frontiers | Frontiers in Robotics and AI

TYPE Original Research
PUBLISHED 30 October 2025
DOI 10.3389/frobt.2025.1426676

OPEN ACCESS

EDITED BY
Dongbing Gu,
University of Essex, United Kingdom

REVIEWED BY
Barış Can Yalçın,
University of Luxembourg, Luxembourg
Ruiheng Zhang,
Beijing Institute of Technology, China

*CORRESPONDENCE
Samantha Chapin,
sglassner@vt.edu
William Chapin,
wchapin@vt.edu

RECEIVED 01 May 2024
ACCEPTED 04 July 2025
PUBLISHED 30 October 2025

CITATION
Chapin S, Chapin W and Komendera E (2025) Semantic and fiducial-aided graph simultaneous localization and mapping (SF-GraphSLAM) for robotic in-space assembly and servicing of large truss structures.
Front. Robot. AI 12:1426676.
doi: 10.3389/frobt.2025.1426676

Semantic and fiducial-aided graph simultaneous localization and mapping (SF-GraphSLAM) for robotic in-space assembly and servicing of large truss structures

Samantha Chapin*, William Chapin* and Erik Komendera

Field and Space Experimental Robotics (FASER) Laboratory, Mechanical Engineering Department, Virginia Polytechnic Institute and State University, Blacksburg, VA, United States

This article proposes a method that uses information about modules and desired assembly locations within a large truss structure to create a semantic and fiducial aided graph simultaneous localization and mapping (SF-GraphSLAM) algorithm that is better tailored for use during robotic in-space assembly and servicing operations. This is achieved by first reducing the number of modules using a mixed assembly method vs. a strut-by-strut method. Then, each module is correlated to a visual tag (in this article, an AprilTag) to reduce the number of elements being observed further from the number of sub-struts in that module to a single AprilTag marker. Two tags are required to ensure proper deployment of most deployable modules. Subsequently, we are able to use semantic information about the desired transformation matrix between any two adjacent module AprilTags within the desired assembly structure. For our experimentation, we expanded a factor graph smoothing and mapping model and added the semantic information, looking at the smaller number of landmark AprilTags, with a camera representing the robot for simplicity. The mathematical approach to arrive at this new method is included in this article, as are simulations to test it against the state of the art (SOA) using no structural knowledge. Overall, this research contributes to the SOA for both general SLAM work and, more specifically, to the underdeveloped field of SLAM for in-space assembly and servicing of large truss structures. It is critical to ensure that as a robot is assembling the modules, each module is within the desired tolerances to ensure the final structure is within the design requirements. Being able to build a virtual twin of the truss structure as it is being assembled is a key tent pole in achieving large space structures.

KEYWORDS

simultaneous localization and mapping, semantic, fiducial, vision, metrology, robotics, in-space servicing assembly and manufacturing, in-space structures

1 Introduction

This article describes the creation of the semantic and fiducial aided graph simultaneous localization and mapping (SF-GraphSLAM) method that is tailored for robotic assembly and servicing of large truss structures, including deployable modules. This research is novel because it will be the first to integrate the semantic input of truss modules, relative

goal positioning of modules to create the desired end structure, and fiducials into a SLAM algorithm to greatly reduce the state vector for robotic assembly of large structures. Working on the SF-GraphSLAM algorithm in parallel with the development of a space truss methodology focused on mixed assembly of deployable and close-out assembled modules allowed for the development of a test case scenario. The built on-orbit robotically assembled gigatruss (BORG) uses an array of deployable modules that are arranged in a checkerboard pattern and connects them with strut and square close-out elements. Using this approach reduces the number of unique modules required to assemble a given truss structure. This greatly benefits the SF-GraphSLAM case because there are fewer structure state vectors due to the fewer modules, which results in quicker processing speeds when analysis is performed between assembly steps. For testing purposes, a $3 \times 3 \times 3$ truss structure was developed, but the state vector reduction benefit increases as the structure is scaled.

The SF-GraphSLAM goal is to combine methods of focusing measurements on sparsely placed fiducials and using knowledge about the structure's deployment mechanisms and assembled component relationships to be able to quickly predict the structure's state and add robustness to pose and measurement errors. This new method was based on the existing GraphSLAM approach, which is the state-of-the-art (SOA) method chosen to compare against. First, mathematical derivations for how semantic knowledge could be added to a GraphSLAM base were completed. Then, simulations of the GraphSLAM SOA and SF-GraphSLAM algorithms were created in order to test the effectiveness on an example BORG truss model. Creating a SLAM method tailored to the robotic assembly of truss structures allows this research to contribute greatly to the SOA of the larger field of robotic in-space servicing, assembly, and manufacturing (ISAM). Although space robotic operations have heritage, there are unique challenges presented by working on the problem of robotically assembling large space trusses. Providing a SLAM method for aiding with the autonomous robotic assembly of movable modules to create larger structures will be critical for future missions, such as robotically assembling a large antenna structure or a space telescope. The core methodology examined how to best utilize information in a large-scale structure environment, including non-static flexible or deployable modules. Adequately mapping the structure environment could have broader applications to the field of robotic operations dealing with terrestrial structures such as bridge surveying.

This article focuses on the description and simulated validation of the SF-GraphSLAM algorithm; for details on the physical implementation and validation, please refer to Chapin et al. (2024).

2 Materials and methods

2.1 In-space assembly and servicing background

The in-space servicing, assembly, and manufacturing (ISAM) field is vast and has promises to revolutionize the space ecosystem (Cavaciuti et al., 2022) by allowing space assets to be created in new ways and maintained over longer lifetimes. Robotic ISAM enables the construction of structures on scales never seen before in space. No longer constrained by the size and mass limits of a single launch vehicle transit to space, multiple launches could be utilized to send the raw material for manufacturing or modules for assembly to create a variety of large space structures. Furthermore, designing structures to be assembled inherently provides an avenue for more servicing opportunities. Robotically servicing existing space assets can be extremely useful, and structures designed to be maintained robotically can offer robustness to unexpected failure during and after beginning operation.

