:

$$p(\mathbf{X}) = \prod_{i,j \in E_A} p(\|X_i - X_j\| - \bar{L}_{i,j}) \tag{7.3}$$

The process noises on each edge are independent, as are the measurements. For a substructure K, this becomes:

Algorithm 16 The maximum likelihood estimation function calculates the log likelihood function, that is, the logarithm of the joint probability distribution of the length functions and the measurement functions over the node variables. Then, using a gradient ascent algorithm (I used Mathematica's **FindMaximum**), and starting at the predicted structure, find the local maximum, and return it.

MaximumLikelihoodEstimation$(L_K, M_K, A_K, \sigma_L, \sigma_M)$

1: $\bar{X} \leftarrow \mathcal{F}_{V_K}(L_{K,A})$
2: $p_{log} = \sum_{f,b \in B,(f,B) \in A_K} -\frac{(\|X_f - X_b\| - L_{f,b})^2}{2\sigma_L^2} + \sum_{i,j \in M_K} -\frac{(\|X_i - X_j\| - M_{i,j})^2}{2\sigma_M^2}$
3: $\hat{X} \leftarrow$ **GradientAscent**(p_{log}) starting at $\bar{X}$
4: **return** $\hat{X}$

$$p_{\log} = \sum_{f,b \in B,(f,B) \in A_K} -\frac{(\|X_f - X_b\| - L_{f,b})^2}{2\sigma_L^2} + \sum_{i,j \in M_K} -\frac{(\|X_i - X_j\| - M_{i,j})^2}{2\sigma_M^2} \tag{7.4}$$

And the most likely state $\hat{X}$ is

$$\hat{X}_K = \underset{X_K}{\operatorname{argmax}}(p_{log}) \tag{7.5}$$

With this form, **MaximumLikelihoodEstimation**, Algorithm 16, can be described. The mean lengths L_K and collected measurements M_K are passed in, along with the assembly sequence, the variances, and the step size. One obvious method to solve for the maximum likelihood is to solve the derivative $\frac{dp_{log}}{d\mathbf{X}} = 0$, but this results in an unwieldy system of equations due to the derivative of each term having the form $-\frac{(v_i - v_j)(\|X_i - X_j\| - V_{i,j})}{\sigma^2 \|X_i - X_j\|}$, where v stands in for $x, y,$, or z, and V stands in for L or M.

It is computationally simpler to find the maximum numerically. Mirror solutions are a problem, as previously discussed for $\mathcal{F}$. There exists a local maximum for every combination of mirrored cells, so it is important to start in the neighborhood of the correct solution. The solution is to start the search at the length-wise nominal structure $\mathbf{X}_L$ [2] . The expression for p_{log} is then found with all node positions x_i, y_i, z_i. Finally, a gradient ascent function can be used to find the local maximum of p_{log} with respect to the node positions.

[2] Since the lengths L_K may include lengths that were altered from the nominal starting lengths, this is not equal to $\bar{\mathbf{X}}$.

7.5 Comparison of Estimators

I compared the four estimators by simulation using **AssemblyWithEstimation**, Algorithm 17, which performs closed loop assembly. After each new node is added, measurements are taken, the state estimate is updated, and the process continues. The random adjustments to length and measurements are taken from the distributions defined by σ_L, σ_M. In a physical trial, these processes are physical, and the hidden state $\mathbf{X}$ is unseen.

To perform the comparisons, I ran Algorithm 17 for 100 trials using all four estimators on three structure types: the inner 18 nodes of the space telescope truss defined in [66], the inner 45 nodes of the same telescope, and the 20-node triple helix. For the telescopes, I varied the definition of *neighbors* to be either only the set of base nodes, or the entire set of adjacent nodes to the node currently being added (for example, if the added node has 5 neighbors, all 5 are used for measurements). The redundant struts are attached by passive IPJRs that can extend and contract when the node is moved by the active IPJRs, and are only there to measure the lengths. For the triple helix and the small telescope, with strut lengths nominally $1m$, I set $\sigma_L = 0.1m, \sigma_M = 0.01m$, very large numbers to test how well the estimators can handle nonlinearity. For the large telescope, I used the values of $\sigma_L = 0.0015m, \sigma_M = 0.00025m$, on the order of the 3D IPJR variances, to predict how well a physical assembly experiment could be expected to work using the hardware I built.

7.5.1 Telescope Truss Assembly with Estimation

The results of the 18-node simulations are shown in Figures 7.1 and 7.2. The open loop covariance trace of the random sequence is $4.9m^2$; the minimized sequence $2.8m^2$. In both sequences, all of the estimators have similar outcomes with and without measuring passive struts; all estimators vastly improve the results. For the random sequence, the linear least squares estimator performs more poorly, as expected, due to the large nonlinearity of these tests. For the minimized sequence, this trend is not as visible. In any case, the standard deviation of node errors is large enough that

Algorithm 17 A stochastic algorithm for generating structures, generalized and agnostic to which estimator is used. It sets the IPJR lengths based on the estimate of the structure, adds length noise to the node function, then takes measurements of the struts attached to the newest node (including redundant struts if desired), adds measurement noise, then estimates the structure state.

AssemblyWithEstimation$(\bar{\mathbf{X}}, A, M_A, \sigma_L, \sigma_M)$

$\mathbf{X} \leftarrow 0$ is hidden from the estimator
$\hat{\mathbf{X}} \leftarrow 0$
for all $(f, B) \in A, (f, neighbors) \in M_A$ **do**
  $L_{f,B} \leftarrow \|\bar{\mathbf{X}}_f - X_{b,estimate}\| \forall b \in B$
  $X_f \leftarrow \mathcal{F}_f(X_B, L_{f,B} + \mathcal{N}(0, \sigma_L^2 I))$, simulation or IPJR command and welding
  $M_{f,neighbors} \leftarrow \|X_f - X_n\| + \mathcal{N}(0, \sigma_M^2)) \forall n \in neighbors$, simulation or IPJR length measurements
  $\hat{\mathbf{X}} \leftarrow$ Estimation function, parameter list varies based on which estimator used
return $\hat{\mathbf{X}}$

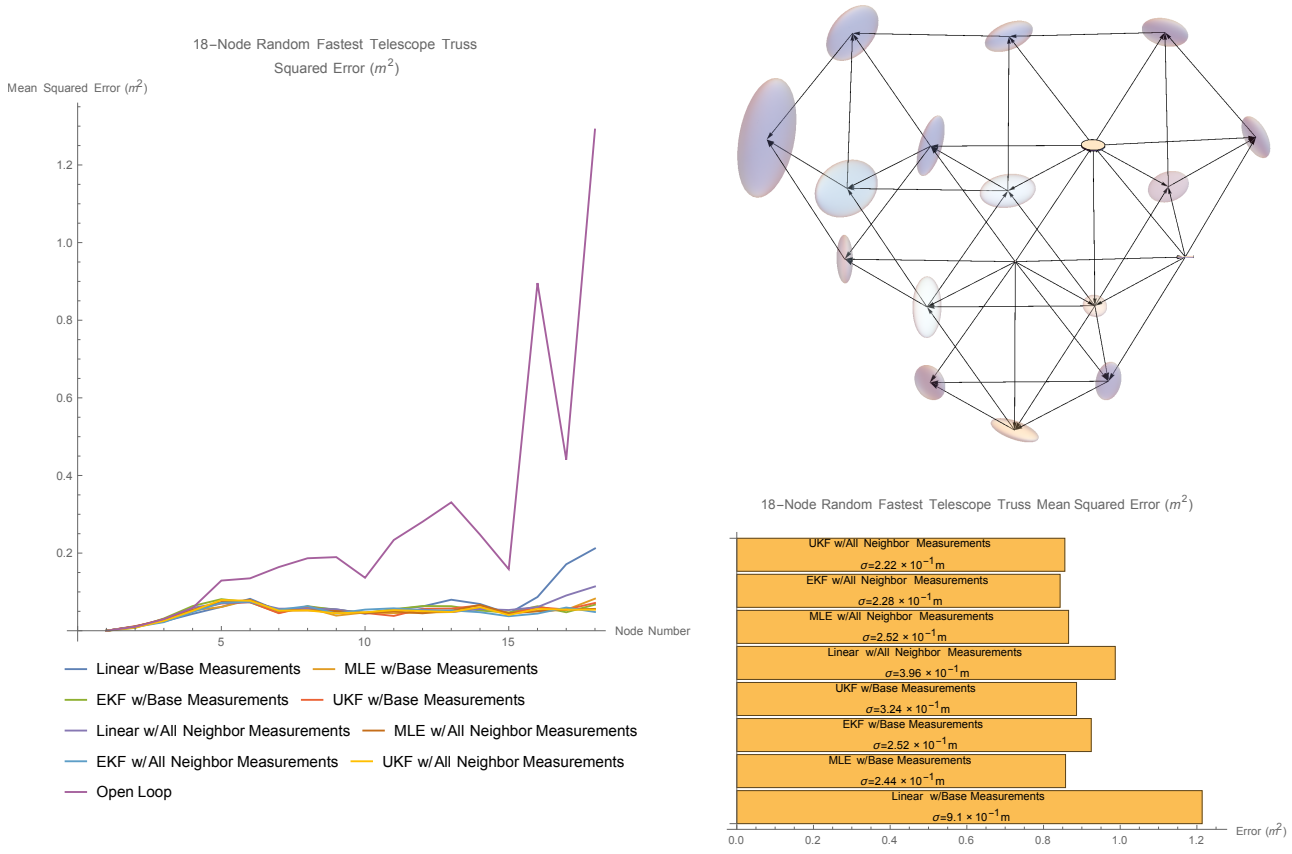


Figure 7.1: For the open loop assembly sequence (upper right) of the 18-node truss with a random sequence, the traces per node of the open loop assembly sequence are compared to the four estimators, which vary by whether passive struts are measured or not (left). Means and sample standard deviations of the node errors are shown (lower right).

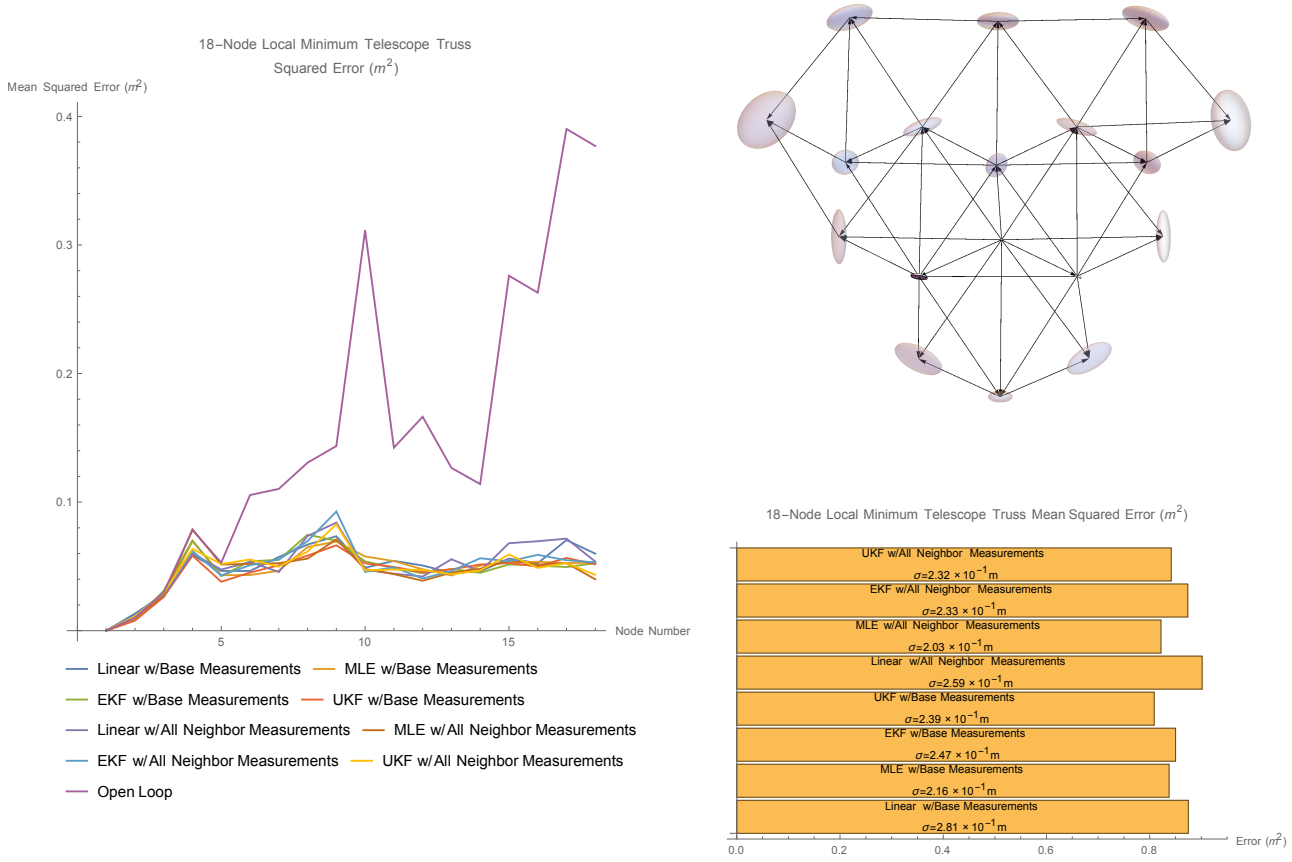


Figure 7.2: For the open loop assembly sequence (upper right) of the 18-node truss with a locally optimal sequence, the traces per node of the open loop assembly sequence are compared to the four estimators, which vary by whether passive struts are measured or not (left). Means and sample standard deviations of the node errors are shown (lower right).

the distinction between linear least squares and the other estimators is only visible in the long run. The estimators also overcome the large difference in open loop trace between the randomized and the minimized sequences. While the random sequence is 1.74 times more imprecise when using open loop, this gap is much smaller when using estimation. For example, when using MLE estimation using all measurements, the local minimum mean squared error is $0.82m^2$, while the random mean squared error is $0.87m^2$.

The results of the 45-node simulations are shown in Figures 7.3 and 7.4. The open loop covariance trace of the random sequence is $3.6 \times 10^{-3}m^2$; the minimized sequence $5.6 \times 10^{-3}m^2$.

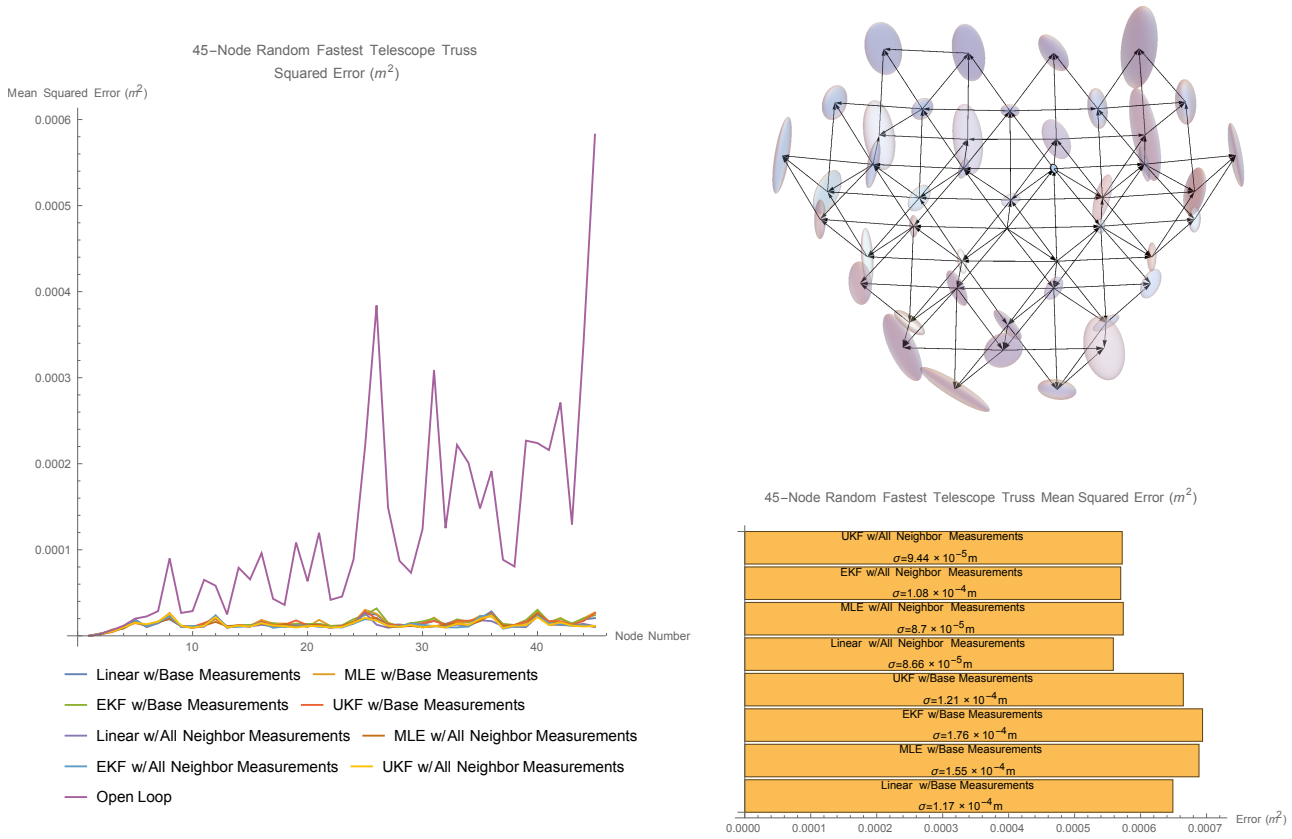


Figure 7.3: For the open loop assembly sequence (upper right) of the 45-node truss with a random sequence, the traces per node of the open loop assembly sequence are compared to the four estimators, which vary by whether passive struts are measured or not (left). Means and sample standard deviations of the node errors are shown (lower right).

Unlike the smaller structure, the benefits of measuring passive strut lengths are clearly visible, especially with the randomized sequence. Also, unlike the 18-node truss, the linear least squares estimator is not noticeably worse; with the small sample size, it appears better. This is due to the smaller variances used in the 45-node tests, and the resultant improvement in the accuracy of linearization. Again, the use of an estimator closes the uncertainty gap between the randomized and the locally minimum sequences. The random sequence is 1.56 times more imprecise when using open loop. But when using MLE and all strut measurements, the local minimum mean squared error is $5.3 \times 10^{-4} m^2$, while the random mean squared error is $5.7 \times 10^{-4} m^2$.

These tests were designed to show a distinction not only between open loop and estimation

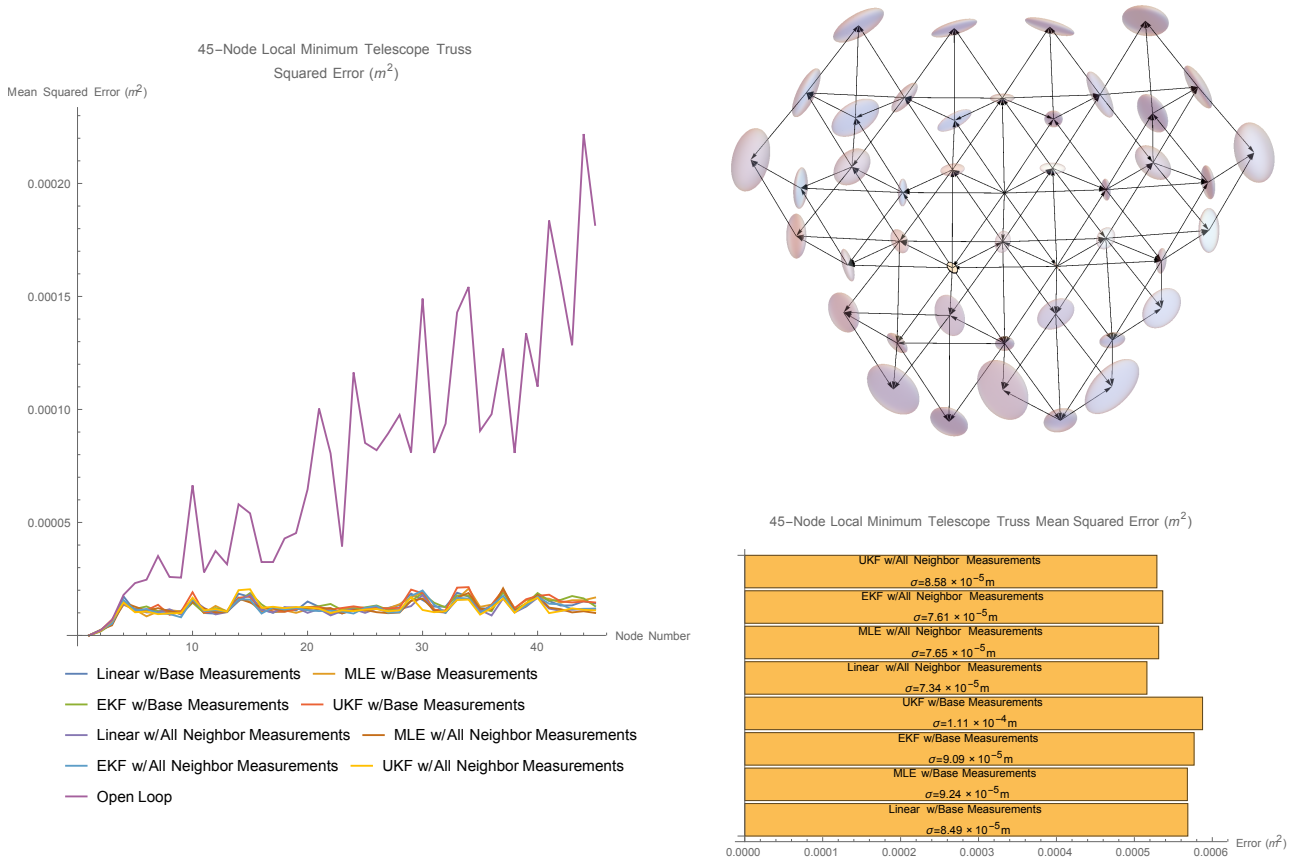


Figure 7.4: For the open loop assembly sequence (upper right) of the 45-node truss with a locally optimal sequence, the traces per node of the open loop assembly sequence are compared to the four estimators, which vary by whether passive struts are measured or not (left). Means and sample standard deviations of the node errors are shown (lower right).

assembly, which is obviously visible, but also between the four estimators and the selection of struts that are measured. While the linear least squares estimator was somewhat worse in the average for the highly nonlinear 18-node tests, the estimators were essentially equivalent when the variances were lower, as in the 45-node tests. The effect of measuring more struts is more visible with larger structures, which is to be expected since larger structures will have more such measurements, and the effects of those measurements are propagated down longer chains of nodes.

7.5.2 Triple Helix Assembly with Estimation

I devised the triple helix assembly test to attempt to force apart the results of the four estimators, since, for the telescope cases, they are mostly equivalent. With a large standard deviation of $0.1m$ for the setting of strut lengths, the node function is very nonlinear, resembling a more extreme version of errors than in the bottom middle image on Figure 5.6. Results are in Figure 7.5. The linear least squares function (blue) proved to be the worst: not only did the results have a large variance, but there is a bias toward shorter structures, increasing the overall mean error. The bias is visible for the Extended Kalman Filter results too (green), but the covariance is reduced. The Uncented Kalman Filter (red) places the nodes closer to the expected mean. The Maximum Likelihood Estimator (orange) shows a smaller covariance and is closest to the mean. Since the least squares and the EKF tend to bias toward a shorter structure, the linearization in both estimators appears to be overestimating the length of the structure, creating shorter lengths than it should. A good way to picture this is to refer to the bottom middle image in Figure 5.6 again: these estimators predict the structure to be at the orange points, but it is actually at the blue points, so the estimators compensate by shrinking the structure. The capable handling of nonlinearities of the UKF seems to overcome the bias, but does not improve on the structure covariance, and instead trends quite closely to the EKF in the node error plot in the lower right of Figure 5.6. Given that the EKF is noticeably biased while the UKF appears not to be, the UKF spreads out the structure points further, which is visible by the slightly larger covariance ellipsoids.

The maximum likelihood estimator is the clear best in every case: it performs the most accurate and precise assemblies. The reason is that, unlike the other estimators, MLE operates on the actual, non-Gaussian probability distribution function. While it does not return an estimated mean and only finds the point that maximizes the PDF, it proves to be a better estimation than the linearization schemes. Additionally, during simulation, the MLE ran the fastest as well, giving MLE every advantage.

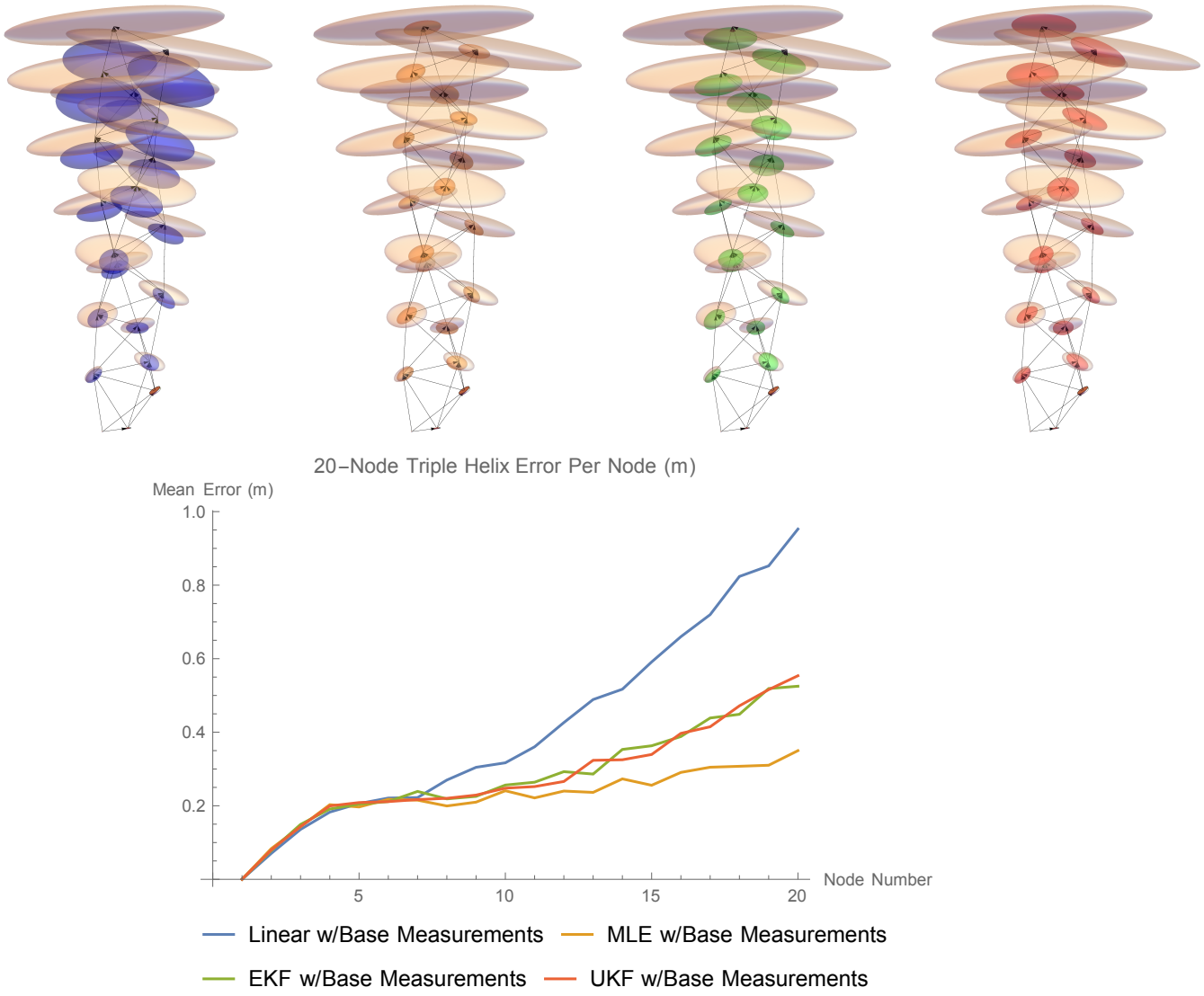


Figure 7.5: Comparisons of the closed loop assembly algorithm using the linear least squares estimator (blue), maximum likelihood estimator (orange), Extended Kalman Filter (green), and Unscented Kalman Filter (red), for $\sigma_L = 0.1m, \sigma_M = 0.01m$. The marginal covariance ellipsoids of the sample covariance matrices for each trial are shown with the marginal covariance ellipsoid of the structure assembled without measurements. Bottom: the mean error per node for each of the estimators.

7.6 Discussion

The results in this chapter establish that the maximum likelihood estimator, used with measurements of the active and passive struts, result in the best outcomes. However, the smaller the

process and measurement variances, the more applicable the Kalman filters and the linear least squares estimators become. For space telescopes with a precision requirement on the order of millimeters or smaller, these methods are essentially equivalent.

Additionally, the use of measurements can almost completely overcome a poor choice in assembly sequence. The difference between two assembly traces is reduced considerably. However, a good assembly sequence should still be chosen using a **AssemblyLocalSearch** on a random fastest assembly sequence to reduce the covariance as much as possible.

Chapter 8

Physical Assembly of Telescope Truss by Intelligent Precision Jigging Robots

This chapter presents the results of the physical assembly trials using the MLE algorithm. I first predicted what the outcomes would be compared to open loop assembly, then physically assembled 3 structures using the open loop algorithm and 3 structures using the MLE algorithm.

8.1 IPJR Hardware Design

In order to forgo premade accurate building materials, I chose commodity telescoping aluminum rods (30" long, 1/4" and 7/32" diameter) that can be locked in place with a shaft collar and that connect to the node balls via neodymium magnets. These design choices make assemblies temporary and allow us to reuse materials in different experiments. Each IPJR is designed to adjust the lengths of the telescoping struts by attaching to them during the assembly of a new part of the structure.

Since these components are reusable, permanent welding is not performed, and fixed struts can be easily adjusted. Thus, to simulate the expected on-orbit application with welding, the strut is measured after it is fixed to the structure, and is considered permanent until the trial is complete.

I built five IPJRs for use in this experiment, but only used 3 at a time. Each IPJR, shown in Figure 8.1, is an autonomous robot, consisting of a Raspberry Pi Model B for high level algorithmic control and communications, an Arduino for actuation and sensing, a custom motor driver board, and an Edimax WiFi dongle. The principal actuator is a Firgelli L-16 linear actuator with $140mm$ extension and potentiometer length feedback discretized to actuator steps of length

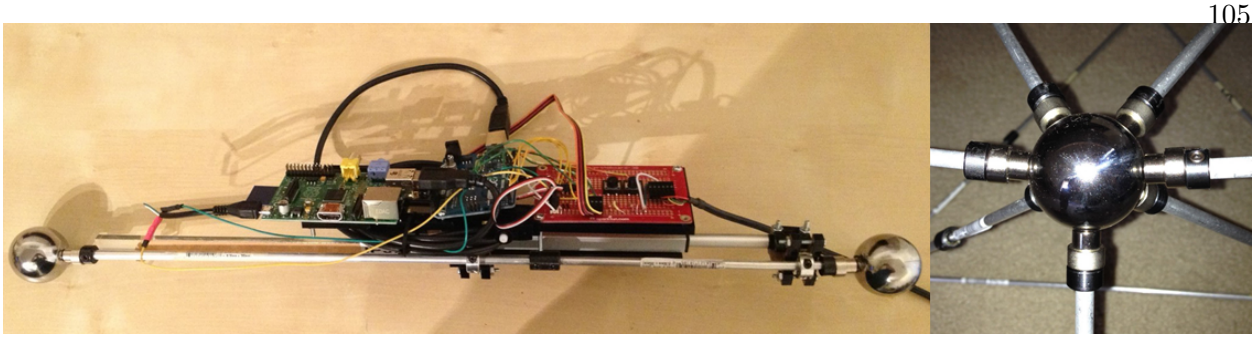


Figure 8.1: The Intelligent Precision Jigging Robot model, for use with telescoping aluminum struts and 3D trusses, is shown attached to a strut, with a node ball at either end attached to the strut by neodymium magnets. The control hardware consists of a Raspberry Pi Model B for high level control, an Arduino for low level control and sensing, and a custom motor driver board. The hardware design permits future expansions, such as motor-mounted cameras and automated strut attachment and detachment mechanisms.

$\delta_{step} = 0.138mm$. To attach to both tubes of a strut, each IPJR has two shaft collars, enabling the actuator to extend and contract the strut. All components are fixed to a frame consisting of laser cut, 1/4" acrylic sheets. Each IPJR was powered by a power supply that provided 5V and 12V output for the electronics and motors.

While each IPJR is fully capable of running my implementation of the MLE assembly algorithm as-is, I chose to run the algorithm from a central PC to avoid communication challenges. The Raspberry Pis instead run an HTTP server, which receives commands in the form of GET requests, and returns the data in response. Commands used in this experiment include checking the length potentiometer voltage, commanding the IPJR to move until it stops near a commanded location (without further control), and checking to see if the IPJR has finished moving. The control PC executes the experiment using a **Mathematica** implementation of the algorithms presented here, which controls and monitors the IPJRs through GET requests.

While I have previously demonstrated fully robotic external manipulation and on-board precise distance measurements [35], for this experiment, a human external manipulator is required to attach the IPJRs to struts and to place each new cell in a coarse configuration. Once placed, the IPJRs refine the cell by taking measurements of the distances between node balls. The MLE

Algorithm 18 The IPJR command algorithm is a simple feedback controller that attempts to adjust the node-node distance to a desired length plus a tolerance ϵ.

CommandIPJRToLength$(L, \Delta_{step}, \epsilon)$

1: $M_L \leftarrow$ node-node length measurement
2: **while** $\|M_L - L\| \geq \epsilon$ **do**
3: Command IPJR to move $\Delta_{step}(L - M_L)$ steps
4: $M_L \leftarrow$ node-node length measurement

algorithm requires direct measurements of the structure, but the IPJRs lacked that capability for these experiments. Instead, a ruler was used in order to obtain an unbiased measurement, not subject to bending due to the weight of the IPJR or play in the shaft collar attachment mechanism. The ruler was also used for verification of the physical experiments.

The setting of an IPJR's length is performed by a simple feedback controller, shown in Algorithm 18, in which the linear actuator adjusts its length based on the difference in measured length vs. desired length. When the measured length is within a specified tolerance of the desired length L, the shaft collars are fixed and the IPJR is removed. The range of the IPJR's linear actuator is $140mm$, so to handle a larger range of strut lengths, the IPJR should be attached to the strut such that the final length is within the range. This length is the initial length, which is measured as M_L. Then a loop occurs: Command IPJR to move $\delta_{step}(L - M_L)$ steps, take a new measurement M_L, until M_L is within a tolerance $\epsilon = 0.5mm$. While this controller will diverge if the δ_{step} is underestimated, in my experience the actuators converged almost always within 3 steps, and the simple controller was sufficient.

8.2 Simulation Experiment Results

For very small structures, such as the 10-node telescope, all assembly sequences can be enumerated. The 10-node telescope truss in Figure 8.2 has 12708 distinct assembly sequences. The traces of the assembly sequences are clumped into several distinct groups as shown in the top of Figure 8.2. The cause in distinction is due to nearly degenerate assembly steps contributing large errors, the same phenomenon was observed in the 84-node telescope.

I simulated MLE assembly on both the global optimal sequence and the median sequence

to observe what benefits MLE may provide to suboptimal sequences. The median is found in the lowest cluster. The results are shown in Figure 8.2. We ran 1000 trials of the MLE algorithm to generate the MLE covariance matrix and marginal ellipsoids shown in the bottom row.

The trace of the open loop optimal sequence is $2.84 \times 10^{-5} m^2$, and the open loop median sequence trace is $5.98 \times 10^{-5} m^2$, an increase of a factor of 2.1. However, as the middle row of Figure 8.2 shows, using MLE is much more effective on the median sequence. The traces of the optimal and median assembly sequences using MLE are $1.56 \times 10^{-5} m^2$ and $1.86 \times 10^{-5} m^2$, the median trace being only 1.19 times worse.

These results show that an optimal assembly sequence will perform better overall with the MLE estimator, but the distinction between the optimal and other sequences is not as severe as in the open loop cases. Therefore, algorithms that can find near-optimal sequences quickly, such as **FindLocallyOptimalSequence**, are sufficient for all but the most strict precision demands.

8.3 Physical Experiment Results

I chose to use MLE due to its favorable performance. To test the validity of the MLE algorithm on real hardware, I performed assembly trials of the optimal 10-node telescope sequence; 3 open loop trials, and 3 MLE trials. Assembly experiments were performed with the calibrated variances $\sigma_L^2 = 3.57 \times 10^{-7} m^2$ and $\sigma_M^2 = 6.25 \times 10^{-8} m^2$. I did not model physical phenomena such as gravity, strut deflection under the weight of the IPJRs, and free play in the fixed struts; I predicted that the MLE algorithm would be able to overcome the biases those phenomena would add to the system. Each trial took one or more hours to run, mostly due to issues attaching and detaching the IPJRs. Occasionally, the magnets on the struts would attract to each other instead of the node balls, requiring careful detachment. A few restarts were necessary.

To measure the final result of the structure after the experiment is completed, I used a modified version of the MLE filter with the following differences: the process probability terms are removed, and every pair of nodes with a distance of under $0.7m$ was measured three times. The extra measurements, including several edges never spanned by IPJRs, provided a more accurate

result. To estimate the error of the modified MLE filter, I simulated 1000 sets of measurements and compared the estimated result with the hidden structure and found this to be very accurate, a maximum mean squared error of just $0.25mm^2$ — these are the error bars in the individual trial plots in the middle row of Figure 8.3.

The results are shown in Figure 8.3. The physical trials followed the predicted patterns in simulation. On average, the open loop mean squared error is $4.33 \times 10^{-5}m^2$, while the MLE mean squared error is $1.09 \times 10^{-5}m^2$. The open loop trials averaged 1.6 times worse than predicted, while the MLE trials averaged 1.4 times **better** than predicted. Due to the small number of trials, and therefore the large error on the means, this is not surprising. The low error on node 7 in one of the open loop trials, for example, skewed the average to be better than the MLE average for that node. Given that node 7 depends on nodes 1 through 5, the almost-perfect positioning of node 7 in that single trial is coincidental.

The aforementioned physical biases were not an obstacle for MLE, as predicted, but the length offsets do show a small bias, which can be seen in the lower right of Figure 8.3. Most edges ended up slightly shorter on average; when all edges are considered, the mean error is $-0.13mm$. This may be due to a combination of free play in the joints of each strut, and the large mass of the $38.1mm$-diameter steel balls compressing the struts. I did expect to see a lengthening of struts due to the bowing caused by the weight of the IPJRs, and the subsequent restoration of linearity when the IPJRs were removed, but I did not observe this independently of the other biases. The variance of the edge length errors is $\sigma_L^2 = 3.54 \times 10^{-7}m^2$, almost equal to the previously estimated value of σ_L^2. The fact that MLE performed very well on the physical trials despite the length bias shows that, while modeling various forces would certainly improve the result, they are not **necessary** for precise assembly when MLE-SLAM is used.