Manufacturing, assembling, and servicing large structures have specific challenges, such as thermal robustness, feasibility of scaling, determining the size of modularization, and interfaces, that need to be addressed to attain even larger structures in space. We are focusing on addressing the concern of ensuring that the final assembled structure meets the requirements necessary for operation. To date, the biggest structures assembled or serviced in space, the International Space Station (Garcia, 2022) and the Hubble Space Telescope (Garner, 2018), were built through astronaut extravehicular activities with aid from large robotic manipulators. As the scale of structures in space increases, the reliance on astronaut-aided operations is less practical, and more autonomous robotic solutions are crucial. To build the next generation of large space telescopes and other structures, such as antennas, the ability to autonomously robotically assemble structures to the required precision will be crucial.

When trying to assemble large space structures, a robotic system is required to handle the very large quantity of states resulting from each strut being able to be represented by six state variables. As the structure scales, this problem only increases, as do the physical limitations of being able to properly collect data from cameras viewing a possibly dense collection of struts simultaneously. Additional complexity is introduced when the structure is actively being assembled because the struts are then not static, and their overall state of being in storage, being manipulated, or being placed in the final structure must be considered. In addition, for large structures, the smaller assembly robots will need to either move along the structure or around it to be able to fully assemble the much larger structure. The work described here evolves from an earlier study using multiple robots and EKF-SLAM to assemble and deploy a prototype solar array (Komendera et al., 2017).

Due to the broad interest in autonomously assembled structures, there is a wide range of previous and current related autonomous ISAM studies covering the full breadth of research challenges. The following list is a small selection of articles covering a range of areas of research needed to enable autonomous ISAM but is by no means complete. Precision autonomous truss assembly is performed by robots that move over the structure and mechanically join each truss cell (Gregg and Cheung, 2024). A novel pose estimation approach via sensor fusion for autonomously assembled space structure elements is described by Moser et al. (2024). A method for autonomously planning and verifying the assembly sequences of large space structures is described by Rodríguez et al. (2021). Multiple current ISAM studies and activities at the Jet Propulsion Laboratory are described by Mukherjee (2023). Many approaches for ISAM favor modularity in the assembling agents and in the structure (Post et al., 2021).

2.2 Space vision background

Space robotics utilizes machine vision algorithms that allow a robotic system to understand its environment with imaging sensors to achieve two main objectives: (1) pose estimation of the robots relative to their environment and (2) locations of important features within the environment (Henshaw et al., 2022). The Orbit Servicing, Assembly and Manufacturing (OSAM) State of Play recorded space inspection and meteorology projects and differentiated them by sensor type (visual or other), operation mode (free-flying or anchored), and flight status (Dale Arney and Mulvaney, 2023). There were six recorded free-flyers utilizing vision sensors, all previously flown, and one other sensor method in development. Two anchored examples utilized vision sensors, one flown and one in development, while three other examples used a different type of sensor (Dale Arney and Mulvaney, 2023). In 2007, the Orbital Express Demonstration System (OEDS) performed a flight demonstration servicing the NextSat spacecraft (Ogilvie et al., 2008). This included an autonomous free-flying capture with a robotic arm and is enabled with Vis-STAR, a machine vision system (Henshaw et al., 2022). This flight test had two modes of vision operation depending on the range of the spacecraft. When the NextSat was more than 10 m away, the outline of the spacecraft was compared to an outline database generated from a 3D model to estimate the range and orientation (Leinz et al., 2008). NextSat had difficulty performing this estimation with spacecraft that were rotationally symmetric. When the spacecraft was within 10 m, and the camera's field of view could no longer see the entire outline, optical fiducials on the client satellite were relied upon. This is only one example of flight heritage for the use of AprilTag-like, black and white, square-patterned fiducial decals. It has been proposed to equip satellites with fiducials to enable the possibility of easier future robotic servicing for the low cost of some vestigial mass (Reed et al., 2017). The planned On-Orbit Servicing, Assembly and Manufacturing (OSAM)-1 and Robotic Servicing of Geosynchronous Satellites (RSGS) satellite servicing missions both plan to utilize machine vision to allow for autonomous grappling of the client spacecraft's Marman ring (Obermark et al., 2007). There has also been work on exploiting map landmark-based simultaneous localization and mapping (SLAM) for the purpose of relative navigation in space applications for tasks such as rendezvous proximity operations (RPO) (Ticozzi and Tsiotras, 2025; Bettens et al., 2024; Schlenker et al., 2019). This includes efforts to use known models of spacecraft to be able to identify and track them in complex scenarios, including uncontrolled tumbling (Tweddle et al., 2015; Asri1 and Zhu, 2025).

2.3 Simultaneous localization and mapping background

SLAM describes the methodology of using sensor data to map a robot's surroundings while localizing itself in those surroundings. The state of the art is to use visual SLAM (VSLAM) with either cameras or LIDAR to collect data on all elements surrounding a robot (Abaspur Kazerouni et al., 2022). SLAM can also combine sensors to add additional data into the estimation, such as an inertial measurement unit (IMU) to help track a robot's movement. Often, it is assumed the robot is mobile and the observed objects are static (Chen et al., 2022). Some limitations include slow processing for real-time operations (Chen et al., 2022).

This work expands on the use of factor graphs (Dellaert and Kaess, 2017), which are a commonly used framework in modern SLAM approaches. An early implementation in SLAM of factor graphs is the GraphSLAM algorithm (Thrun and Montemerlo, 2006). GraphSLAM, and factor graphs in general, solve for the optimal posterior of the state estimate by treating the posterior as a least squares problem. In a typical SLAM problem, factors are one of three types: a prior estimate of the state of the environment and agents, a measurement generally linking some aspect of the environment with the time and state of the agent taking the measurement, and a state transition probability linking an agent's state with its prior state. When visualized as a graph, each factor represents an edge, and each estimated state is a node. Each factor is also represented by a function $\phi(x)$ that represents a conditional or prior probability of either a measurement or a state transition. When the conditional or prior probabilities are assumed to be Gaussian, the posterior can be reduced to the sum of the negative log conditional probability functions.

In this work, the addition of factors representing the mechanisms of the structure is a novel contribution. The name "SF-GraphSLAM" acknowledges the origin of this approach with the GraphSLAM algorithm. Although other, newer algorithms branch from the GraphSLAM approach, such as factor graph-based formulations, SF-GraphSLAM is compared directly to GraphSLAM to determine its performance increases against the method it was based on as a control.