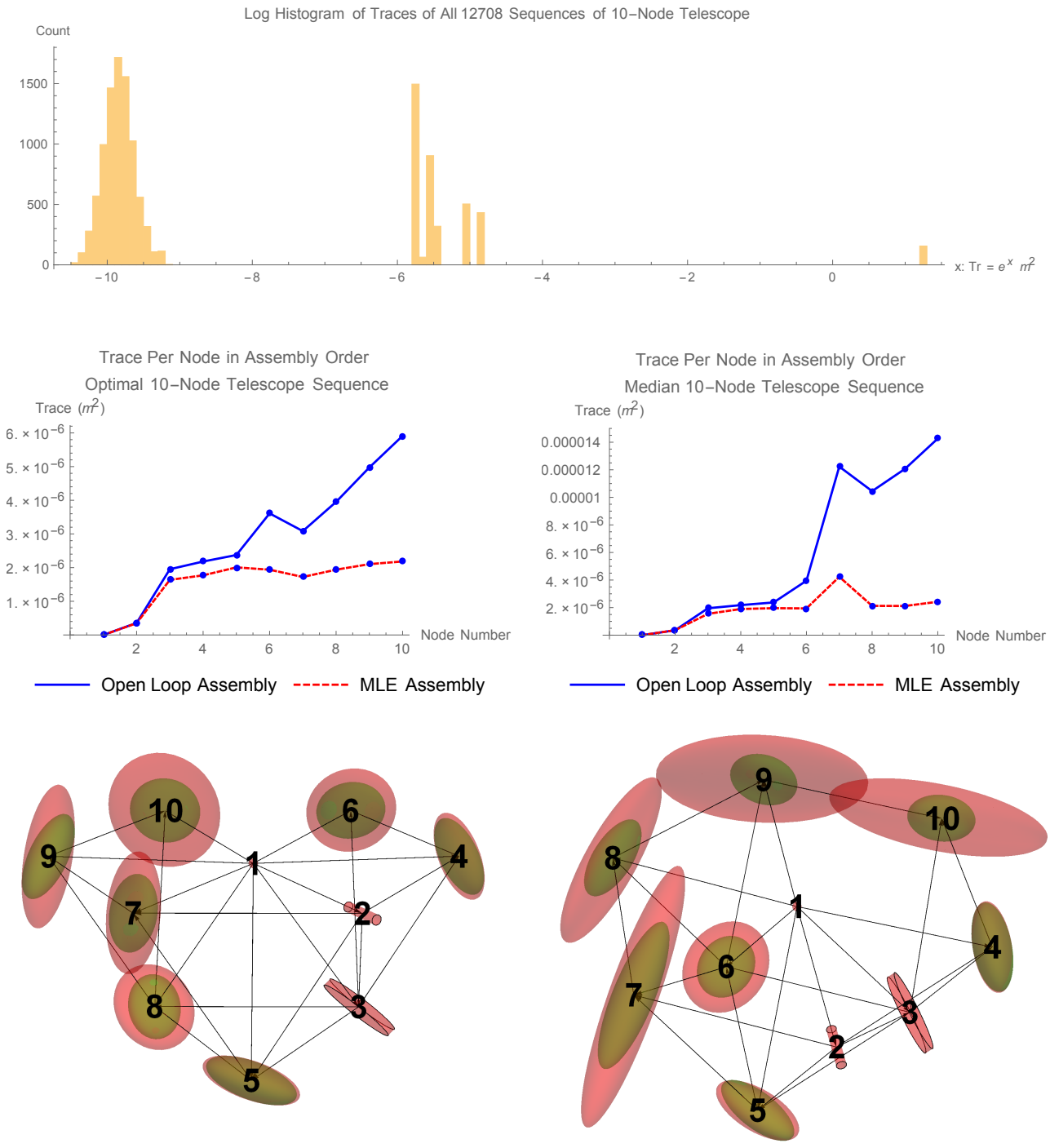


Figure 8.2: The10-node space telescope has 12708 assembly sequences, whose traces are distributed as shown in the top for $\sigma_L^2 = 3.57 \times 10^{-7} m^2$ and $\sigma_M^2 = 6.25 \times 10^{-8} m^2$. The optimal and median sequences, shown in the bottom row, are simulated with open loop and MLE. While the median trace is over 2 times worse, using MLE reduces this factor to just over 1. The ellipsoids are exaggerated to be visible.

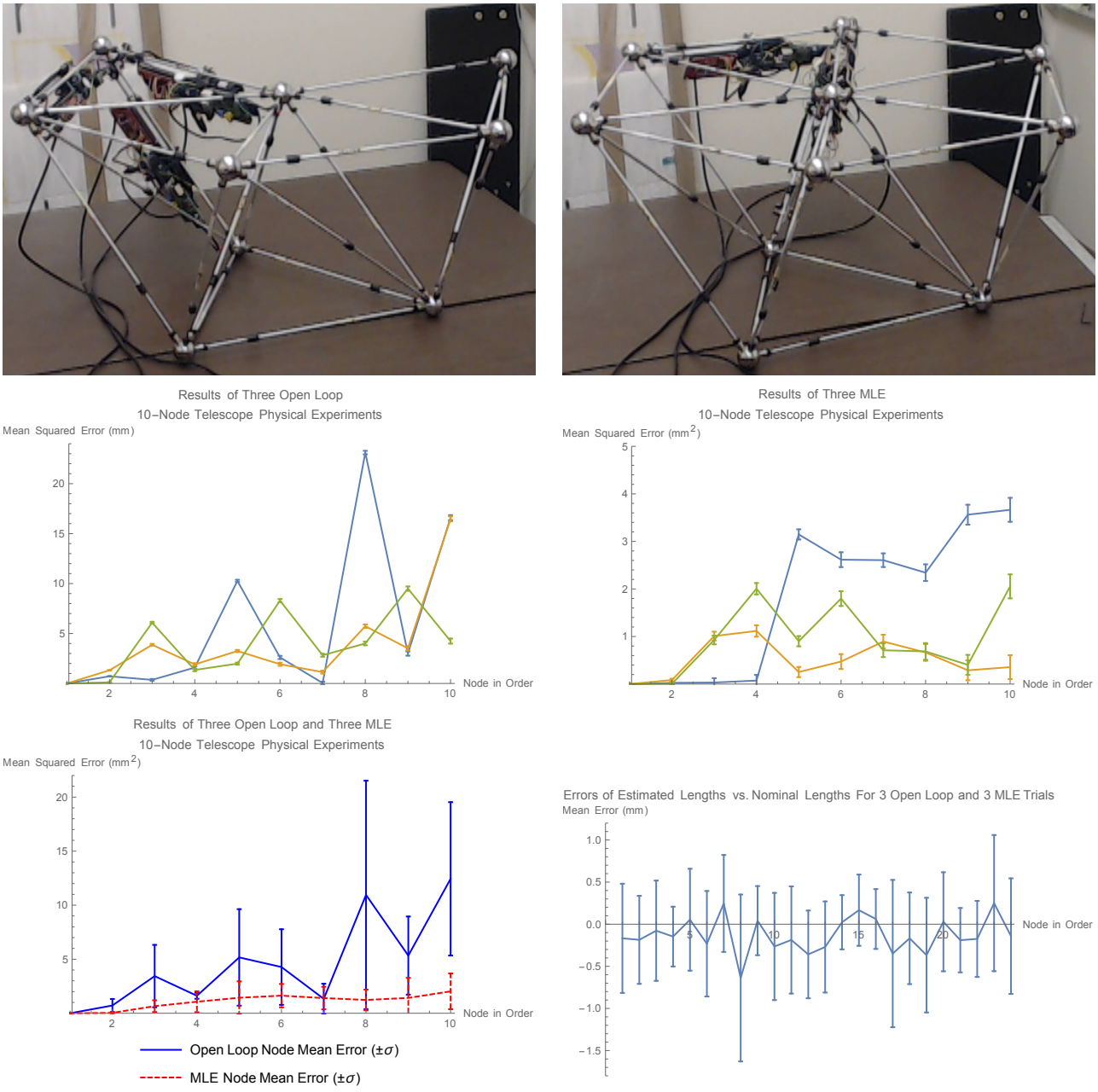


Figure 8.3: The results of 3 open loop and 3 MLE physical assembly trials are shown. Top row: the final two assembly steps. Middle row: node squared errors (mm^2) for the open loop trials (left) and the MLE trials (right). Bottom left: the means of the open loop and MLE cases plotted together. Bottom right: the mean errors of the estimated final strut lengths vs. the nominal lengths.

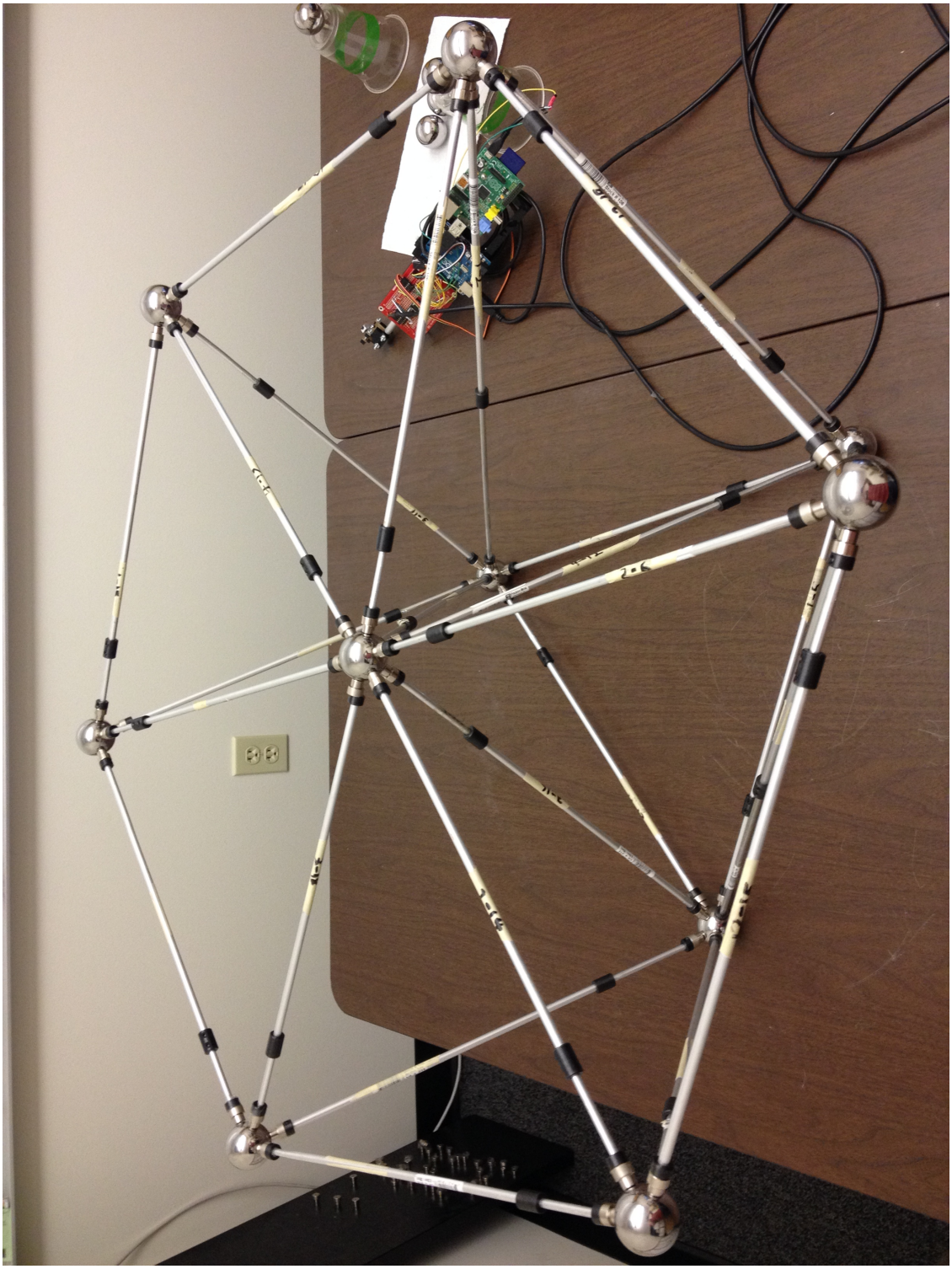


Figure 8.4: The 10-node telescope truss after assembly by IPJRs.

Chapter 9

Discussion

I have shown that a smart choice of the build sequence and error correction using SLAM can lead to assemblies with predictable accuracy using commodity parts. I validated this with simulations and physical trials, and showed that the SLAM algorithm could mitigate both the choice of a suboptimal sequence and the unmodeled, non-Gaussian physical processes that occur during physical trials. What remains to be seen, however, is whether this approach is suitable to achieve accuracies in the order of microns, such as required for telescope optical benches.

9.1 Case Study: Feasibility of Constructing 84-Node Truss with MLE Assembly

Without using MLE-SLAM, assembling the 84-node space telescope optical bench to the required precision would require actuators that are technically not feasible. To achieve a tolerance of $10\mu m$ on this particular structure requires that the worst case node have an average mean squared error of under $(0.00001m)^2 = 10^{-10}m^2$. The locally optimal sequence shown in Figure 9.1 requires that the process noise needs to be $\sigma_L^2 = 6.6 \times 10^{-13}m^2$. To put into context, the standard deviation of the length actuation must be, at most, **812 nanometers**. When processes such as thermal deformation under welding and sunlight are considered, even thermally favorable materials such as titanium, with its coefficient of thermal expansion of 8.6 microns per meter per Kelvin, cannot match that precision no matter which actuator is used[1] .

[1] Composite struts can be designed so that the coefficient of thermal expansion is 0 under certain conditions, but these would not be commodity materials.

Using MLE-based SLAM, however, drastically changes this picture as it mitigates build-up of error. This enables commodity sensors and actuators to reach the required precision, such as the actuator and laser distance sensor that I used in [35]. The Ultra-Motion linear actuator I incorporated into the IPJR prototype has an estimated process standard deviation of $\sigma_L = 2\mu m$ based on the observation that at least 95% of the time, any desired length is within, at most, $4\mu m$ from an actuator step length — two standard deviations. Using the same reasoning, the Keyence laser distance sensor is a measurement standard deviation of $\sigma_M = 0.5\mu m$. However, to be conservative in how I interpreted the precisions of the actuator and the laser sensor, and to allow room for the aforementioned physical processes, I inflated the noises to $\sigma_L = 8\mu m$ and $\sigma_M = 1\mu m$, and simulated MLE assembly trials on the sequence in Figure 9.1. It is obvious that each individual strut length can only be within $16\mu m$ 95% of the time, meaning that the tolerance of $10\mu m$ cannot be guaranteed with the hardware I used in [35]. That said, with MLE in the assembly process, the average trace is $3.13 \times 10^{-10} m^2$, making the mean error $17.7\mu m$ as shown in Figure 9.1, only slightly worse than the $16\mu m$ per strut error. More importantly, error grows very slowly, meaning that if I used a better actuator, I could realistically expect to match the overall precision requirement. In a realistic deployment, however, local sensing would be complemented by an absolute laser-based measurement system. Such systems, albeit expensive, are able to provide measurements in the order of micrometers over ranges of tens of meters, which would allow to provide lower bounds on assembly error across the structure.

9.2 Discussion

I showed that truss structures made of commodity, imprecise components can be assembled by Intelligent Precision Jigging Robots to a high degree of precision. I first derived a probability model and an error metric based on the covariance trace for open loop assembly. I showed that the linearization approximation of the node function $\mathcal{F}$ is valid on the order of $< 1mm$ for structures with strut lengths of $O(1m)$. Next, I described algorithms for identifying and traversing the space of assembly sequences, and enumerating the number of sequences. Then I presented a combined

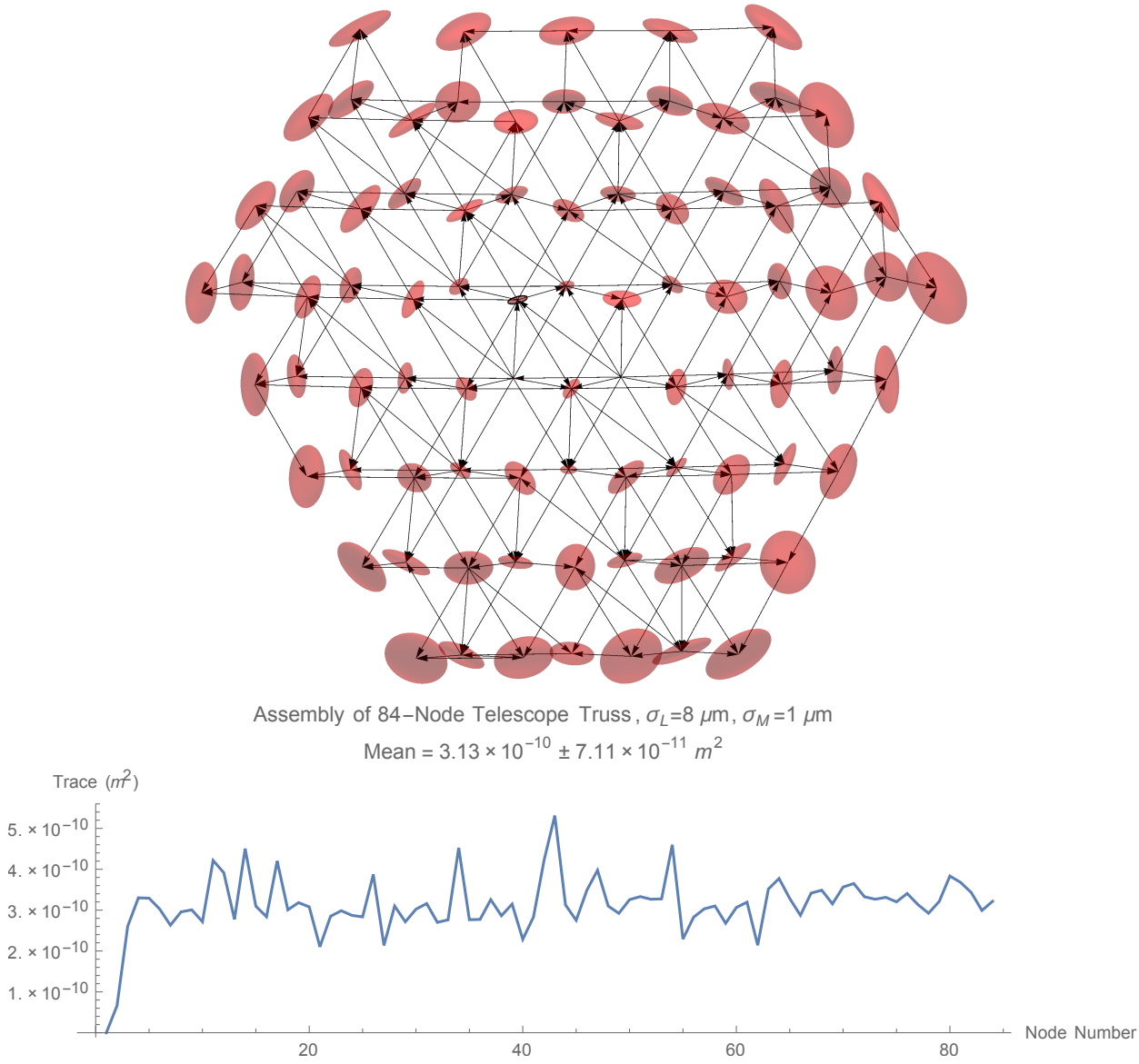


Figure 9.1: Top: A locally optimal assembly sequence for the 84-node telescope truss from [66]. Bottom: the results of 200 simulations of the full 84-node telescope truss as built by the MLE assembly algorithm using the actuators and sensors from [35] shows that the mean error does not visibly increase as node count increases, and that the mean node error is $17.7\mu m$, just over twice the standard deviation of a single strut.

algorithm for generating a near-optimal assembly sequence, which starts by choosing a central starting location, assembling a sequence by greedily choosing assembly steps with the minimum trace, and swapping steps of the resultant sequence until a minimum is found. I then described

an SLAM assembly algorithm, which builds a map of the structure and the IPJRs' positions while assembly is performed. I showed that Maximum Likelihood Estimation is preferable to a linear least squares method, the Extended Kalman Filter, and the Unscented Kalman Filter when process noise is large. Finally, I demonstrated that IPJRs can use the MLE assembly algorithm to produce precise truss structures made of steel node balls and aluminum telescoping tubes in spite of physical phenomena such as bending struts, which were not taken into account into the noise model.

An empirical method for estimating assembly sequence count has a number of benefits. For example, a method for estimating the extreme values of a metric (ie. nodal covariance) based on the distribution of a random sample of structures may be more accurate when the sequence count is known; this is future work. Additionally, empirical estimates may reveal a certain class of structure to have sub-exponential numbers of assembly sequences, which may lead to tractable algorithms to explore the full space of assembly sequences. While my truss sequence model is shown to be exponential, our model has very few constraints, and no geometrical constraints. The use of constraints, and the difference between empirical and theoretical estimates, may lead in turn to better theoretical estimates.

The exponential number of assembly sequences poses a challenge that is unlikely to be to overcome for larger structures. While starting from the center and performing a local search around a greedy sequence returned consistent results, the true global minimum trace may be significantly lower than any result found by the algorithm presented here. At first glance, given that trace is additive, one may think that finding a shortest path through the graph V, E using Dijkstra's algorithm would solve the problem quickly; but this approach does not work since a node's error depends on every strut leading up to it, and not just the previous node added. Likewise, an optimal configuration for a single node is not a requirement for the optimality of its descendants, ruling out a basic dynamic programming approach.

The $O(S(N)N^3)$ runtime of **FindLocallyOptimalSequence** may be polynomial, but still results in impractical runtimes. The algorithms presented here are not optimized; the first task is to reduce the number of times the covariance is calculated, taking advantage of shared assembly

steps between sequences and saved trace values. Another method, the implementation of which is future work, is to divide the structure into smaller substructures that are assembled independently of one another and combined later, in which each substructure is considered a single element to be constructed. Doing so would constrain the set of assembly sequences that can be identified, almost certainly removing the true optimum from the set, but the speed gains may be worth it.

IPJRs are simple: they only need to be able to grasp the nodes and to control their lengths with precision, a minimum of one degree of freedom if the graspers only have an on and off state. They should be less expensive and easier to make than robots with extra degrees of freedom. And given the parallelization definition presented in this paper, several groups consisting of three IPJRs can work in parallel on different branches of the structure. This also gives robustness to the assembly process; should any IPJR fail, the others will rearrange their groups to account for the reduced number. Only three are needed to assemble a structure. Since the precision is built in, the external manipulators and welders have a reduced need for precision of their own — only enough to grasp an IPJR, a strut, and weld — enabling them to be less expensive as well.

Chapter 10

Conclusion and Future Work

Numerous problems remain. Do better algorithms for generating optimal sequences exist? When real time measurements are too numerous for the MLE algorithm to function properly, are other SLAM algorithms developed for large-scale robotic problems more suitable? Does modeling the additional physical phenomena provide enough of a boost to precision to make it worthwhile? How does assembling several substructures independently and combining them at a later stage affect the overall precision? Answers to these questions are the subject of future research.

A dedicated on-orbit experiment to assemble a space telescope optical bench will require hardware that is even more precise than the sensors and actuators that I used in [35]. It will also make use of external measurement systems, such as precise cameras, to further improve the estimates. Yet, the value of the IPJR paradigm is that it can proceed in the absence of such precision measurements, e.g., when parts of the structure are occluded from the absolute position sensor or when individual IPJRs fail.

While I showed that SLAM can overcome unmodeled physical biases, the truss model can still benefit from including them. Consider that my physical trials showed a negative length bias; even with MLE-SLAM, every strut will be slightly shorter than intended even when SLAM is used. Instead of the error propagating, it will be reset at each node, but it will remain. Finite element analysis software has long been used to model physical processes, and my work in [46] demonstrates truss assembly with gravity under consideration.

I am not completely satisfied by the performance of **FindLocallyOptimalSequence**. While

a near-optimal sequence combined with the MLE assembly algorithm is likely to be nearly identical in performance to the global optimum combined with the MLE assembly algorithm, the fact that a global optimum exists and is not easily identifiable lends itself to numerous search algorithms. In addition to what I documented here, I tried simulated annealing; the results were poor, and I was unable to determine if it was due to poorly chosen temperature parameters or an inconvenient search space. More research is needed.

I also wanted to give the 3D IPJRs greater autonomy. I designed them to run on batteries. I wanted to install strut grippers, but could not due to excessive mass causing struts to detach. I installed and later removed cameras mounted on servos, and intended to use those to do monocular SLAM on the node balls. I designed parts that would make it easy for an external manipulator like the Baxter robot to move the IPJRs around. My original vision was a completely autonomous experiment, but time constraints and the fact that I worked alone on these robots hindered me.

The parallelization method described in this paper allows IPJRs to work in independent groups, each with an external manipulator to feed struts and to reposition the IPJRs, enabling swift and robust assembly. I therefore envision the IPJR paradigm being used not only to assemble the 84-node telescope, but much larger trusses, using materials and robots launched in multiple launches, not only in space but possibly also on earth.

I believe that the IPJR paradigm — and in particular the formulation as a SLAM problem to reduce the overall error — holds for a larger class of incrementally assembled structures. The IPJR concept is not limited to long struts and node balls; it can extend to robots that can precisely position any components that need to be cut, welded, and measured. The key contributions of the IPJR paradigm are that robotic structure assembly can be precise, can use commodity materials, can be robust to failure, and can use teams of robots with distributed resources. When commodity components are used instead of self-interlocking components, and are cut and shaped as needed, care must be taken to keep track of the structure error to enable structures to be built reliably. Minimizing the covariance trace is the natural metric for choosing assembly sequences, adding it to the list of other metrics found in the literature. Modeling robotic assembly as a SLAM problem

with added environmental manipulation has the potential to enable the construction of very large structures by teams of robots in open and uncertain environments, far from humans that can take over at a moment's notice.

Long-term visions of robotic construction will require concepts from a large intersection of disciplines to become reality. The most important of these is the ability for the robots to reason about their environment; to learn from sensory clues and to make adjustments as the need arises. The IPJR paradigm addresses a small aspect of this problem: how to adjust assembly plans based on sensed errors. Whatever form the algorithms and robots take, I am certain that large-scale robotic construction projects of the future will use some form of SLAM and some form of sequence optimization, and the IPJR paradigm is the first step.

Bibliography

[1] Vicon homepage. `http://www.vicon.com`. Accessed: 2014 October 22.

[2] S. R. Barros dos Santos, S. N. Givigi, and C. L. Nascimento. Autonomous construction of structures in a dynamic environment using reinforcement learning. In Systems Conference (SysCon), 2013 IEEE International, pages 452–459. IEEE, 2013.

[3] S. Berman, A. Halasz, M. A. Hsieh, and V. Kumar. Optimized stochastic policies for task allocation in swarms of robots. IEEE Transactions on Robotics, 25(4):927–937, 2009.

[4] J. L. Blanco, J. A. Fernandez-Madrigal, and J. González. Efficient probabilistic range-only slam. In Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, pages 1017–1022. IEEE, 2008.

[5] A. Bolger, M. Faulkner, D. Stein, L. White, S. Yun, and D. Rus. Experiments in decentralized robot construction with tool delivery and assembly robots. In Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, pages 5085–5092. IEEE, 2010.

[6] M. Bonani, S. Magnenat, P. Rétornaz, and F. Mondada. The hand-bot, a robot design for simultaneous climbing and manipulation. In Intelligent Robotics and Applications, pages 11–22. Springer, 2009.

[7] T. Bretl, S. Rock, J. C. Latombe, B. Kennedy, and H. Aghazarian. Free-climbing with a multi-use robot. In Experimental Robotics IX, pages 449–458. Springer, 2006.

[8] H. G. Bush, C. L. Herstrom, W. L. Heard, T. J. Collins, and W. B. Fichter. Design and fabrication of an erectable truss for precision segmentedreflector application. Journal of Spacecraft and Rockets, 28(2):251–257, 1991.

[9] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. SIAM Journal on Computing, 14(1):210–223, 1985.

[10] B. Chu, K. Jung, C. S. Han, and D. Hong. A survey of climbing robots: Locomotion and adhesion. International journal of precision engineering and manufacturing, 11(4):633–647, 2010.

[11] J. L. Crassidis and J. L. Junkins. Optimal estimation of dynamic systems. CRC press, 2011.

[12] T. DeFazio and D. Whitney. Simplified generation of all mechanical assembly sequences. IEEE Journal of Robotics and Automation, RA-3(6):640–658, 1987.

[13] C. Detweiler, M. Vona, Y. Yoon, S. Yun, and D. Rus. Self-assembling mobile linkages. Robotics & Automation Magazine, IEEE, 14(4):45, 2007.

[14] M. Dogar, R. A. Knepper, A. Spielberg, C. Choi, H. I. Christensen, and D. Rus. Towards coordinated precision assembly with robot teams. In Proceedings of the 2014 International Symposium on Experimental Robotics, page TBD, 2014.

[15] W. Doggett. Robotic assembly of truss structures for space systems and future research plans. In Aerospace Conference Proceedings, 2002. IEEE, volume 7, pages 3587–3598. IEEE, 2002.

[16] J. T. Dorsey, W. Doggett, **E. Komendera**, N. Correll, R. Hafley, and B. D. King. An efficient and versatile means for assembling and manufacturing systems in space. In Proceedings of the AIAA SPACE Conference, pages 1–19, 2012.

[17] J. T. Dorsey, T. C. Jones, W. R. Doggett, J. S. Brady, F. C. Berry, G. G. Ganoe, E. J. Anderson, B. D. King, and C. D. Mercer. Recent developments in the design, capabilities and autonomous operations of a lightweight surface manipulation system and test bed. In Proceedings of the AIAA Space Conference, 2011.

[18] D. Ebbets, J. DeCino, and J. Green. Architecture concept for a 10m uv-optical space telescope. Proceedings of SPIE, 6265(62651S-1), 2006.

[19] B. R. Fox and K. G. Kempf. Opportunisitic scheduling for robotic assembly. In Proceedings of the International Conference on Robotics and Automation, pages 880–889, 1985.

[20] S. Gottschlich, C. Ramos, and D. Lyons. Assembly and task planning: A taxonomy. IEEE Robotics and Automation Magazine, 1(3):4–12, 1994.

[21] B. Hamner, S. Koterba, J. Shi, R. Simmons, and S. Singh. An autonomous mobile manipulator for assembly tasks. Autonomous Robots, 28(1):131–149, 2010.

[22] F. W. Heger. Assembly Planning in Constrained Environments: Building Structures with Multiple Mobile Robots. PhD thesis, Carnegie Mellon University, 2010.

[23] L. S. Homem de Mello and S. Lee. Computer Aided Mechanical Assembly Planning. Springer, 1991.

[24] L. S. Homem de Mello and A. C. Sanderson. Correct and complete algorithm for the generation of mechanical assembly sequences. IEEE Transactions on Robotics and Automation, 7(2):228–240, 1991.

[25] R. A. Horn and C. R. Johnson. Matrix analysis. Cambridge university press, 2012.

[26] R. P. Hoyt, J. I. Cushing, G. J. Jimmerson, J. St. Luise, and J. T. Slostad. Trusselator: On-orbit fabrication of high-performance composite truss structures. In Proceedings of the 2014 AIAA SPACE Conference, page TBD, 2014.

[27] A. E. Jimenez-Cano, J. Martin, G. Heredia, A. Ollero, and R. Cano. Control of an aerial robot with multi-link arm for assembly tasks. In Robotics and Automation (ICRA), 2013 IEEE International Conference on, pages 4916–4921. IEEE, 2013.

[28] S. J. Julier and J. K. Uhlmann. A new extension of the kalman filter to nonlinear systems. In Int. symp. aerospace/defense sensing, simul. and controls, volume 3, pages 3–2. Orlando, FL, 1997.

[29] G. Kantor and S. Singh. Preliminary results in range-only localization and mapping. In Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on, volume 2, pages 1818–1823. IEEE, 2002.

[30] R. A. Knepper, T. Layton, J. Romanishin, and D. Rus. Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In Robotics and Automation (ICRA), 2013 IEEE International Conference on, pages 855–862. IEEE, 2013.

[31] D. E. Knuth. Estimating the efficiency of backtrack programs. Mathematics of computation, 29(129):122–136, 1975.

[32] D. Koller and N. Friedman. Probabilistic graphical models: principles and techniques. MIT press, 2009.

[33] **E. Komendera** and N. Correll. Precise assembly of 3d truss structures using ekf-based error prediction and correction. In Proceedings of the 2014 International Symposium on Experimental Robotics, page TBD, 2014.

[34] **E. Komendera** and N. Correll. Precise assembly of 3d truss structures using mle-based error prediction and correction. International Journal of Robotics Research, Under Review, 2015.

[35] **E. Komendera**, J. T. Dorsey, W. R. Doggett, and N. Correll. Truss assembly and welding by intelligent precision jigging robots. In Proceedings of the 6th Annual IEEE Int. Conf. on Technologies for Practical Robot Applications (TEPRA), page TBD, 2014.

[36] **E. Komendera**, D. Reishus, J. T. Dorsey, W. R. Doggett, and N. Correll. Precise truss assembly using commodity parts and low precision welding. In Proceedings of the Fifth Annual IEEE International Conference on Technologies for Practical Robot Applications, 2013.

[37] **E. Komendera**, Dustin Reishus, John T. Dorsey, William R. Doggett, and Nikolaus Correll. Precise truss assembly using commodity parts and low precision welding. Intelligent Service Robotics, 7(2):93–102, 2014.

[38] M. S. Lake. Launching a 25-meter space telescope, are astronauts a key to the next technically logical step after ngst? In IEEE Aerospace Conference, 2001.

[39] M. S Lake, L. D. Peterson, M. M. Mikulas, J. D. Hinkle, L. R. Hardaway, and J. Heald. Structural concepts and mechanics issues for ultra-large optical systems. In 1999 Ultra Lightweight Space Optics Workshop, 1999.

[40] B. Lazzerini and F. Marcelloni. A genetic algorithm for generating optimal assembly plans. Artificial Intelligence in Engineering, 14:319–329, 2000.

[41] S. Lee and H. Moradi. Disassembly sequencing and assembly sequence verification using flow networks. In Proceedings of the International Conference on Robotics and Automation, 1999.

[42] Q. Lindsey and V. Kumar. Distributed construction of truss structures. In Algorithmic Foundations of Robotics X, pages 209–225. Springer, 2013.

[43] P. C. Mahalanobis. On the generalized distance in statistics. Proceedings of the National Institute of Sciences (Calcutta), 2:49–55, 1936.

[44] M. J. Mahoney and A. C. Ibbott. A large deployable reflector assembly scenario, a space station utilization study. Technical Report NASA JPL D-5942, 1988.

[45] J. H. Mathews and K. D. Fink. Numerical methods using MATLAB. Pearson, 2004.

[46] M. A. McEvoy, **E. Komendera**, and N. Correll. Assembly path planning for stable robotic construction. In Proceedings of the Sixth Annual IEEE International Conference on Technologies for Practical Robot Applications, page TBD, 2014.

[47] M. M. Mikulas, T. J. Collins, and M. F. Card. Structural stiffness, strength and dynamic characteristics of large tetrahedral space truss structures. NASA TM X-74001, 1977.

[48] M. M. Mikulas, T. J. Collins, and J. M. Hedgepeth. Preliminary design approach for large high precision segmented reflectors. NASA TM 102605, 1990.

[49] M. M. Mikulas and J. T. Dorsey. An integrated in-space construction facility for the 21st century. NASA TM 101515, 1988.

[50] M. Montemerlo and S. Thrun. Fastslam 2.0. FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics, pages 63–90, 2007.

[51] N. Napp and R. Nagpal. Distributed amorphous ramp construction in unstructured environments. Robotica, 32(02):279–290, 2014.

[52] W. R. Oegerle, L. R. Purves, J. G. Budinoff, R. V. Moe, T. M. Carnahan, D. C. Evans, and C. K. Kim. Concept for a large scalable space telescope: In-space assembly. Proceedings of SPIE, 6265(62652C), 2006.

[53] F. Röhrdanz, H. Mosemann, and F. M. Wahl. High lap: A high level system for generating, representing and evaluating assembly sequences. In In Proceedings of the International Joint Symposia on Intelligence and Systems, pages 134–141, 1996.

[54] R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith. First results in the coordination of heterogeneous robots for large-scale assembly. In Experimental Robotics VII, pages 323–332. Springer, 2001.

[55] S. S. Skiena. The Algorithm Design Manual. Springer, 2008.

[56] R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. The international journal of Robotics Research, 5(4):56–68, 1986.

[57] J. Sondow. Problem 11132. The American Mathematical Monthly, 112(2):180, 2005.

[58] A. Stroupe, T. Huntsberger, B. Kennedy, H. Aghazarian, .E Baumgartner, A. Ganino, M. Garrett, A. Okon, M. Robinson, and J. Townsend. Heterogeneous robotic systems for assembly and servicing. In 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space, ESA. ESA, volume 603, page 82, 2005.

[59] A. Stroupe, T. Huntsberger, A. Okon, and H. Aghazarian. Precision manipulation with cooperative robots. pages 235–248. Springer, 2005.

[60] S. Sundaram, I. Remmler, and N. M. Amato. Disassembly sequencing using a motion planning appproach. In Proceedings of the International Conference on Robotics and Automation, 2001.

[61] S. Thrun, W. Burgard, and D. Fox. Probabilistic robotics. MIT press, 2005.

[62] S. Thrun and M. Montemerlo. The GraphSLAM algorithm with applications to large-scale mapping of urban structures. International Journal on Robotics Research, 25(5/6):403–430, 2005.

[63] R. Van Der Merwe, A. Doucet, N. De Freitas, and E. Wan. The unscented particle filter. In NIPS, pages 584–590, 2000.

[64] M. Vona, C. Detweiler, and D. Rus. Shady: Robust truss climbing with mechanical compliances. In Experimental Robotics, pages 431–440. Springer, 2008.

[65] E. A. Wan and R. Van Der Merwe. The unscented kalman filter for nonlinear estimation. In Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000, pages 153–158. IEEE, 2000.

[66] J. J. Watson, T. J. Collins, and H. G. Bush. A history of astronaut construction of large space structures at nasa langley research center. In Aerospace Conference Proceedings, 2002. IEEE, volume 7, pages 7–3569. IEEE, 2002.

[67] J. Wawerla, G. S. Sukhatme, and M. J. Mataric. Collective construction with multiple robots. In Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on, volume 3, pages 2696–2701. IEEE, 2002.

[68] J. Werfel, K. Petersen, and R. Nagpal. Designing collective behavior in a termite-inspired robot construction team. Science, 343(6172):754–758, 2014.