2.3.1 Semantic SLAM background

Semantic SLAM can detect and identify target objects in a scene using semantic information provided beforehand (Chen et al., 2022). Semantic information includes any environmental information that can aid a robot in determining what it is sensing. Often, semantic SLAM has a segmentation step where observed data are labeled in a map based on the semantic information related to them (Chen et al., 2022) The data used to identify what should fall within the different map types vary on the application, which can include identifying an object based on shape outline, color, 3D model, size, etc. (Xia et al., 2020; Mahmoud and Atia, 2022) Research into quickly identifying and classifying semantic imagery information during SLAM operations is crucial to this method's success (Zhang et al., 2025; Zhang et al., 2024; Zhang et al., 2022). For SF-GraphSLAM, fiducials were selected to allow for quicker identification and pose estimation, and then the semantic relationships between the fiducials were identified.

2.3.2 Fiducial SLAM background

The aid of fiducials is often used to provide identification, pose, and orientation of a marker attached to a known position/orientation on an object (Fiala, 2010). Many types of fiducials are available via open-source software. They are commonly formed with black and white contrast with arrays of cells that can have either value to attribute a different identifier (Kostak and Slaby, 2021). They are often in the shape of a square for corner identification, but there are also circular (Lightbody et al.,

2017) and other variants. Fiducials can provide faster pose and orientation data than via SLAM (Pfrommer and Daniilidis, 2019) but are sensitive to problems such as variations in lighting, motion blur, and partial covering (Fiala, 2010). In addition, fiducials are often attached on a flat surface and are viewed best from particular angles. Their accuracy can be expressed as a function of relative camera distance and angle (Abawi et al., 2004). Fiducials can be used to augment SLAM, such as being placed around a building corridor being traversed and mapped to improve the estimation output (DeGol et al., 2018). For this experimentation, the AprilTag fiducial (Olson, 2011) was selected due to the vast amount of open-source resources for it and ease of integration into testing.

2.3.3 Filtering

The position and orientation of assembly components relative to robots can be determined with SLAM. Filtering or smoothing architectures can be utilized to allow for position and orientation determination and target model generation to be carried out simultaneously. Some popular filters include the Kalman filter, which can be applied to implement the Bayes estimator optimally when a system is linear (Kal, 2019), and its many derivations, such as the extended Kalman filter (EKF) and the unscented Kalman filter (UKF) (Chen, 2003).

2.3.4 SF-GraphSLAM's combination of state-of-the-art approaches and innovations for highly controlled applications, such as in-space assembly

The application of SLAM for in-space assembly is unique due to the controlled nature of the operations and the ability to have a large amount of prior knowledge. SLAM is often used to map unknown environments, and even with semantic SLAM, the prior knowledge is often generalized to common but not specific structures, such as identifying the general shape of a chair to maneuver around it. For in-space assembly, the structure is known beforehand, including the desired sequence of assembly steps, the module dimensions, the expected final structure, etc. This gives SF-GraphSLAM a unique opportunity to leverage this plethora of semantic information to better estimate the poses of the modules being assembled and ensure they are accurate compared to the ideal model before continuing assembly. This article will show how SF-GraphSLAM uses semantic knowledge of the module's kinematics, assembly tolerances, and degrees of freedom to enable the repeated verification of the structure's accuracy throughout its dynamic assembly. Additionally, SF-GraphSLAM reduces the difficulty of the estimations by leveraging fiducials and minimizes the effect of increased complexity as the size of the structure state vector increases by only using the minimum required fiducials to define modules. SF-GraphSLAM can leverage the highly controlled nature of in-space assembly and resulting semantic information to achieve higher accuracy pose estimations irrespective of introduced sensor and measurement errors. This article focuses on in-space assembly, but this SF-GraphSLAM approach could extend to other highly controlled applications where the structure and sub-modules are well known and manufactured to a high accuracy.

2.4 Built on-orbit robotically assembled gigatruss (BORG)

The "Built On-orbit Robotically Assembled Gigatruss (BORG): Mixed Assembly Architecture Trade Study" (Chapin, 2023) mixed assembly approach truss structure was used as the reference structure for this SF-GraphSLAM simulation. It comprises three types of modules: (1) deployable modules; (2) close-out strut; (3) close-out square. These modules are assembled in a checkerboard pattern to create structures of N × N × N dimensions. This analysis is completed on an example 3 × 3 × 3 BORG truss. The assembly, measurement, and correction process are shown in Figure 1a, and the modules and the assembled BORG truss are shown in Figure 1b.