[69] R. H. Wilson. On Geometric Assembly Planning. PhD thesis, Stanford University, 1992.

[70] J. Wolter. On the automatic generation of assembly plans. In Proceedings of the International Conference on Robotics and Automation, pages 62–68, 1989.

[71] J. Worcester, R. Lakaemper, and M. A. Hsieh. 3-dimensional tiling for distributed assembly by robot teams. In Proceedings of the 13th International Symposium on Experimental Robotics (ISER2012), pages 143–154. Springer, 2012.

[72] S. Yun. Coordinating construction by a distributed multi-robot system. PhD thesis, Massachusetts Institute of Technology, 2010.

[73] S. Yun, D. A. Hjelle, E. Schweikardt, H. Lipson, and D. Rus. Planning the reconfiguration of grounded truss structures with truss climbing robots that carry truss elements. In Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, pages 1327–1333. IEEE, 2009.

[74] S. Yun, M. Schwager, and D. Rus. Coordinating construction of truss structures using distributed equal-mass partitioning. In Robotics Research, pages 607–623. Springer, 2011.

[75] D. Zimpfer, P. Kachmar, and S. Tuohy. Autonomous rendezvous, capture and in-space assembly: past, present and future. In 1st Space Exploration Conference: Continuing the Voyage of Discovery, volume 1, pages 234–245, 2005.

Appendix A

Derivation of Node Placement Functions

This appendix presents the full derivation of the node functions $\mathcal{F}$ and their derivatives with respect to node positions and lengths.

A.1 Node Placement Functions

X_f is a found as a function of the base triple it is attached to, and the struts connecting the float to the bases, of the following form:

$$X_f = \mathcal{F}_f(X_{B_f}, L_{B_f}) \tag{A.1}$$

A.1.1 Forming a Triangle from a Strut Base

In the truss assembly model, a triangle is formed on the XY-plane. Finding the position of a node given two bases and two lengths is equivalent to solving the following system of equations for X_f with base nodes i and j and floating node f:

$$\begin{aligned} L_{i,f} &= ||X_i - X_f|| \\ L_{j,f} &= ||X_j - X_f|| \end{aligned} \tag{A.2}$$

To solve this cleanly while minimizing the number of terms, the following derivation begins by calculating the float on the XY-plane given nodes at the origin and the x-axis, then transforming

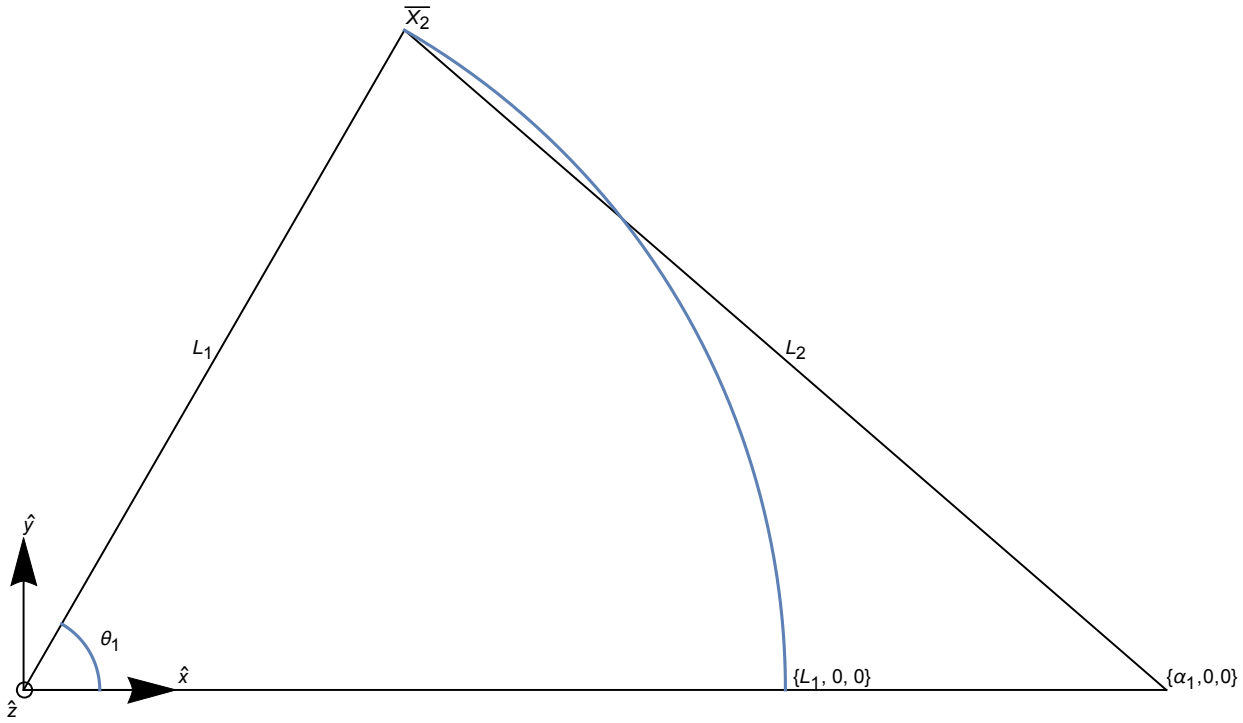


Figure A.1: Finding the position of $\bar{X_2}$ requires rotation of an L_1-length vector until it has a length of L_2 from the point $\alpha_1, 0, 0$.

the result to the truss reference frame. A visual reference for the untransformed triangle is shown in Figure A.1, and the full result is shown in Figure A.2.

Begin by defining a vector $\{L_1 > 0, 0, 0\}$ and an angle θ_1. Let $\bar{X_2}$ be the vector resulting by rotating $\{L_1, 0, 0\}$ by θ_1 around $\{0, 0, 1\}$. This represents a vector whose distance from the origin is L_1 The result is:

$$\bar{X_2} = \begin{pmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} L_1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} L_1 \cos(\theta_1) \\ L_1 \sin(\theta_1) \\ 0 \end{pmatrix} \tag{A.3}$$

Let $\{\alpha_1 > 0, 0, 0\}^T$ be a vector that has a distance of L_2 from $\bar{X_2}$. Determining $\bar{X_2}$ as a function only of L_1, L_2, and α_1 requires solving for θ_1:

$$\left\| \begin{pmatrix} L_1 \cos(\theta_1) \\ L_1 \sin(\theta_1) \\ 0 \end{pmatrix} - \begin{pmatrix} \alpha_1 \\ 0 \\ 0 \end{pmatrix} \right\| = \sqrt{(L_1 \cos(\theta_1) - \alpha_1)^2 + L_1^2 \sin^2(\theta_1)} = L_2 \tag{A.4}$$

Solving for θ_1 and using the identity $A^2 = A^2 \cos\theta + A^2 \sin\theta$, there are two values for θ_1, corresponding to the two possible positions for $\bar{X_2}$, each one a reflection of the other over $\{\alpha_1, 0, 0\}^T$:

$$\theta_1 = \pm \cos^{-1}\left(\frac{\alpha_1^2 + L_1^2 - L_2^2}{2\alpha_1 L_1}\right) \tag{A.5}$$

$$\tag{A.6}$$

The positive solution corresponds to $\bar{X_2}$ having a positive Y-value. Inserting this value of θ_1 into Equation A.3 gives:

$$\bar{X_2} = \begin{pmatrix} \frac{L_1^2 - L_2^2 + \alpha_1^2}{2\alpha_1} \\ L_1 \sqrt{1 - \frac{(L_1^2 - L_2^2 + \alpha_1^2)^2}{4 L_1^2 \alpha_1^2}} \\ 0 \end{pmatrix} \tag{A.7}$$

Equation A.7 gives the solution to the triangle apex when the base nodes are located at the origin and on the positive X-axis. The triangle inequality must be observed:

$$\alpha_1 \leq L_1 + L_2 \tag{A.8}$$

Failure to keep this inequality will result in a complex value for $\bar{X_2}$.

For the general case, where the base nodes are arbitrary points on the XY-plane, substitutions must be made. The following identities are used for the base nodes and float node:

$$
\begin{aligned}
X_i &= \{x_i, y_i, 0\}^T \\
X_j &= \{x_j, y_j, 0\}^T \\
X_f &= \{x_f, y_f, 0\}^T \\
X_{j,i} &= X_j - X_i \\
X_{f,i} &= X_f - X_i \\
X_{f,j} &= X_f - X_j \\
||X_{j,i}|| &= \sqrt{X_{j,i}.X_{j,i}} = \alpha_1 \\
L_{i,f} &= L_1 \\
L_{j,f} &= L_2
\end{aligned} \tag{A.9}
$$

The untransformed location of $\bar{X_2}$ found in Equation A.7 becomes:

$$
\bar{X_2} = \begin{pmatrix} \frac{L_1^2 - L_2^2 + X_{j,i}.X_{j,i}}{2\sqrt{X_{j,i}.X_{j,i}}} \\ L_1\sqrt{1 - \frac{\left(L_1^2 - L_2^2 + X_{j,i}.X_{j,i}\right)^2}{4X_{j,i}.X_{j,i}L_1^2}} \\ 0 \end{pmatrix} \tag{A.10}
$$

To transform this to the correct frame of reference — that is, to find X_f — this needs to be rotated such that the X-axis in the untransformed space becomes parallel to $X_{j,i}$, then translated such that the origin of the untransformed space becomes X_i. Using the fact that the column vectors of any rotation matrix are the new basis vectors, the transformed X-axis is simply the normalized vector between the base node j and base node i, $\frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}$. Since the new triangle remains in the XY-plane, the transformed Z-axis is unchanged, and is $\hat{z}$. Finally, the transformed Y-axis is the cross product of the transformed Z and X axes. Thus, let $R_{i,j,f}$ be the rotation matrix defined by:

$$
R_{i,j,f} = \left(\frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}, \hat{z} \times \frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}, \hat{z} \right) \tag{A.11}
$$

Then, $\mathcal{F}_f$ can be found:

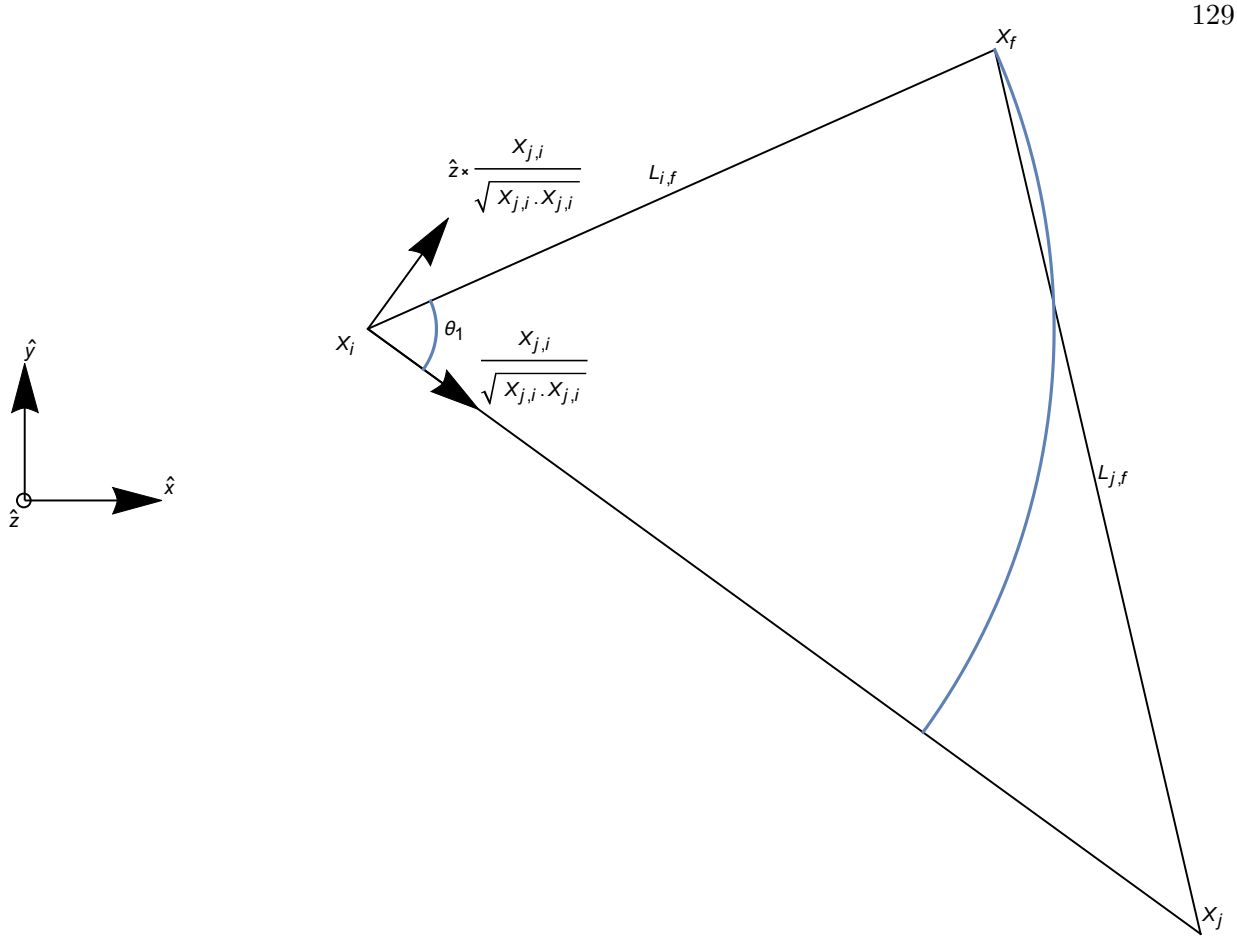


Figure A.2: The position of X_f as a function of $X_i, X_j, L_{i,f}, L_{j,f}$ for a triangle example.

$$\begin{aligned}
\mathcal{F}_f(X_i, X_j, L_{i,f}, L_{j,f}) &= X_i + R_{i,j,f}.\bar{X}_2 \\
&= X_i + \frac{X_{j,i}\left(L_{i,f}^2 - L_{j,f}^2 + X_{j,i}.X_{j,i}\right)}{2X_{j,i}.X_{j,i}} \\
&\quad + \frac{L_{i,f}\sqrt{1 - \frac{\left(L_{i,f}^2 - L_{j,f}^2 + X_{j,i}.X_{j,i}\right)^2}{4L_{i,f}^2 X_{j,i}.X_{j,i}}}\,(\hat{z} \times X_{j,i})}{\sqrt{X_{j,i}.X_{j,i}}}
\end{aligned} \tag{A.12}$$

Due to choosing the positive value for θ_1 in Equation A.6, the cross product of $X_{j,i}$ and $X_{f,i}$ will be $\hat{z}$. Swapping the nodes i and j in the function has the same effect as choosing the negative value for θ_1, and the result will be the reflection over $X_{j,i}$ of X_f.

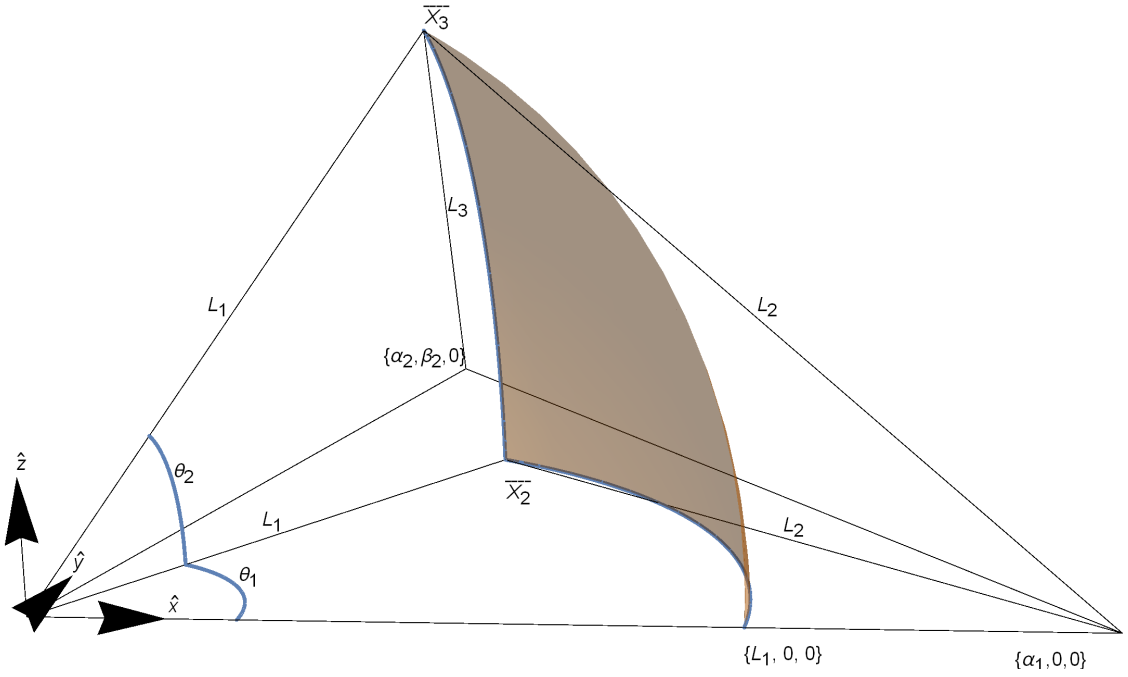


Figure A.3: Finding the position of $\bar{X}_3$ requires rotation of an L_1-length vector until it has a length of L_2 from the point $\alpha_1, 0, 0$, forming the intermediate point $\bar{X}_2$, then rotating it around the X-axis until it has a length of L_3 from the point $\alpha_2, \beta_2, 0$.