2.5 Model derivation

2.5.1 Benefit of using a mixed assembly method with sparse fiducials

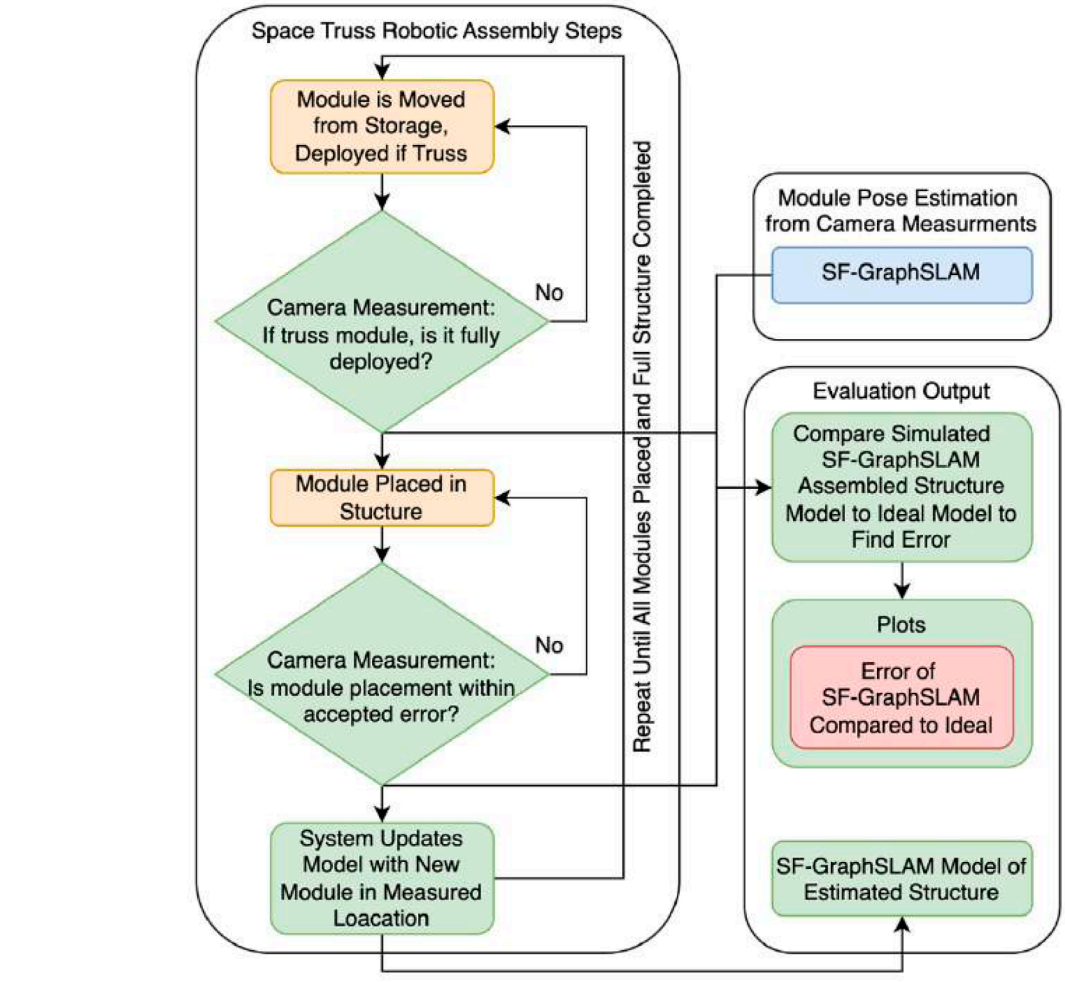
The SOA approach to solving vision for this application would be to assume all struts have six state variables and use either semantic SLAM or fiducials for each strut (Lynch and Park, 2017). The six state variables would include three states for Cartesian coordinates for position ($x_t^i\ y_t^i\ z_t^i$) and three states for angular orientation ($\Psi_t^i\ \theta_t^i\ \Phi_t^i$) to define each strut in the structure state vector, X_s shown in Equation 1, where $i = 1, 2, n$, and t is the time index.

$$X_s = \left[\ldots x_t^i\ y_t^i\ z_t^i\ \Psi_t^i\ \theta_t^i\ \Phi_t^i\ \ldots\right]' \tag{1}$$

For the state of the art, n would be the number of struts, which in our 3*x*3*x*3 truss would be 252, including diagonals. Therefore, the structure state vector would have 1,512 states. All these added modules will continue to be viewed as individual entities instead of a newly formed structure. This will further be added to the state vectors of the robots in the scene, X_r, to create the entire state vector, X, shown in Equation 2.

$$X = \left[X_r\ X_s\right]' \tag{2}$$

Figure 2a shows an example module that requires 12 measurements to define the pose of all the struts. Figure 2b shows the new approach using sparingly placed fiducials to reduce the number of measurements to only 2 to fully define the deployable module. Figure 2c shows that as the number of cells in an assembled example $n \times n \times n$ cube truss structure increases, the disparity of the number of markers needed for an SOA strut-by-strut approach, in black, is larger than the new approach, in pink. Both the number of measurement points (solid lines) and the overall structure state vectors (dotted lines) are plotted. The number of measurements required to define the structure is calculated with Equation 3 for the strut-by-strut assembly approach and with Equation 4 for the mixed assembly with sparsely placed fiducials. The breakdown of how these numbers of struts and modules are calculated is further explained by Chapin et al. (2023) when calculating the scalability of the mixed assembly method. The difference in this calculation is that a single measurement is calculated for each strut in the strut-by-strut approach, while the mixed assembly method calculates a



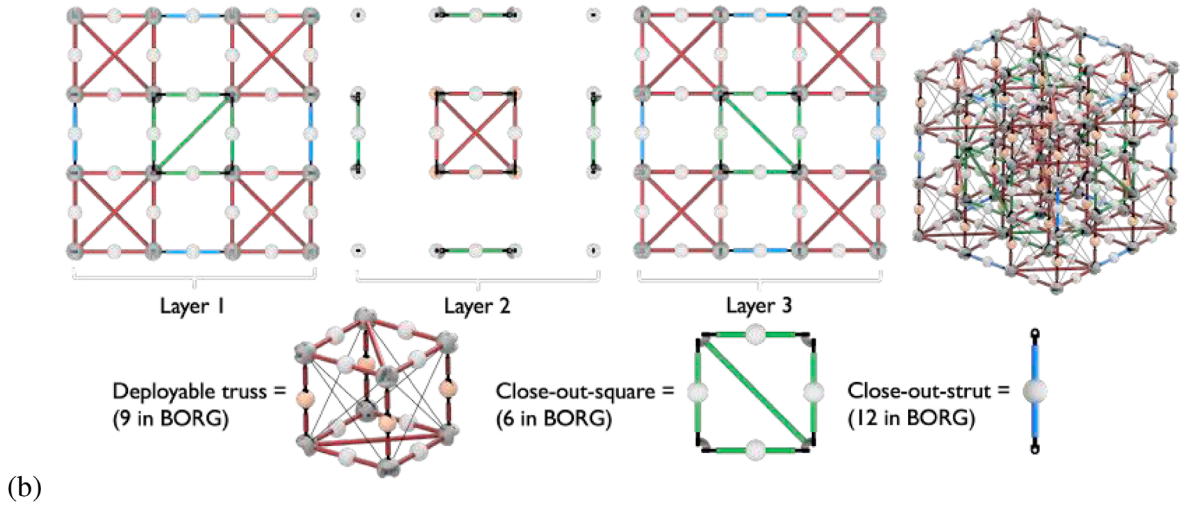


FIGURE 1
(a) Flowchart showing the use of SF-GraphSLAM in a robotic in-space assembly application. **(b)** Example of an in-space assembly truss structure on which SF-GraphSLAM will be tested to aid simulated assembly. The 3 × 3 × 3 BORG truss comprises three module types: deployables, close-out squares, and close-out struts.

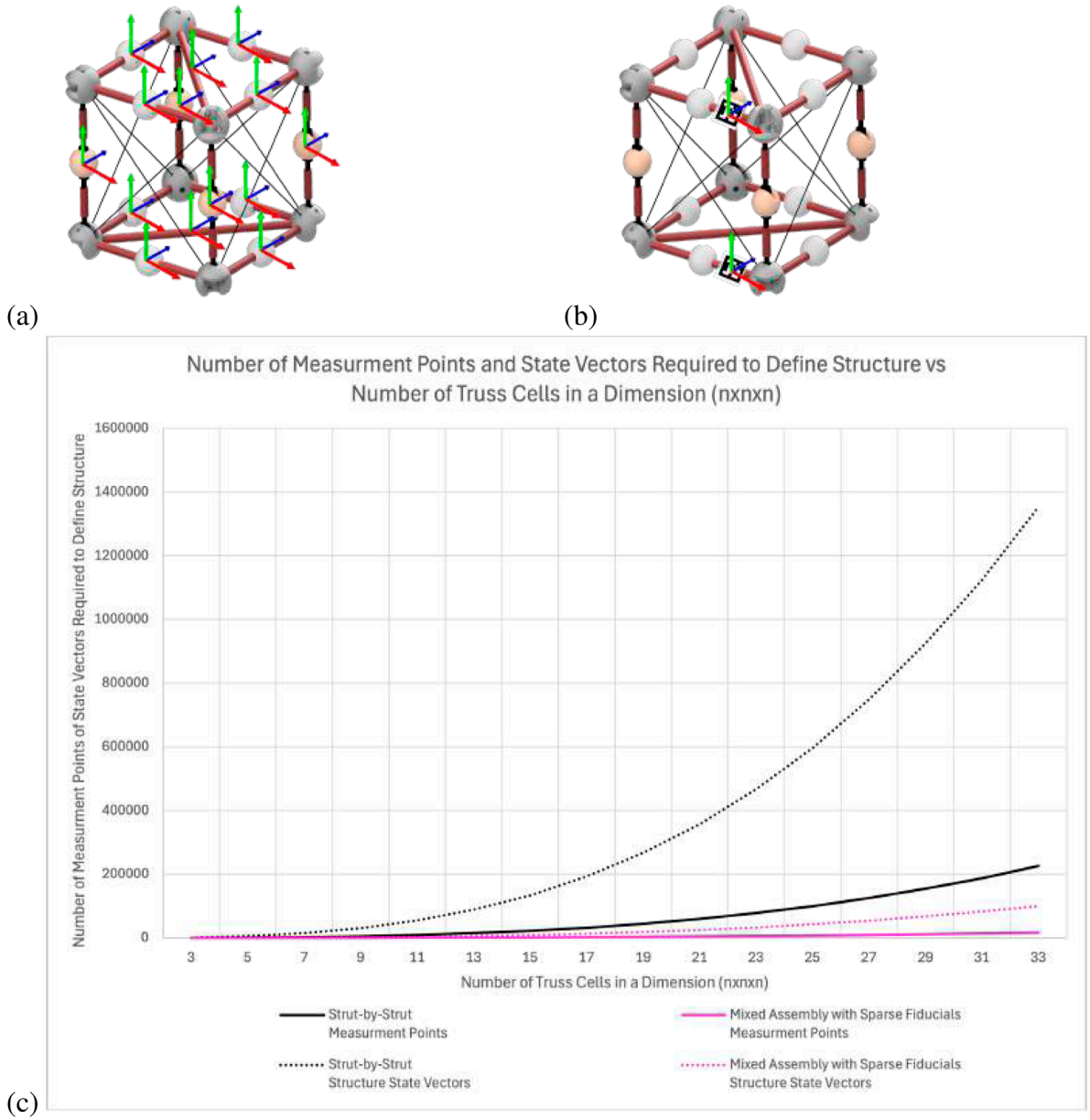


FIGURE 2
(a) State-of-the-art observation of each strut of the truss structure. **(b)** Proposed approach to simplify by observing sparse AprilTag fiducials (Olson, 2011) on the structure. **(c)** Comparing the scalability of the strut-by-strut and mixed assembly with sparse fiducial methods and how it affects the number of metrology measurements and resulting state vector size.

measurement for each close-out strut and close-out square and two measurements for the deployable modules.

$$M_{sbs} = 6n^3 + 9n^2 + 3n \tag{3}$$

$$M_{ma} = \frac{1}{2}\left(n^3 - 3n^2 + 21n - 23\right) \tag{4}$$

These two equations show that the number of measurements needed are $O(n^3)$ for an $n \times n \times n$ truss structure, only differing by a constant. That said, mathematical optimization algorithms are super-linear in the dimension of the problem, where the complexity depends on the linearity of the problem and the constraint types [Jamieson et al. (2012) give an example of a derivative-free optimization algorithm that is $O(n^{3/2})$ in state dimension]. Any reduction in the state dimension will result in a computational time reduction greater than the constant difference in the number of measurement points, which is highly beneficial in ISAM scenarios where time is critical and energy consumption must be limited.

Many types of VSLAM could be used as the SOA reference that do not account for structure-specific information. However, if the structure is treated as being composed of non-static agents, it loses the benefits associated with eliminating the static states from the filter. Overall, the prediction is that this SOA method will prove to be

very slow in handling the very large state vector resulting from all the struts of the large structure, in addition to difficulties dealing with so many dynamic elements due to the components being robotically assembled. The mixed assembly with sparse fiducials decreases the complexity of the analysis by minimizing the state vector. The complexity does increase as the number of fiducials and associated semantic relationships increases but is still less complex than the alternative SOA approach. In addition, if point-cloud mapping were used for this approach, even more measurement points would need to be utilized to be able to identify the module being observed, and it would not yield the benefit of identification that AprilTag fiducials can provide in addition to pose estimation.

2.5.2 Identifying the factor graph basis for SF-GraphSLAM

Factor graphs are used in offline SLAM problems, such as GraphSLAM (Thrun and Montemerlo, 2006), meaning the computation is completed after all robotic movements are done. This results in increased computation time because the entire robot operation is evaluated, but assuming the initial conditions are good, offline SLAM tends to be more accurate than online SLAM, which is active during robotic operation.