A.1.2 Forming a Tetrahedron from a Triangle Base

A similar method can be used to the node position X_f when there are three bases; however, the resulting function is no longer confined to a plane, and an extra transformation must take place, resulting in a formula with a much larger number of terms. A visual reference for the untransformed tetrahedron is shown in Figure A.3, and the full result is shown in Figure A.4. As before, I define the position X_f as the solution to the positions of the nodes i, j, k and the lengths L_i, L_j, L_k:

$$\begin{aligned} L_{i,f} &= ||X_i - X_f|| \\ L_{j,f} &= ||X_j - X_f|| \\ L_{k,f} &= ||X_k - X_f|| \end{aligned} \tag{A.13}$$

The solution to this problem follows the same steps leading up to the definition of $\bar{X}_2$ in

Equation A.7, with the two edges L_1, L_2 corresponding to $L_{i,j}, L_{j,f}$. However, an additional step takes place: the resultant vector $\bar{X_2}$ must be rotated around the X-axis such that it has a distance of $L_3 = L_{k,f}$ from a vector $\alpha_2, \beta_2 > 0, 0$, which corresponds to the relative position of X_k to X_i and X_j. Let $\bar{X_3}$ be that vector, and θ_2 be the rotation angle about the X-axis:

$$\begin{aligned}
\bar{X_3} &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_2) & -\sin(\theta_2) \\ 0 & \sin(\theta_2) & \cos(\theta_2) \end{pmatrix} . \bar{X_2} \\
&= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_2) & -\sin(\theta_2) \\ 0 & \sin(\theta_2) & \cos(\theta_2) \end{pmatrix} \cdot \begin{pmatrix} \frac{\alpha_1^2 + L_1^2 - L_2^2}{2\alpha_1} \\ L_1 \sqrt{1 - \frac{\left(\alpha_1^2 + L_1^2 - L_2^2\right)^2}{4\alpha_1^2 L_1^2}} \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} \frac{L_1^2 - L_2^2 + \alpha_1^2}{2\alpha_1} \\ \cos(\theta_2)\, L_1 \sqrt{1 - \frac{\left(L_1^2 - L_2^2 + \alpha_1^2\right)^2}{4L_1^2\alpha_1^2}} \\ \sin(\theta_2)\, L_1 \sqrt{1 - \frac{\left(L_1^2 - L_2^2 + \alpha_1^2\right)^2}{4L_1^2\alpha_1^2}} \end{pmatrix}
\end{aligned} \tag{A.14}$$

L_3 is the norm of $\bar{X_3}$ and $\alpha_2, \beta_2 > 0, 0$:

$$L_3 = \left\| \begin{pmatrix} \frac{L_1^2 - L_2^2 + \alpha_1^2}{2\alpha_1} \\ \cos(\theta_2)\, L_1 \sqrt{1 - \frac{\left(L_1^2 - L_2^2 + \alpha_1^2\right)^2}{4L_1^2\alpha_1^2}} \\ \sin(\theta_2)\, L_1 \sqrt{1 - \frac{\left(L_1^2 - L_2^2 + \alpha_1^2\right)^2}{4L_1^2\alpha_1^2}} \end{pmatrix} - \begin{pmatrix} \alpha_2 \\ \beta_2 \\ 0 \end{pmatrix} \right\| \tag{A.15}$$

To simplify the representation of the solution to θ_2, let:

$$\begin{aligned}
t_1 &= \frac{\alpha_1^2 + L_1^2 - L_2^2}{2\alpha_1} \\
t_2 &= L_1 \sqrt{1 - \frac{\left(\alpha_1^2 + L_1^2 - L_2^2\right)^2}{4\alpha_1^2 L_1^2}}
\end{aligned} \tag{A.16}$$

Then:

$$
\begin{aligned}
L_3 &= \left\| \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_2) & -\sin(\theta_2) \\ 0 & \sin(\theta_2) & \cos(\theta_2) \end{pmatrix} \cdot \begin{pmatrix} t_1 \\ t_2 \\ 0 \end{pmatrix} - \begin{pmatrix} \alpha_2 \\ \beta_2 \\ 0 \end{pmatrix} \right\| \\
&= \sqrt{(t_1 - \alpha_2)^2 + (t_2 \cos(\theta_2) - \beta_2)^2 + t_2^2 \sin^2(\theta_2)}
\end{aligned} \tag{A.17}
$$

Solving for θ_2, again using the identity $A^2 = A^2\cos\theta + A^2\sin\theta$, gives the following:

$$
\theta_2 = \pm\cos^{-1}\left(\frac{\alpha_2^2 + \beta_2^2 - L_3^2 - 2\alpha_2 t_1 + t_1^2 + t_2^2}{2\beta_2 t_2}\right) \tag{A.18}
$$

$$
= \pm\cos^{-1}\left(\frac{\alpha_1\beta_2^2 + \alpha_1\alpha_2^2 - \alpha_1^2\alpha_2 + (\alpha_1 - \alpha_2)L_1^2 - \alpha_1 L_3^2 + \alpha_2 L_2^2}{\alpha_1\beta_2 L_1\sqrt{-\frac{-2L_1^2(\alpha_1^2+L_2^2)+(L_2^2-\alpha_1^2)^2+L_1^4}{\alpha_1^2 L_1^2}}}\right) \tag{A.19}
$$

Inserting the positive result in Equation A.19 into Equation A.14 gives:

$$
\bar{X_3} = \begin{pmatrix} \frac{L_1^2 - L_2^2 + \alpha_1^2}{2\alpha_1} \\ \frac{(\alpha_1-\alpha_2)L_1^2 + \alpha_1\alpha_2^2 + \alpha_1\beta_2^2 - L_3^2\alpha_1 + L_2^2\alpha_2 - \alpha_1^2\alpha_2}{2\alpha_1\beta_2} \\ L_1\sqrt{1 - \frac{(L_1^2 - L_2^2 + \alpha_1^2)^2}{4L_1^2\alpha_1^2}}\sqrt{\frac{((\alpha_1-\alpha_2)L_1^2 + \alpha_1\alpha_2^2 + \alpha_1\beta_2^2 - L_3^2\alpha_1 + L_2^2\alpha_2 - \alpha_1^2\alpha_2)^2}{(L_1^4 - 2(L_2^2+\alpha_1^2)L_1^2 + (L_2^2-\alpha_1^2)^2)\beta_2^2} + 1} \end{pmatrix} \tag{A.20}
$$

Equation A.20 gives the solution to the tetrahedron apex when the base nodes are the origin, on the X-axis, and on the XY-plane. The result has a positive z-axis value. The principle behind the triangle inequality holds for tetrahedra as well, with the requirements for L_1 and L_2 the same, and the added requirement that the distance between $\bar{X_2}$ and $\{\alpha_2, \beta_2, 0\}^T$ has to be at most L_3:

$$
\begin{aligned}
\alpha_1 &\leq L_1 + L_2 \\
\left\|\bar{X_2} - \{\alpha_2, \beta_2, 0\}^T\right\| &\leq L_3
\end{aligned} \tag{A.21}
$$

As with the triangle case, $\bar{X_3}$ must be transformed such that it gives the correct result for any arbitrary X_i, X_j, X_k. The following identities are used for the base nodes and float node:

$$
\begin{aligned}
X_i &= \{x_i, y_i, z_i\}^T \\
X_j &= \{x_j, y_j, z_j\}^T \\
X_k &= \{x_k, y_k, z_k\}^T \\
X_f &= \{x_f, y_f, z_f\}^T \\
X_{j,i} &= X_j - X_i \\
X_{k,i} &= X_k - X_i \\
X_{f,i} &= X_f - X_i \\
X_{f,j} &= X_f - X_j \\
X_{f,k} &= X_k - X_j \\
\|X_{j,i}\| &= \sqrt{X_{j,i}.X_{j,i}} = \alpha_1 \\
\frac{X_{i,f}.X_{j,f}}{\|X_{i,f}\|} &= \frac{X_{i,f}.X_{j,f}}{\sqrt{X_{i,f}.X_{i,f}}} = \alpha_2 \\
\sqrt{1 - \frac{(X_{i,f}.X_{j,f})^2}{||X_{i,f}\|^2}} &= \sqrt{1 - \frac{(X_{i,f}.X_{j,f})^2}{X_{i,f}.X_{i,f}}} = \beta_2 \\
L_{i,f} &= L_1 \\
L_{j,f} &= L_2 \\
L_{k,f} &= L_3
\end{aligned}
\tag{A.22}
$$

The expressions for α_2, β_2 are derived from the definition of the dot product: $A.B = \|A\|\|B\| \cos\theta$, the identities $A_x = \|A\| \cos\theta$ and $A_y = \|A\| \sin\theta$, and the identity $A^2 = A^2 \cos\theta + A^2 \sin\theta$, which lead to the following reductions:

$$
\begin{aligned}
\alpha_2 &= \|X_{i,f}\| \cos\theta_1 = \frac{X_{i,f}.X_{j,f}}{\|X_{i,f}\|} \\
\beta_2 &= \|X_{i,f}\| \sin\theta_1 = \sqrt{1 - \frac{(X_{i,f}.X_{j,f})^2}{||X_{i,f}\|^2}}
\end{aligned}
\tag{A.23}
$$

Inserting the above identities into Equation A.20 and simplifying gives the following untrans-

formed $\bar{X_3}$:

$$\bar{X_3} = \begin{pmatrix} \frac{L_1^2 - L_2^2 + X_{j,i}.X_{j,i}}{2\sqrt{X_{j,i}.X_{j,i}}} \\ \frac{X_{j,i}.X_{k,i}\left(L_2^2 - L_1^2\right) + X_{j,i}.X_{j,i}\left(L_1^2 - L_3^2 - X_{j,i}.X_{k,i} + X_{k,i}.X_{k,i}\right)}{2X_{j,i}.X_{j,i}\sqrt{X_{k,i}.X_{k,i} - \frac{\left(X_{j,i}.X_{k,i}\right)^2}{X_{j,i}.X_{j,i}}}} \\ L_1\sqrt{1 - \frac{\left(L_1^2 - L_2^2 + X_{j,i}.X_{j,i}\right)^2}{4X_{j,i}.X_{j,i}L_1^2}}\sqrt{1 - \frac{\left(X_{j,i}.X_{k,i}\left(L_1^2 - L_2^2\right) + X_{j,i}.X_{j,i}\left(-L_1^2 + L_3^2 + X_{j,i}.X_{k,i} - X_{k,i}.X_{k,i}\right)\right)^2}{\left((X_{j,i}.X_{k,i})^2 - X_{j,i}.X_{j,i}X_{k,i}.X_{k,i}\right)\left(L_2^4 - 2\left(L_1^2 + X_{j,i}.X_{j,i}\right)L_2^2 + \left(X_{j,i}.X_{j,i} - L_1^2\right)^2\right)}} \end{pmatrix} \tag{A.24}$$

$\bar{X_3}$ must then be transformed to the correct frame of reference by rotating to the basis formed by the triangle base, then translating it by X_i. As with the triangle case, the transformed X-axis basis and the first column of the rotation matrix is the same as with the triangle case. The transformed Z-axis basis is normal to the triangle base, and can be found by taking the cross product of $X_{j.i}$ and $X_{k,i}$ in that order and normalizing it:

$$\frac{X_{j,i} \times X_{k,i}}{\sqrt{X_{j,i} \times X_{k,i}.X_{j,i} \times X_{k,i}}} \tag{A.25}$$

The transformed Y-axis basis is the cross product of the transformed Z-axis basis and the transformed X-axis basis, and when combined with the other bases, the full rotation matrix $R_{i,j,k,f}$ is:

$$R_{i,j,k,f} = \left\{ \frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}, \frac{X_{j,i} \times X_{k,i}}{\sqrt{X_{j,i} \times X_{k,i}.X_{j,i} \times X_{k,i}}} \times \frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}, \frac{X_{j,i} \times X_{k,i}}{\sqrt{X_{j,i} \times X_{k,i}.X_{j,i} \times X_{k,i}}} \right\} \tag{A.26}$$

With the rotation matrix known, $\mathcal{F}_f$ is shown below, with products broken up over multiple lines for readability:

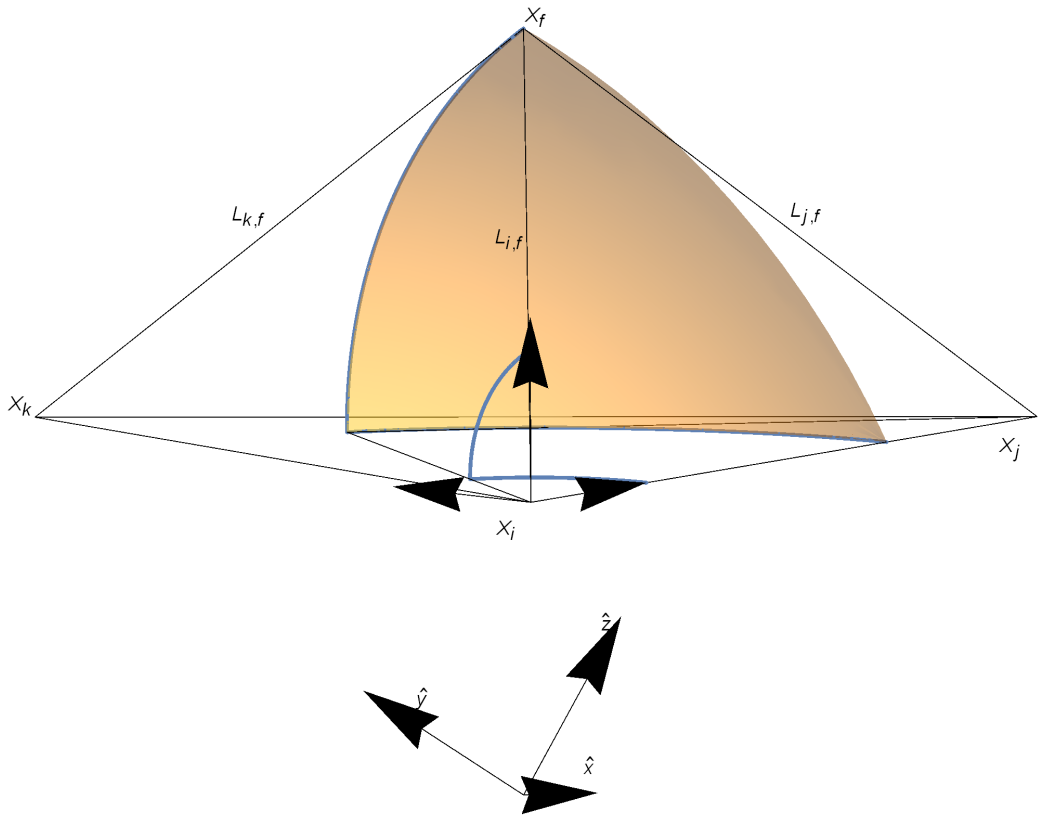


Figure A.4: The position of X_f as a function of $X_i, X_j, X_k, L_{i,f}, L_{j,f}, L_{k,f}$ for a tetrahedron example.

$$
\begin{aligned}
X_f &= \mathcal{F}_f(X_i, X_j, X_k, L_{i,f}, L_{j,f}, L_{k,f}) \\
&= X_i + R_{i,j,k,f}.\bar{X}_3 \\
&= X_i + \frac{X_{j,i}\left(L_{i,f}^2 - L_{j,f}^2 + X_{j,i}.X_{j,i}\right)}{2X_{j,i}.X_{j,i}} \\
&\quad + \left(\frac{X_{j,i} \times X_{k,i}}{\sqrt{X_{j,i} \times X_{k,i}.X_{j,i} \times X_{k,i}}} \times \frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}\right) \\
&\quad \left(\frac{\left(L_{j,f}^2 - L_{i,f}^2\right) X_{j,i}.X_{k,i} + X_{j,i}.X_{j,i}\left(L_{i,f}^2 - L_{k,f}^2 - X_{j,i}.X_{k,i} + X_{k,i}.X_{k,i}\right)}{2X_{j,i}.X_{j,i}\sqrt{X_{k,i}.X_{k,i} - \frac{(X_{j,i}.X_{k,i})^2}{X_{j,i}.X_{j,i}}}}\right) \\
&\quad + \left(\frac{L_{i,f}\sqrt{1 - \frac{\left(L_{i,f}^2 - L_{j,f}^2 + X_{j,i}.X_{j,i}\right)^2}{4L_{i,f}^2 X_{j,i}.X_{j,i}}} X_{j,i} \times X_{k,i}}{\sqrt{X_{j,i} \times X_{k,i}.X_{j,i} \times X_{k,i}}}\right) \\
&\quad \left(\sqrt{1 - \frac{\left(\left(L_{i,f}^2 - L_{j,f}^2\right) X_{j,i}.X_{k,i} + X_{j,i}.X_{j,i}\left(-L_{i,f}^2 + L_{k,f}^2 + X_{j,i}.X_{k,i} - X_{k,i}.X_{k,i}\right)\right)^2}{\left((X_{j,i}.X_{k,i})^2 - X_{j,i}.X_{j,i}X_{k,i}.X_{k,i}\right)\left(-2L_{j,f}^2\left(L_{i,f}^2 + X_{j,i}.X_{j,i}\right) + \left(X_{j,i}.X_{j,i} - L_{i,f}^2\right)^2 + L_{j,f}^4\right)}}\right)
\end{aligned}
\tag{A.27}
$$

A.2 Derivatives of Node Placement Functions

The derivative of the node placement functions is a foundation to all of the algorithms contained in this dissertation. However, the representation of the derivative functions is considerably long compared to the node placement functions themselves. I have made every effort to reduce the size of the representation.

Since each node placement function is a function of the positions of the base nodes and the lengths between the base nodes and the float node, there will be two Jacobian matrices: the derivative of the float node XYZ with respect to the base node XYZ variables, and with respect to the lengths.

A.2.1 Derivative of Single Strut With Respect to Length

By definition, the X-axis node is the second node placed, and is connected to the origin node:

$$\frac{\delta X_2}{\delta L_{1,2}} = \{1, 0, 0\}^T \tag{A.28}$$

A.2.2 Derivative of Triangle Function With Respect to Lengths

The triangle function returns two node position variables along with a fixed Z-axis position — the derivative of the triangle function with respect to lengths is a Jacobian matrix with two node position rows and two length columns. Using the notation described in Equation A.9, the derivatives of X_f with respect to $L_{i,f}$ and $L_{j,f}$ are:

$$\begin{aligned}\frac{\delta X_f}{\delta L_{i,f}} &= \frac{L_{i,f}X_{j,i}}{X_{j,i}.X_{j,i}} \\ &+\frac{\sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4L_{i,f}^2X_{j,i}.X_{j,i}}}\hat{z}\times X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}} \\ &+\frac{L_{i,f}\left(\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{2L_{i,f}^3X_{j,i}.X_{j,i}}-\frac{L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}}{L_{i,f}X_{j,i}.X_{j,i}}\right)\hat{z}\times X_{j,i}}{2\sqrt{X_{j,i}.X_{j,i}}\sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4L_{i,f}^2X_{j,i}.X_{j,i}}}}\end{aligned} \tag{A.29}$$

$$\begin{aligned}\frac{\delta X_f}{\delta L_{j,f}} &= -\frac{L_{j,f}X_{j,i}}{X_{j,i}.X_{j,i}} \\ &+\frac{L_{j,f}\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)\hat{z}\times X_{j,i}}{2L_{i,f}\left(X_{j,i}.X_{j,i}\right)^{3/2}\sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4L_{i,f}^2X_{j,i}.X_{j,i}}}}\end{aligned} \tag{A.30}$$