First, we will establish the notation that will be used throughout this article. The time index is labeled with t, and in SLAM, time is usually discrete. At time t, the robot pose is x_t. For the purposes of our system, we will let the robot pose and camera pose be equal to simplify the math. To show all the poses from time 1 to t, we will use $x_{1:t}$. The world is represented by the map, m, which is a set of landmarks m_j. For this application, the landmarks are described by Asri1 and Zhu (2025). We assume the map is time-invariant because our measurements will be completed after an assembly deployment or placement is completed, and the truss is in a static state.

Cameras are used in our application, and the main sensor measurement is the pose and orientation calculation of the AprilTag relative to the camera or robot. At time t, z_t represents the measurement. Because the robot must be able to have multiple AprilTags in view in any camera frame to better estimate their relationship to each other, each individual measurement can be specified as z_t^i. A measurement function, h, is used to describe how the measurement is generated in Equation 5 with added noise using a Gaussian random variable $\epsilon_t^i \sim \mathcal{N}(0, Q_t)$ and the map feature m_j measured at time t by the i-th measurement:

$$z_t^i = h\left(x_t, m_j, i\right) + \epsilon_t^i \tag{5}$$

For our application, the camera will measure the AprilTags' relative positions and orientations with respect to the camera's or robots' position. The AprilTag represents a single 6-degree-of-freedom reference point for the truss structure it is attached to, and therefore, its location relative to the camera can be expressed with the position $(p1_t^i,\ p2_t^i,\ p3_t^i)$ and Euler angles orientation $(r1_t^i,\ r2_t^i,\ r3_t^i)$. This is shown in Equation 6, in the form of the $i-th$ AprilTag's measurement pose with respect to the camera at time t. This measurement is generated by running an AprilTag detection algorithm on the saved camera video, and the AprilTag number is used to identify the map feature m_j being measured.

$$z_t^i = \left[p1_t^i,\ p2_t^i,\ p3_t^i,\ r1_t^i,\ r2_t^i,\ r3_t^i\right] \tag{6}$$

Equation 5 suggests a multivariate Gaussian distribution, with Q_t representing the zero mean and covariance, the logarithm of which is as follows in Equation 7:

$$\log p\left(z_t^i | x_t, m\right) = const.exp - \frac{1}{2}\left(z_t^i - h\left(x_t, m_j, i\right)\right)^T Q_t^{-1}\left(z_t^i - h\left(x_t, m_j, i\right)\right) \tag{7}$$

Because the robot is changing its pose as it is taking measurements, the control commands of the robot between time intervals $t-1$ and t can be represented by u_t. The state transition of robot poses, Equation 8, is controlled by the function g, the kinematic model of the robot, where the model command noise is modeled by $\delta_t \sim \mathcal{N}(0, R_t)$ [Equation 4 from Thrun and Montemerlo (2006)]:

$$x_t = g\left(u_t, x_{t-1}\right) + \delta_t \tag{8}$$

Similar to the h function, the g function for our application is simply the position and orientation of the robot with respect to the previous position. This can be calculated by applying the known robot control u_t to the last known robot position, x_{t-1}, to calculate the camera position at time t, x_t. This can also be represented by a 6-degree-of-freedom (DOF) pose, shown in Equation 9. For simulation, we are simplifying the scenario for the camera to be representative of the robot and assuming we know its motion from measurement to measurement. In testing, this can be modified to incorporate the actual kinematics of the robot performing the camera measurements, in this case, a Stewart platform, and it can also be compared against an external global metrology system with markers on the robot.

$$x_t = \left[p1_t,\ p2_t,\ p3_t,\ r1_t,\ r2_t,\ r3_t\right] \tag{9}$$

Equation 8 can be used to determine the state transition probability, as shown in Equation 10:

$$\log p\left(x_t | u_t, x_{t-1}\right) = const.exp - \frac{1}{2}\left(x_t - g(u_t, x_{t-1})\right)^T R_t^{-1}\left(x_t - g(u_t, x_{t-1})\right) \tag{10}$$

Equation 11 shows the posterior probability over the map m and robot path x_{1-t} to create the offline SLAM posterior. Note the probability is not only at a single pose, x_t, but over the full robot path, $x_{1:t}$.

$$p\left(x_{1:t}, m | z_{1:t}, u_{1:t}\right) \tag{11}$$

An additional element must be added for SF-GraphSLAM: a semantic relationship between two map measurements to compare against an expected relationship between them in the ideal map. For this, the measurement of the k-th AprilTag map feature at time t, $a_t^{i,k}$, can be estimated by measuring a related feature seen at the same time step, z_t^i, and applying the expected relation between the two map features, $r_{i,k}$. This is shown in Equation 12. Similar to the measurement function, noise can be accounted for using a Gaussian random variable $\epsilon_t^i \sim \mathcal{N}(0, S_t)$. The form of $a_t^{i,k}$ is the same 6-DOF AprilTag measurement as z_t^i, only for another AprilTag, but the lettering is distinguished to make the derivation easier to follow.

$$a_t^{i,k} = s\left(z_t^i, r_{i,k}\right) + \epsilon_t^i \tag{12}$$

AprilTag location comparisons can only be made locally due to visibility limitations and slices of time, so in practice, the number of $a_t^{i,k}$ comparisons that can be made will be a constant factor of the number of AprilTags, instead of the worst-case scenario of total time slices × total possible AprilTag pairs.

How this additional relation is derived for the full SLAM posterior is further explained below.

2.5.3 Deriving the full posterior for SF-GraphSLAM

Let Equation 13 be the state variable, y, as the concatenation of all the camera poses, x, from time 0 to t, robot path $x_{0:t}$, and the map, m. A momentary state y_t can be defined with robot position and the map:

$$y_{0:t} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_t \\ m \end{pmatrix} \text{ and } y_t = \begin{pmatrix} x_t \\ m \end{pmatrix} \tag{13}$$

In a traditional SLAM problem, the posterior can be defined by Equation 14 as an implementation of Bayes' theorem, where the familiar normalizer is represented by η, the controls are $u_{1:t}$, and the familiar measurements are $z_{1:t}$ with correspondences $c_{1:t}$.

$$p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}) = \eta p(z_t|y_{0:t}, z_{1:t-1}, u_{1:t}, c_{1:t}) p(y_{0:t}|z_{1:t-1}, u_{1:t}, c_{1:t}) \tag{14}$$

For SF-GraphSLAM, we modify the full SLAM posterior by adding an additional semantic step, as shown in Equation 15.

$$\begin{aligned} p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}, a_{1:t}, r_{1:t}) &= \eta p(z_t|y_{0:t}, z_{1:t-1}, u_{1:t}, c_{1:t}, a_{1:t}, r_{1:t}) \\ &\times p(a_t|z_{1:t-1}, u_{1:t}, c_{1:t}, a_{1:t}, r_{1:t}) \\ &\times p(y_{0:t}|z_{1:t-1}, u_{1:t}, c_{1:t}, a_{1:t}, r_{1:t}) \end{aligned} \tag{15}$$

Due to the Markov property, measurements only depend on the current location of the sensing agents and the environment. The current agent state only depends on the previous state, and the positional relationship between two tags only depends on the measurement linking them and the knowledge of the structure. Thus, the posterior can be simplified:

$$\begin{aligned} &p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}, a_{1:t}, r_{1:t}) \\ &= \eta p(y_0) \prod_t p(x_t|x_{t-1}, u_t) \prod_i \prod_t \\ &\times p\left(z_t^i|y_t, c_t^i\right) \prod_i \prod_k \prod_t p\left(a_t^{i,k}|z_t^i, r_{i,k}\right) \end{aligned} \tag{16}$$

The prior $p(y_0)$ can be factored into $p(x_0)$ and $p(m)$. Normally, SLAM does not have prior map m knowledge, but there is prior knowledge in the SF-GraphSLAM case. Therefore, the factor $p(m)$ cannot be subsumed into the normalizer η and must be taken into account along with $p(x_0)$ within $p(y_0)$. Again, z_t^i is the i-th measurement taken at time t. This nomenclature is carried over for the relation of the k-th map element at time t for the semantic information, $a_t^{i,k}$.

Logarithmic form can be used to represent the probabilities in information form. Equation 17 shows the log-SF-GraphSLAM posterior.

$$\begin{aligned} &\log p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}, a_{1:t}, r_{1:t}) \\ &= const. + \log p(x_0) \\ &+ \sum_i \sum_t \log p(x_t|x_{t-1}, u_t) + \sum_t \log p\left(z_t^i|y_t, c_t^i\right) \\ &+ \sum_i \sum_k \sum_t \log p\left(a_t^{i,k}|z_t^i, r_{i,k}\right) \end{aligned} \tag{17}$$

The sum of terms is the simple form of this posterior. This includes a prior for control u_t and measurement z_t^i.

Next, the measurement, motion, and semantic models can be approximated using linear functions with error distributions that are Gaussian. The deterministic motion function, g, and a motion error covariance, R_t, can be used to create a normally distributed robot motion of $N(g(u_t, x_{t-1}), R_t)$. Similarly, $N(h(y_t, c_t^i), Q_t)$ is used to generate measurements z_t^i using the measurement function h and the covariance error Q_t. Semantic information uses a similar $N(s(a_t^{i,k}, r_{i,k}), S_t)$ function with a semantic function s and covariance matrix S_t. These equations are shown in Equation 18.

$$\begin{aligned} p(x_t|x_{t-1}, u_t) &= \eta \exp\left\{-\frac{1}{2}(x_t - g(u_t, x_{t-1}))^T R_t^{-1} (x_t - g(u_t, x_{t-1}))\right\} \\ p\left(z_t^i|y_t, c_t^i\right) &= \eta \exp\left\{-\frac{1}{2}\left(z_t^i - h\left(y_t, c_t^i\right)\right)^T Q_t^{-1} \left(z_t^i - h\left(y_t, c_t^i\right)\right)\right\} \\ p\left(a_t^{i,k}|z_t^i, r_{i,k}\right) &= \eta \exp\left\{-\frac{1}{2}\left(a_t^{i,k} - s\left(z_t^i, r_{i,k}\right)\right)^T S_t^{-1} \left(a_t^{i,k} - s\left(z_t^i, r_{i,k}\right)\right)\right\} \end{aligned} \tag{18}$$

The prior, $p(x_0)$, sets x_0, the initial pose, to the global coordinate system's origin $x_0 = (0\ 0\ 0)^T$. The prior can be expressed as a Gaussian-type distribution, shown in Equation 19.

$$p(x_0) = \eta \exp\left\{-\frac{1}{2} x_0^T \Omega_0^{-1} x_0\right\} \tag{19}$$

Ω_0 is shown in Equation 20. The value of ∞ can be substituted by a very large positive number to make the posterior equivalent to a likelihood.

$$\Omega_0 = \begin{bmatrix} \infty & 0 & 0 \\ 0 & \infty & 0 \\ 0 & 0 & \infty \end{bmatrix} \tag{20}$$

This can be used to create the quadratic form of the log-SF-GraphSLAM posterior, shown in Equation 21. This information form of the full SLAM posterior is composed of quadratic terms for the prior, controls, measurements, and semantic relations.

$$\begin{aligned} &\log p(y_{0:t}|z_{1:t}, u_{1:t}, c_{1:t}, a_{1:t}, r_{1:t}) \\ &= const. - \frac{1}{2}\Big[x_0^T \Omega_0^{-1} x_0 \\ &+ \sum_t (x_t - g(u_t, x_{t-1}))^T R_t^{-1} (x_t - g(u_t, x_{t-1})) \\ &+ \sum_i \sum_t \left(z_t^i - h\left(y_t, c_t^i\right)\right)^T Q_t^{-1} \left(z_t^i - h\left(y_t, c_t^i\right)\right) \\ &+ \sum_i \sum_k \sum_t \left(a_t^{i,k} - s\left(z_t^i, r_{i,k}\right)\right)^T S_t^{-1} \left(a_t^{i,k} - s\left(z_t^i, r_{i,k}\right)\right)\Big] \end{aligned} \tag{21}$$