A.2.3 Derivative of Tetrahedron Function With Respect to Lengths

Like the triangle case, the derivative of the tetrahedron function with respect to lengths is a Jacobian matrix: three node position rows and three length columns. Also like the triangle case, the length variables do not appear within any vector in Equation 3.3, allowing a single expression for each length. Using the notation in Equation A.22, the derivatives of X_f with respect to $L_{i,f}, L_{j,f}, L_{k,f}$ are:

$$\begin{aligned}\frac{\delta X_f}{\delta L_{i,f}} &= \frac{L_{i,f}X_{j,i}}{X_{j,i}.X_{j,i}} + \frac{L_{i,f}\left(X_{j,i}.X_{j,i} - X_{j,i}.X_{k,i}\right)\frac{X_{j,i}\times X_{k,i}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}} \times \frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}}{X_{j,i}.X_{j,i}\sqrt{X_{k,i}.X_{k,i} - \frac{\left(X_{j,i}.X_{k,i}\right)^2}{X_{j,i}.X_{j,i}}}} \\ &+ \left(X_{j,i}\times X_{k,i}\left(X_{k,i}.X_{k,i}\left(L_{j,f}^2 - L_{i,f}^2\right) - 2\left(X_{j,i}.X_{k,i}\right)^2 + \right.\right. \\ &\left.\left. X_{j,i}.X_{k,i}\left(2L_{i,f}^2 - L_{j,f}^2 - L_{k,f}^2 + X_{k,i}.X_{k,i}\right) + X_{j,i}.X_{j,i}\left(-L_{i,f}^2 + L_{k,f}^2 + X_{j,i}.X_{k,i}\right)\right)\right) \\ &\left(\frac{1}{\left(X_{j,i}.X_{j,i}X_{k,i}.X_{k,i} - \left(X_{j,i}.X_{k,i}\right)^2\right)\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}\right) \\ &\left(\frac{1}{\sqrt{-\frac{-2L_{j,f}^2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2+L_{j,f}^4}{L_{i,f}^2X_{j,i}.X_{j,i}}}}\right) \\ &\left(\frac{1}{\sqrt{1-\frac{\left(\left(L_{i,f}^2-L_{j,f}^2\right)X_{j,i}.X_{k,i}+X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)\right)^2}{\left(\left(X_{j,i}.X_{k,i}\right)^2-X_{j,i}.X_{j,i}X_{k,i}.X_{k,i}\right)\left(-2L_{j,f}^2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2+L_{j,f}^4\right)}}}\right)\end{aligned} \tag{A.31}$$

$$\begin{aligned}\frac{\delta X_f}{\delta L_{j,f}} &= -\frac{L_{j,f}X_{j,i}}{X_{j,i}.X_{j,i}} + \frac{L_{j,f}X_{j,i}.X_{k,i}\frac{X_{j,i}\times X_{k,i}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}} \times \frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}}{X_{j,i}.X_{j,i}\sqrt{X_{k,i}.X_{k,i} - \frac{\left(X_{j,i}.X_{k,i}\right)^2}{X_{j,i}.X_{j,i}}}} \\ &+L_{j,f}X_{j,i}\times X_{k,i}\left(X_{k,i}.X_{k,i}\left(L_{i,f}^2 - L_{j,f}^2\right) - X_{j,i}.X_{k,i}\left(L_{i,f}^2 - L_{k,f}^2 + X_{k,i}.X_{k,i}\right) + X_{j,i}.X_{j,i}X_{k,i}.X_{k,i}\right) \\ &\left(\frac{1}{\left(X_{j,i}.X_{j,i}X_{k,i}.X_{k,i} - \left(X_{j,i}.X_{k,i}\right)^2\right)\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}\right) \\ &\left(\frac{1}{L_{i,f}\sqrt{-\frac{-2L_{j,f}^2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2+L_{j,f}^4}{L_{i,f}^2X_{j,i}.X_{j,i}}}}\right) \\ &\left(\frac{1}{\sqrt{1-\frac{\left(\left(L_{i,f}^2-L_{j,f}^2\right)X_{j,i}.X_{k,i}+X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)\right)^2}{\left(\left(X_{j,i}.X_{k,i}\right)^2-X_{j,i}.X_{j,i}X_{k,i}.X_{k,i}\right)\left(-2L_{j,f}^2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2+L_{j,f}^4\right)}}}\right)\end{aligned} \tag{A.32}$$

$$
\begin{aligned}
\frac{\delta X_f}{\delta L_{k,f}} = & -\frac{L_{k,f}\frac{X_{j,i}\times X_{k,i}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}\times\frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}}{\sqrt{X_{k,i}.X_{k,i}-\frac{\left(X_{j,i}.X_{k,i}\right)^2}{X_{j,i}.X_{j,i}}}} \\
& -\left(2L_{i,f}L_{k,f}X_{j,i}.X_{j,i}\sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4L_{i,f}^2X_{j,i}.X_{j,i}}}X_{j,i}\times X_{k,i}\right) \\
& \left(\left(L_{i,f}^2-L_{j,f}^2\right)X_{j,i}.X_{k,i}+X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)\right) \\
& \left(\frac{1}{\left(\left(X_{j,i}.X_{k,i}\right)^2-X_{j,i}.X_{j,i}X_{k,i}.X_{k,i}\right)\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}\right) \\
& \left(\frac{1}{-2L_{j,f}^2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2+L_{j,f}^4}\right) \\
& \left(\frac{1}{\sqrt{1-\frac{\left(\left(L_{i,f}^2-L_{j,f}^2\right)X_{j,i}.X_{k,i}+X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)\right)^2}{\left(\left(X_{j,i}.X_{k,i}\right)^2-X_{j,i}.X_{j,i}X_{k,i}.X_{k,i}\right)\left(-2L_{j,f}^2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2+L_{j,f}^4\right)}}}\right)
\end{aligned}
\tag{A.33}
$$

A.2.4 Derivative of Node Placement Functions With Respect to Node Positions

The derivatives with respect to node positions require differentiating by the components of the node position vectors, such as x_i. This leads to node vectors such as X_i being represented as functions of the component variables, i.e. $X_i(x_i)$. This, in turn, leads to the use of the derivative chain rule, which leads to an explosion in terms given the already-large number of terms in the functions:

$$
\frac{\delta f(g(x))}{\delta x} = \frac{\delta f(g(x))}{\delta g(x)}\frac{\delta g(x)}{\delta x} \tag{A.34}
$$

The chain rule will result in derivatives of the position vectors with respect to their individual constituent variables. Conveniently, each of the simplifying definitions found in Equations A.9 and A.22 have simple derivatives with respect to their constituent variables (definitions for $X_{k,i}$ are defined like the definitions for $X_{j,i}$):

$$
\begin{aligned}
\frac{\delta X_i}{\delta x_i} &= \frac{\delta\{x_i, y_i, z_i\}^T}{\delta x_i} = \{1,0,0\}^T = \hat{x} \\
\frac{\delta X_i}{\delta y_i} &= \frac{\delta\{x_i, y_i, z_i\}^T}{\delta y_i} = \{0,1,0\}^T = \hat{y} \\
\frac{\delta X_i}{\delta z_i} &= \frac{\delta\{x_i, y_i, z_i\}^T}{\delta z_i} = \{0,0,1\}^T = \hat{z} \\
\frac{\delta X_{j,i}}{\delta x_i} &= \frac{\delta(\{x_j, y_j, z_j\} - \{x_i, y_i, z_i\})^T}{\delta x_i} = \{-1,0,0\}^T = -\hat{x} \\
\frac{\delta X_{j,i}}{\delta y_i} &= \frac{\delta(\{x_j, y_j, z_j\} - \{x_i, y_i, z_i\})^T}{\delta y_i} = \{0,-1,0\}^T = -\hat{y} \\
\frac{\delta X_{j,i}}{\delta z_i} &= \frac{\delta(\{x_j, y_j, z_j\} - \{x_i, y_i, z_i\})^T}{\delta z_i} = \{0,0,-1\}^T = -\hat{z} \\
\frac{\delta X_{j,i}}{\delta x_j} &= \frac{\delta(\{x_j, y_j, z_j\} - \{x_i, y_i, z_i\})^T}{\delta x_j} = \{1,0,0\}^T = \hat{x} \\
\frac{\delta X_{j,i}}{\delta y_j} &= \frac{\delta(\{x_j, y_j, z_j\} - \{x_i, y_i, z_i\})^T}{\delta y_j} = \{0,1,0\}^T = \hat{y} \\
\frac{\delta X_{j,i}}{\delta z_j} &= \frac{\delta(\{x_j, y_j, z_j\} - \{x_i, y_i, z_i\})^T}{\delta z_j} = \{0,0,1\}^T = \hat{z}
\end{aligned}
\tag{A.35}
$$

The derivitaves of each of the node position vectors is a (possibly negated) basis normal vector, allowing for reduced expression count and interchangeability. For example, the derivative of a function f with respect to x_i is $\frac{\delta f(X_i)}{x_i} = f'(X_i)\hat{x}$, and can be transformed into the derivative for y_i by swapping $\hat{x}$ for $\hat{y}$, resulting in $\frac{\delta f(X_i)}{y_i} = f'(X_i)\hat{y}$. To generalize, I use the term v_i to represent a node position variable, and its generic basis vector $\hat{v}$ to represent the derivative of the node position vector. To find the derivative with respect to a specific variable, simply swap v_i and $\hat{v}$ with the proper values from Equation A.35.

The following two subsections present the full forms of the derivatives. The output was generated by Mathematica and imported as PDFs into this document. Each equation uses v and $\hat{v}$, allowing it to represent each of the three axis variables for the node in question being derived by swapping x, y, or z in for v. Equations A.36 and A.37 show the derivatives of the triangle function with respect to changes in the X_i and X_j nodes, respectively. Likewise, Equations A.38, A.39, and A.40 show the derivatives of the tetrahedron function with respect to changes in the X_i, X_j, and

X_k functions. The subsections that follow will display only the equations with no additional text, to minimize the footprints of the equations.

A.2.5 Derivative of Triangle Function With Respect to X_i

$$\frac{\delta X_f}{\delta v_i} =$$

$$\frac{\hat{z}\times(-\hat{v})\, L_{i,f}\, \sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,L_{i,f}^2\,X_{j,i}.X_{j,i}}}}{\sqrt{X_{j,i}.X_{j,i}}} + \frac{L_{i,f}\;\hat{z}\times X_{j,i}\left(\frac{((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,L_{i,f}^2\,(X_{j,i}.X_{j,i})^2}-\frac{((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)}{2\,L_{i,f}^2\;X_{j,i}.X_{j,i}}\right)}{2\sqrt{X_{j,i}.X_{j,i}}\;\sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,L_{i,f}^2\;X_{j,i}.X_{j,i}}}} -$$

$$\frac{L_{i,f}\;((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,L_{i,f}^2\;X_{j,i}.X_{j,i}}}\;\hat{z}\times X_{j,i}}{2\,(X_{j,i}.X_{j,i})^{3/2}} - \frac{\hat{v}\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)}{2\,X_{j,i}.X_{j,i}} -$$

$$\frac{X_{j,i}\;((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)}{2\,(X_{j,i}.X_{j,i})^2} + \frac{X_{j,i}\;((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))}{2\,X_{j,i}.X_{j,i}} + \hat{v}$$

(A.36)

A.2.6 Derivative of Triangle Function With Respect to X_j

$$\frac{\delta\, X_f}{\delta\, v_j} = \frac{L_{i,f}\ \hat{z}\times X_{j,i}\left(\frac{\left(\hat{v}.X_{j,i}+X_{j,i}.\hat{v}\right)\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,L_{i,f}^2\,\left(X_{j,i}.X_{j,i}\right)^2}-\frac{\left(\hat{v}.X_{j,i}+X_{j,i}.\hat{v}\right)\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)}{2\,L_{i,f}^2\ X_{j,i}.X_{j,i}}\right)}{2\sqrt{X_{j,i}.X_{j,i}}\ \sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,L_{i,f}^2\ X_{j,i}.X_{j,i}}}}+$$

$$\frac{\hat{z}\times\hat{v}\ L_{i,f}\ \sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,L_{i,f}^2\ X_{j,i}.X_{j,i}}}}{\sqrt{X_{j,i}.X_{j,i}}}-\frac{L_{i,f}\ (\hat{v}.X_{j,i}+X_{j,i}.\hat{v})\ \sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,L_{i,f}^2\ X_{j,i}.X_{j,i}}}\ \hat{z}\times X_{j,i}}{2\,(X_{j,i}.X_{j,i})^{3/2}}+$$

$$\frac{\hat{v}\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)}{2\,X_{j,i}.X_{j,i}}-\frac{X_{j,i}\ (\hat{v}.X_{j,i}+X_{j,i}.\hat{v})\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)}{2\,(X_{j,i}.X_{j,i})^2}+\frac{X_{j,i}\ (\hat{v}.X_{j,i}+X_{j,i}.\hat{v})}{2\,X_{j,i}.X_{j,i}} \tag{A.37}$$

A.2.7 Derivative of Tetrahedron Function With Respect to X_i

$$\frac{\delta X_f}{\delta v_i} =$$

$$\left(\frac{X_{j,i}\times X_{k,i}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}\times\frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}\left((X_{k,i}.(-\hat{v})-X_{j,i}.(-\hat{v}))\,X_{j,i}.X_{j,i}+((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\left(L_{i,f}^2-L_{k,f}^2-X_{j,i}.X_{k,i}+X_{k,i}.X_{k,i}\right)+\right.\right.$$

$$\left.\left.((-\hat{v}).X_{k,i}+X_{j,i}.(-\hat{v}))\left(L_{j,f}^2-L_{i,f}^2\right)\right)\right)\Bigg/\left(2\,X_{j,i}.X_{j,i}\sqrt{X_{k,i}.X_{k,i}-\frac{(X_{j,i}.X_{k,i})^2}{X_{j,i}.X_{j,i}}}\right)+\hat{v}+$$

$$\left(X_{j,i}\times X_{k,i}\;L_{i,f}\left(\frac{((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,(X_{j,i}.X_{j,i})^2\;L_{i,f}^2}-\frac{((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)}{2\,X_{j,i}.X_{j,i}\;L_{i,f}^2}\right)\right.$$

$$\left.\sqrt{1-\frac{\left(X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)+X_{j,i}.X_{k,i}\left(L_{i,f}^2-L_{j,f}^2\right)\right)^2}{((X_{j,i}.X_{k,i})^2-X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i})\left(L_{j,f}^4-2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)L_{j,f}^2+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2\right)}}\right)\Bigg/$$

$$\left(2\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}\sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,X_{j,i}.X_{j,i}\;L_{i,f}^2}}\right)+\frac{1}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}$$

$$((-\hat{v})\times X_{k,i}+X_{j,i}\times(-\hat{v}))\;L_{i,f}\sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,X_{j,i}.X_{j,i}\;L_{i,f}^2}}$$

$$\sqrt{1-\frac{\left(X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)+X_{j,i}.X_{k,i}\left(L_{i,f}^2-L_{j,f}^2\right)\right)^2}{((X_{j,i}.X_{k,i})^2-X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i})\left(L_{j,f}^4-2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)L_{j,f}^2+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2\right)}}-\frac{1}{2\,(X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i})^{3/2}}$$

$$X_{j,i}\times X_{k,i}\;(((-\hat{v})\times X_{k,i}+X_{j,i}\times(-\hat{v})).X_{j,i}\times X_{k,i}+X_{j,i}\times X_{k,i}.((-\hat{v})\times X_{k,i}+X_{j,i}\times(-\hat{v})))\;L_{i,f}\sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,X_{j,i}.X_{j,i}\;L_{i,f}^2}}$$

$$\sqrt{1-\frac{\left(X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)+X_{j,i}.X_{k,i}\left(L_{i,f}^2-L_{j,f}^2\right)\right)^2}{((X_{j,i}.X_{k,i})^2-X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i})\left(L_{j,f}^4-2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)L_{j,f}^2+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2\right)}}-$$

$$\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)\hat{v}}{2\,X_{j,i}.X_{j,i}}+\left(X_{j,i}\times X_{k,i}\;L_{i,f}\sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,X_{j,i}.X_{j,i}\;L_{i,f}^2}}\right.$$

$$\left(-\left(\left(2\left(X_{j,i}.X_{j,i}\,(X_{j,i}.(-\hat{v})-X_{k,i}.(-\hat{v}))+((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)+\right.\right.\right.\right.$$

$$\left.\left.\left.\left.((-\hat{v}).X_{k,i}+X_{j,i}.(-\hat{v}))\left(L_{i,f}^2-L_{j,f}^2\right)\right)\left(X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)+X_{j,i}.X_{k,i}\left(L_{i,f}^2-L_{j,f}^2\right)\right)\right)\right)\Big/$$

$$\left(\left((X_{j,i}.X_{k,i})^2-X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i}\right)\left(L_{j,f}^4-2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)L_{j,f}^2+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2\right)\right)+$$

$$\left((2\,((-\hat{v}).X_{k,i}+X_{j,i}.(-\hat{v}))\,X_{j,i}.X_{k,i}-X_{j,i}.X_{j,i}\,((-\hat{v}).X_{k,i}+X_{k,i}.(-\hat{v}))-((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\,X_{k,i}.X_{k,i}\right)$$

$$\left(X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)+X_{j,i}.X_{k,i}\left(L_{i,f}^2-L_{j,f}^2\right)\right)^2\Big)\Big/$$

$$\left(\left((X_{j,i}.X_{k,i})^2-X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i}\right)^2\left(L_{j,f}^4-2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)L_{j,f}^2+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2\right)\right)+$$

$$\left(\left(2\,((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)-2\,((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\,L_{j,f}^2\right)\right.$$

$$\left.\left(X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)+X_{j,i}.X_{k,i}\left(L_{i,f}^2-L_{j,f}^2\right)\right)^2\Big)\Big/\right.$$

$$\left.\left.\left(\left((X_{j,i}.X_{k,i})^2-X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i}\right)\left(L_{j,f}^4-2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)L_{j,f}^2+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2\right)^2\right)\right)\right)\Bigg/$$

$$\left(2\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}\sqrt{1-\frac{\left(X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)+X_{j,i}.X_{k,i}\left(L_{i,f}^2-L_{j,f}^2\right)\right)^2}{((X_{j,i}.X_{k,i})^2-X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i})\left(L_{j,f}^4-2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)L_{j,f}^2+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2\right)}}\right)$$

$$
+\left(\left(\frac{X_{j,i}\times X_{k,i}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}\times\left(-\frac{\hat{v}}{\sqrt{X_{j,i}.X_{j,i}}}-\frac{((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\,X_{j,i}}{2\,(X_{j,i}.X_{j,i})^{3/2}}\right)+\left(\frac{(-\hat{v})\times X_{k,i}+X_{j,i}\times(-\hat{v})}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}-\right.\right.
$$

$$
\left.\left.\frac{X_{j,i}\times X_{k,i}\,(((-\hat{v})\times X_{k,i}+X_{j,i}\times(-\hat{v})).X_{j,i}\times X_{k,i}+X_{j,i}\times X_{k,i}.((-\hat{v})\times X_{k,i}+X_{j,i}\times(-\hat{v})))}{2\,(X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i})^{3/2}}\right)\times\frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}\right)
$$

$$
\left(X_{j,i}.X_{j,i}\left(L_{i,f}^2-L_{k,f}^2-X_{j,i}.X_{k,i}+X_{k,i}.X_{k,i}\right)+X_{j,i}.X_{k,i}\left(L_{j,f}^2-L_{i,f}^2\right)\right)\Big/\left(2\,X_{j,i}.X_{j,i}\sqrt{X_{k,i}.X_{k,i}-\frac{(X_{j,i}.X_{k,i})^2}{X_{j,i}.X_{j,i}}}\right)+
$$

$$
\left(+\left(\frac{X_{j,i}\times X_{k,i}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}\times\frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}\left((X_{k,i}.(-\hat{v})-X_{j,i}.(-\hat{v}))\,X_{j,i}.X_{j,i}+\right.\right.\right.
$$

$$
\left.\left.\left.((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\left(L_{i,f}^2-L_{k,f}^2-X_{j,i}.X_{k,i}+X_{k,i}.X_{k,i}\right)+((-\hat{v}).X_{k,i}+X_{j,i}.(-\hat{v}))\left(L_{j,f}^2-L_{i,f}^2\right)\right)\right)\Big/
$$

$$
\left(2\,X_{j,i}.X_{j,i}\sqrt{X_{k,i}.X_{k,i}-\frac{(X_{j,i}.X_{k,i})^2}{X_{j,i}.X_{j,i}}}\right)+\frac{((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\,X_{j,i}}{2\,X_{j,i}.X_{j,i}}-
$$

$$
\frac{\frac{X_{j,i}\times X_{k,i}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}\times\frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\left(X_{j,i}.X_{j,i}\left(L_{i,f}^2-L_{k,f}^2-X_{j,i}.X_{k,i}+X_{k,i}.X_{k,i}\right)+X_{j,i}.X_{k,i}\left(L_{j,f}^2-L_{i,f}^2\right)\right)}{2\,(X_{j,i}.X_{j,i})^2\sqrt{X_{k,i}.X_{k,i}-\frac{(X_{j,i}.X_{k,i})^2}{X_{j,i}.X_{j,i}}}}-
$$

$$
\left(\frac{X_{j,i}\times X_{k,i}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}\times\frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}\left(\frac{((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\,(X_{j,i}.X_{k,i})^2}{(X_{j,i}.X_{j,i})^2}-\frac{2\,((-\hat{v}).X_{k,i}+X_{j,i}.(-\hat{v}))\,X_{j,i}.X_{k,i}}{X_{j,i}.X_{j,i}}+(-\hat{v}).X_{k,i}+X_{k,i}.(-\hat{v})\right)\right.
$$

$$
\left.\left(X_{j,i}.X_{j,i}\left(L_{i,f}^2-L_{k,f}^2-X_{j,i}.X_{k,i}+X_{k,i}.X_{k,i}\right)+X_{j,i}.X_{k,i}\left(L_{j,f}^2-L_{i,f}^2\right)\right)\right)\Big/
$$

$$
\left(4\,X_{j,i}.X_{j,i}\left(X_{k,i}.X_{k,i}-\frac{(X_{j,i}.X_{k,i})^2}{X_{j,i}.X_{j,i}}\right)^{3/2}\right)-\frac{((-\hat{v}).X_{j,i}+X_{j,i}.(-\hat{v}))\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)X_{j,i}}{2\,(X_{j,i}.X_{j,i})^2}
\tag{A.38}
$$

A.2.8 Derivative of Tetrahedron Function With Respect to X_j

$$\frac{\delta X_f}{\delta v_j} = \frac{\hat{v}\left(L_{i,f}^2 - L_{j,f}^2 + X_{j,i}.X_{j,i}\right)}{2\,X_{j,i}.X_{j,i}} + \left(X_{j,i}\times X_{k,i}\; L_{i,f}\sqrt{1 - \frac{\left(L_{i,f}^2 - L_{j,f}^2 + X_{j,i}.X_{j,i}\right)^2}{4\,X_{j,i}.X_{j,i}\;L_{i,f}^2}} \right.$$

$$\left(\frac{(2\,\hat{v}.X_{k,i}\;X_{j,i}.X_{k,i} - (\hat{v}.X_{j,i} + X_{j,i}.\hat{v})\,X_{k,i}.X_{k,i})\left(X_{j,i}.X_{k,i}\left(L_{i,f}^2 - L_{j,f}^2\right) + X_{j,i}.X_{j,i}\left(-L_{i,f}^2 + L_{k,f}^2 + X_{j,i}.X_{k,i} - X_{k,i}.X_{k,i}\right)\right)^2}{((X_{j,i}.X_{k,i})^2 - X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i})^2\left(L_{j,f}^4 - 2\left(L_{i,f}^2 + X_{j,i}.X_{j,i}\right)L_{j,f}^2 + \left(X_{j,i}.X_{j,i} - L_{i,f}^2\right)^2\right)} + \right.$$

$$\left((2\,(\hat{v}.X_{j,i} + X_{j,i}.\hat{v})\left(X_{j,i}.X_{j,i} - L_{i,f}^2\right) - 2\,(\hat{v}.X_{j,i} + X_{j,i}.\hat{v})\,L_{j,f}^2\right)$$

$$\left(X_{j,i}.X_{k,i}\left(L_{i,f}^2 - L_{j,f}^2\right) + X_{j,i}.X_{j,i}\left(-L_{i,f}^2 + L_{k,f}^2 + X_{j,i}.X_{k,i} - X_{k,i}.X_{k,i}\right)\right)^2\Big)\Big/$$

$$\left(\left((X_{j,i}.X_{k,i})^2 - X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i}\right)\left(L_{j,f}^4 - 2\left(L_{i,f}^2 + X_{j,i}.X_{j,i}\right)L_{j,f}^2 + \left(X_{j,i}.X_{j,i} - L_{i,f}^2\right)^2\right)\right) -$$

$$\left(2\left(\hat{v}.X_{k,i}\;X_{j,i}.X_{j,i} + \hat{v}.X_{k,i}\left(L_{i,f}^2 - L_{j,f}^2\right) + (\hat{v}.X_{j,i} + X_{j,i}.\hat{v})\left(-L_{i,f}^2 + L_{k,f}^2 + X_{j,i}.X_{k,i} - X_{k,i}.X_{k,i}\right)\right)\right.$$

$$\left.\left.\left(X_{j,i}.X_{k,i}\left(L_{i,f}^2 - L_{j,f}^2\right) + X_{j,i}.X_{j,i}\left(-L_{i,f}^2 + L_{k,f}^2 + X_{j,i}.X_{k,i} - X_{k,i}.X_{k,i}\right)\right)\right)\Big/\right.$$

$$\left.\left(\left((X_{j,i}.X_{k,i})^2 - X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i}\right)\left(L_{j,f}^4 - 2\left(L_{i,f}^2 + X_{j,i}.X_{j,i}\right)L_{j,f}^2 + \left(X_{j,i}.X_{j,i} - L_{i,f}^2\right)^2\right)\right)\right)\Bigg)\Bigg/$$

$$\left(2\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}\;\sqrt{1 - \frac{\left(X_{j,i}.X_{k,i}\left(L_{i,f}^2 - L_{j,f}^2\right) + X_{j,i}.X_{j,i}\left(-L_{i,f}^2 + L_{k,f}^2 + X_{j,i}.X_{k,i} - X_{k,i}.X_{k,i}\right)\right)^2}{((X_{j,i}.X_{k,i})^2 - X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i})\left(L_{j,f}^4 - 2\left(L_{i,f}^2 + X_{j,i}.X_{j,i}\right)L_{j,f}^2 + \left(X_{j,i}.X_{j,i} - L_{i,f}^2\right)^2\right)}}\right) +$$

$$\frac{\frac{X_{j,i}\times X_{k,i}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}\times\frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}\left(-\hat{v}.X_{k,i}\;X_{j,i}.X_{j,i} + \hat{v}.X_{k,i}\left(L_{j,f}^2 - L_{i,f}^2\right) + (\hat{v}.X_{j,i} + X_{j,i}.\hat{v})\left(L_{i,f}^2 - L_{k,f}^2 - X_{j,i}.X_{k,i} + X_{k,i}.X_{k,i}\right)\right)}{2\,X_{j,i}.X_{j,i}\sqrt{X_{k,i}.X_{k,i} - \frac{(X_{j,i}.X_{k,i})^2}{X_{j,i}.X_{j,i}}}} +$$

$$\left(\left(\left(\left(\frac{\hat{v}\times X_{k,i}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}} - \frac{X_{j,i}\times X_{k,i}\;(\hat{v}\times X_{k,i}.X_{j,i}\times X_{k,i} + X_{j,i}\times X_{k,i}.\hat{v}\times X_{k,i})}{2\,(X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i})^{3/2}}\right)\times\frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}} + \right.\right.\right.$$

$$\left.\left.\left.\frac{X_{j,i}\times X_{k,i}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}\times\left(\frac{\hat{v}}{\sqrt{X_{j,i}.X_{j,i}}} - \frac{(\hat{v}.X_{j,i} + X_{j,i}.\hat{v})\,X_{j,i}}{2\,(X_{j,i}.X_{j,i})^{3/2}}\right)\right)\right.$$

$$\left.\left(X_{j,i}.X_{k,i}\left(L_{j,f}^2 - L_{i,f}^2\right) + X_{j,i}.X_{j,i}\left(L_{i,f}^2 - L_{k,f}^2 - X_{j,i}.X_{k,i} + X_{k,i}.X_{k,i}\right)\right)\right)\Big/\left(2\,X_{j,i}.X_{j,i}\sqrt{X_{k,i}.X_{k,i} - \frac{(X_{j,i}.X_{k,i})^2}{X_{j,i}.X_{j,i}}}\right) -$$

$$\frac{\frac{X_{j,i}\times X_{k,i}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}\times\frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}\;(\hat{v}.X_{j,i} + X_{j,i}.\hat{v})\left(X_{j,i}.X_{k,i}\left(L_{j,f}^2 - L_{i,f}^2\right) + X_{j,i}.X_{j,i}\left(L_{i,f}^2 - L_{k,f}^2 - X_{j,i}.X_{k,i} + X_{k,i}.X_{k,i}\right)\right)}{2\,(X_{j,i}.X_{j,i})^2\sqrt{X_{k,i}.X_{k,i} - \frac{(X_{j,i}.X_{k,i})^2}{X_{j,i}.X_{j,i}}}} -$$

$$\left(\frac{X_{j,i}\times X_{k,i}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}\times\frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}\left(\frac{(\hat{v}.X_{j,i} + X_{j,i}.\hat{v})\,(X_{j,i}.X_{k,i})^2}{(X_{j,i}.X_{j,i})^2} - \frac{2\,\hat{v}.X_{k,i}\;X_{j,i}.X_{k,i}}{X_{j,i}.X_{j,i}}\right)\right.$$

$$\left.\left(X_{j,i}.X_{k,i}\left(L_{j,f}^2 - L_{i,f}^2\right) + X_{j,i}.X_{j,i}\left(L_{i,f}^2 - L_{k,f}^2 - X_{j,i}.X_{k,i} + X_{k,i}.X_{k,i}\right)\right)\right)\Big/\left(4\,X_{j,i}.X_{j,i}\left(X_{k,i}.X_{k,i} - \frac{(X_{j,i}.X_{k,i})^2}{X_{j,i}.X_{j,i}}\right)^{3/2}\right)$$

$$+\frac{\hat{v}\times X_{k,i}\; L_{i,f}\sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,X_{j,i}.X_{j,i}\;L_{i,f}^2}}\;\sqrt{1-\frac{\left(X_{j,i}.X_{k,i}\left(L_{i,f}^2-L_{j,f}^2\right)+X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)\right)^2}{\left((X_{j,i}.X_{k,i})^2-X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i}\right)\left(L_{j,f}^4-2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)L_{j,f}^2+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2\right)}}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}-$$

$$\frac{1}{2\,(X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i})^{3/2}}\;X_{j,i}\times X_{k,i}\;(\hat{v}\times X_{k,i}.X_{j,i}\times X_{k,i}+X_{j,i}\times X_{k,i}.\hat{v}\times X_{k,i})\;L_{i,f}\sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,X_{j,i}.X_{j,i}\;L_{i,f}^2}}$$

$$\sqrt{1-\frac{\left(X_{j,i}.X_{k,i}\left(L_{i,f}^2-L_{j,f}^2\right)+X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)\right)^2}{((X_{j,i}.X_{k,i})^2-X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i})\left(L_{j,f}^4-2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)L_{j,f}^2+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2\right)}}+$$

$$\left(+\left(X_{j,i}\times X_{k,i}\;L_{i,f}\left(\frac{(\hat{v}.X_{j,i}+X_{j,i}.\hat{v})\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,(X_{j,i}.X_{j,i})^2\;L_{i,f}^2}-\frac{(\hat{v}.X_{j,i}+X_{j,i}.\hat{v})\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)}{2\,X_{j,i}.X_{j,i}\;L_{i,f}^2}\right)\right.\right.$$

$$\left.\left.\sqrt{1-\frac{\left(X_{j,i}.X_{k,i}\left(L_{i,f}^2-L_{j,f}^2\right)+X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)\right)^2}{((X_{j,i}.X_{k,i})^2-X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i})\left(L_{j,f}^4-2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)L_{j,f}^2+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2\right)}}\right)\Bigg/$$

$$\left(2\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}\;\sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,X_{j,i}.X_{j,i}\;L_{i,f}^2}}\right)+\frac{(\hat{v}.X_{j,i}+X_{j,i}.\hat{v})\;X_{j,i}}{2\,X_{j,i}.X_{j,i}}-\frac{(\hat{v}.X_{j,i}+X_{j,i}.\hat{v})\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)X_{j,i}}{2\,(X_{j,i}.X_{j,i})^2} \tag{A.39}$$

A.2.9 Derivative of Tetrahedron Function With Respect to X_k

$$
\frac{\delta X_f}{\delta v_k} =
$$

$$
\frac{X_{j,i}\times\hat{v}\sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,X_{j,i}.X_{j,i}\;L_{i,f}^2}}\sqrt{1-\frac{\left(X_{j,i}.X_{k,i}\left(L_{i,f}^2-L_{j,f}^2\right)+X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)\right)^2}{\left((X_{j,i}.X_{k,i})^2-X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i}\right)\left(L_{j,f}^4-2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)L_{j,f}^2+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2\right)}}\;L_{i,f}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}-
$$

$$
\frac{1}{2\left(X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}\right)^{3/2}}X_{j,i}\times X_{k,i}\left(X_{j,i}\times\hat{v}.X_{j,i}\times X_{k,i}+X_{j,i}\times X_{k,i}.X_{j,i}\times\hat{v}\right)\sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,X_{j,i}.X_{j,i}\;L_{i,f}^2}}
$$

$$
\sqrt{1-\frac{\left(X_{j,i}.X_{k,i}\left(L_{i,f}^2-L_{j,f}^2\right)+X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)\right)^2}{\left((X_{j,i}.X_{k,i})^2-X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i}\right)\left(L_{j,f}^4-2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)L_{j,f}^2+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2\right)}}\;L_{i,f}+
$$

$$
\left(X_{j,i}\times X_{k,i}\sqrt{1-\frac{\left(L_{i,f}^2-L_{j,f}^2+X_{j,i}.X_{j,i}\right)^2}{4\,X_{j,i}.X_{j,i}\;L_{i,f}^2}}\right.
$$

$$
\left(\frac{\left(2\,X_{j,i}.\hat{v}\;X_{j,i}.X_{k,i}-X_{j,i}.X_{j,i}\left(\hat{v}.X_{k,i}+X_{k,i}.\hat{v}\right)\right)\left(X_{j,i}.X_{k,i}\left(L_{i,f}^2-L_{j,f}^2\right)+X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)\right)^2}{\left((X_{j,i}.X_{k,i})^2-X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i}\right)^2\left(L_{j,f}^4-2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)L_{j,f}^2+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2\right)}-\right.
$$

$$
\left(2\left(X_{j,i}.X_{j,i}\left(-\hat{v}.X_{k,i}+X_{j,i}.\hat{v}-X_{k,i}.\hat{v}\right)+X_{j,i}.\hat{v}\left(L_{i,f}^2-L_{j,f}^2\right)\right)\right.
$$

$$
\left.\left.\left(X_{j,i}.X_{k,i}\left(L_{i,f}^2-L_{j,f}^2\right)+X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)\right)\right)\Big/
$$

$$
\left.\left.\left(\left(X_{j,i}.X_{k,i}\right)^2-X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i}\right)\left(L_{j,f}^4-2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)L_{j,f}^2+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2\right)\right)L_{i,f}\right)\Big/
$$

$$
\left(2\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}\sqrt{1-\frac{\left(X_{j,i}.X_{k,i}\left(L_{i,f}^2-L_{j,f}^2\right)+X_{j,i}.X_{j,i}\left(-L_{i,f}^2+L_{k,f}^2+X_{j,i}.X_{k,i}-X_{k,i}.X_{k,i}\right)\right)^2}{\left((X_{j,i}.X_{k,i})^2-X_{j,i}.X_{j,i}\;X_{k,i}.X_{k,i}\right)\left(L_{j,f}^4-2\left(L_{i,f}^2+X_{j,i}.X_{j,i}\right)L_{j,f}^2+\left(X_{j,i}.X_{j,i}-L_{i,f}^2\right)^2\right)}}\right)+
$$

$$
\frac{\frac{X_{j,i}\times X_{k,i}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}\times\frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}\left(X_{j,i}.X_{j,i}\left(\hat{v}.X_{k,i}-X_{j,i}.\hat{v}+X_{k,i}.\hat{v}\right)+X_{j,i}.\hat{v}\left(L_{j,f}^2-L_{i,f}^2\right)\right)}{2\,X_{j,i}.X_{j,i}\sqrt{X_{k,i}.X_{k,i}-\frac{\left(X_{j,i}.X_{k,i}\right)^2}{X_{j,i}.X_{j,i}}}}+
$$

$$
\left(\left(\left(\frac{X_{j,i}\times\hat{v}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}-\frac{X_{j,i}\times X_{k,i}\left(X_{j,i}\times\hat{v}.X_{j,i}\times X_{k,i}+X_{j,i}\times X_{k,i}.X_{j,i}\times\hat{v}\right)}{2\left(X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}\right)^{3/2}}\right)\times\frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}\right.\right.
$$

$$
\left.\left.\left(X_{j,i}.X_{k,i}\left(L_{j,f}^2-L_{i,f}^2\right)+X_{j,i}.X_{j,i}\left(L_{i,f}^2-L_{k,f}^2-X_{j,i}.X_{k,i}+X_{k,i}.X_{k,i}\right)\right)\right)\Big/\left(2\,X_{j,i}.X_{j,i}\sqrt{X_{k,i}.X_{k,i}-\frac{\left(X_{j,i}.X_{k,i}\right)^2}{X_{j,i}.X_{j,i}}}\right)-
$$

$$
\frac{\frac{X_{j,i}\times X_{k,i}}{\sqrt{X_{j,i}\times X_{k,i}.X_{j,i}\times X_{k,i}}}\times\frac{X_{j,i}}{\sqrt{X_{j,i}.X_{j,i}}}\left(\hat{v}.X_{k,i}-\frac{2\,X_{j,i}.\hat{v}\,X_{j,i}.X_{k,i}}{X_{j,i}.X_{j,i}}+X_{k,i}.\hat{v}\right)\left(X_{j,i}.X_{k,i}\left(L_{j,f}^2-L_{i,f}^2\right)+X_{j,i}.X_{j,i}\left(L_{i,f}^2-L_{k,f}^2-X_{j,i}.X_{k,i}+X_{k,i}.X_{k,i}\right)\right)}{4\,X_{j,i}.X_{j,i}\left(X_{k,i}.X_{k,i}-\frac{\left(X_{j,i}.X_{k,i}\right)^2}{X_{j,i}.X_{j,i}}\right)^{3/2}}
\tag{A.40}
$$

Appendix B

Calculation of Truss Covariance Conditioned on Node Positions

The simple covariance matrix derivation in Equation 5.5 is not the only way to derive it. I discovered the following form prior to the one in Equation 5.5, and I record it here for completeness.

The multivariate canonical form, described in Section 5.3.1, can be used instead with the general form for the propagation of covariance through a function [11], reiterated here:

$$\Sigma_{f(x)} = \frac{df(x)}{dx}\Sigma_x \frac{df(x)}{dx}^T \tag{B.1}$$

The probability of a float node f can be expressed as a function of the probabilities of the base nodes and the struts connecting to base instead of every length going back to the beginning. The result is a linear system whose mean is the sum of the desired position and the Jacobian delta (with respect to the base positions), and whose covariance is the length covariance transformed by the node function:

$$\begin{aligned} J_{f,base} &= \frac{dX_f(X_i, X_j, X_k, L_{i,f}, L_{j,f}, L_{k,f})}{dX_{i,j,k}} \\ J_{f,lengths} &= \frac{dX_f(X_i, X_j, X_k, L_{i,f}, L_{j,f}, L_{k,f})}{dL_{i,j,k,f}} \\ p(X_f) &= \mathcal{N}(X_{f,desired} + J_{f,base}(X_{i,j,k} - X_{i,j,k,desired})J_{f,base}^T \\ &\quad , J_{f,lengths}(\sigma_L^2 I)J_{f,lengths}^T) \end{aligned} \tag{B.2}$$

When no measurements are used, this method can be used to produce an identical covariance matrix to the one found by Equation 5.5 through the following steps:

- Find the canonical form of the initial strut X_1, which depends only on one length, and call the result $k_{running}, h_{running}$.

- For each step, find the linear conditional $k_{f,base}, h_{f,base}$ using Equations B.2 and 5.27, and find the canonical product of it and $k_{running}, h_{running}$ and set $k_{running}, h_{running}$ to the result.

- Marginalize the completed $k_{running}, h_{running}$ over all of the strut lengths to get the covariance in Equation 5.5.