2.5.4 Factor graph formulation

Equation 16 can also be restated in terms of factors in the factor graph formulation, where each conditional and prior probability has an associated factor ϕ, as shown in Equation 22:

$$\begin{aligned}&\phi(y_{0:t}, z_{1:t}, u_{1:t}, c_{1:t}, a_{1:t}, r_{1:t})\\&= \phi_p(y_0) \times \prod_t \phi_g(x_t, x_{t-1}, u_t) \prod_i \prod_t \phi_h\\&\times (z_t^i, y_t, c_t^i) \prod_i \prod_k \prod_t \phi_s(a_t^{i,k}, z_t^i, r_{i,k})\end{aligned} \tag{22}$$

The factors ϕ are generalizations of the Gaussian probability distributions, which eliminate the normalizing constant, and do not change the maximum a posteriori estimate, as shown in Equation 23:

$$\begin{aligned}\phi_p(x_0) &= \frac{1}{\eta} p(x_0)\\ \phi_g(x_t, x_{t-1}, u_t) &= \frac{1}{\eta} p(x_t | x_{t-1}, u_t)\\ \phi_h(z_t^i, y_t, c_t^i) &= \frac{1}{\eta} p(z_t^i | y_t, c_t^i)\\ \phi_s(a_t^{i,k}, z_t^i, r_{i,k}) &= \frac{1}{\eta} p(a_t^{i,k} | z_t^i, r_{i,k})\end{aligned} \tag{23}$$

2.5.5 GraphSLAM graph, information matrix, and summation function extended to SF-GraphSLAM

The goal of the factor graph formulation is to minimize the maximum a posteriori estimate. Our implementation of the SF-GraphSLAM algorithm modifies the GraphSLAM algorithm described by Thrun and Montemerlo (2006). To illustrate the general structure of the algorithm, see Figure 3. This graph shows we have two map features, m_1 and m_2, and three robot poses, x_1, x_2, and x_3. In our case, the map features are AprilTags, and the robot pose also represents that camera pose. There are two types of lines in this diagram: (1) motion lines and (2) measurement lines. Motion lines link consecutive robot poses, while measurement lines link to the map features visible for each measurement. This shows an example measurement cycle between assembly steps where the robot will move the camera to view two AprilTags within the same camera frame, either on a single deployable or from two adjacent modules in the larger structure, to allow for measurement of their relative positioning.

The GraphSLAM_initialize algorithm starts by initializing the mean pose vector, $u_{1:t}$. Each edge is a nonlinear constraint that represents the negative log likelihood of the motion and measurement models. A nonlinear least squares problem results from the sum of the constraints. GraphSLAM linearizes these sets of constraints in order to compute the map posterior. This GraphSLAM_linearize algorithm creates an information vector and a sparse information matrix. The sparseness allows the GraphSLAM_reduce algorithm to apply variable elimination to result in a smaller graph only defined by robot poses. The path posterior is updated using the GraphSLAM_solve algorithm using standard interference techniques. The GraphSLAM_known_correspondence algorithm combines all these previous algorithms to return the best guess of the map, the robot's path, and the mean μ. Note that the full map posterior is not usually recovered because it is quadratic with respect to the size of the map. Therefore, GraphSLAM normally only computes some marginal posteriors over the map and the map itself.

2.5.6 GraphSLAM: building the graph

If we take a set of measurements $z_{1:t}$, correspondence variables $c_{1:t}$, and controls $u_{1:t}$, GraphSLAM can build a graph with these data. As seen in Figure 3, the map features $m = m_j$ and the robot poses $x_{1:t}$ are graph nodes. Edges and lines are events due to the motion of the robot, solid lines connect robot poses or measurements, and dotted lines connect the robot pose and measurements taken with respect to the visible map features. These edges are soft constraints between the features and poses in GraphSLAM.

If a system is linear, the constraints can be directly input into the information matrix, Ω, and the information vector, ξ, of a system of equations. Each control and measurement locally updates Ω and ξ, and results in adding an edge to the GraphSLAM graph. Figure 4 shows the process of creating the graph step by step and updating the information matrix. The measurement z_t^i gives us information at time t between the robot pose x_t and the feature location $j = c_t^i$. This maps to the constraint between m_j and x_t in GraphSLAM. This edge can also be thought of like a spring-mass model's "spring." The measurement constraint can be formulated as shown in Equation 24:

$$\left(z_t^i - h\left(x_t, m_j, i\right)\right)^T Q_t^{-1} \left(z_t^i - h\left(x_t, m_j, i\right)\right) \tag{24}$$

Q_t is the measurement noise covariance, while h is the measurement function. An example of this measurement constraint being added is shown in Figure 4a, with the resulting updating of the GraphSLAM graph on the left and the information matrix on the right.

Pose constraints are added to the information matrix and vector in information form by adding values between the grid rows and columns between consecutive robot poses x_{t-1} and x_t. In this case, the motion model has uncertainty covariance R_t, and the magnitude corresponds to the constraint stiffness. This is shown in Figure 4b, where the motion from robot pose x_0 to x_1 is updated in the information matrix. For this robot motion, the control u_t gives information about the pose from time $t-1$ relative to t. This creates the pose constraint shown below in Equation 25:

$$(x_t - g(u_t, x_{t-1}))^T R_t^{-1} (x_t - g(u_t, x_{t-1}) \tag{25}$$

Above R_t is the motion noise covariance, and g is the robot's kinematic motion model. This is shown in Figure 4b between $t = 0$ and $t = 1$ and updates the information matrix between measurement z_t^i and pose x_t. Because this is additive, the less noisy the sensor is, the higher magnitude will be added to the information matrix and vector because it reflects R_t, the residual uncertainty, of the measurement noise.

Finally, once all the soft constraints are collected from the completed controls $u_{1:t}$ and measurements $z_{1:t}$, shown in Figure 4c, they can be incorporated into the graph. This graph is sparse because the number of constraints is linear within the elapsed time. A function $J_{GraphSLAM}$ can be formed by summing all the graph constraints, shown in Equation 26:

$$\begin{aligned}J_{GraphSLAM} &= x_0^T \Omega_0^{-1} x_0 + \sum_t (x_t - g(u_t, x_{t-1}))^T R_t^{-1} (x_t - g(u_t, x_{t-1}))\\&+ \sum_t \sum_i \left(z_t^i - h\left(y_t, c_t^i, i\right)\right)^T Q_t^{-1} \left(z_t^i - h\left(y_t, c_t^i, i\right)\right)\end{aligned} \tag{26}$$

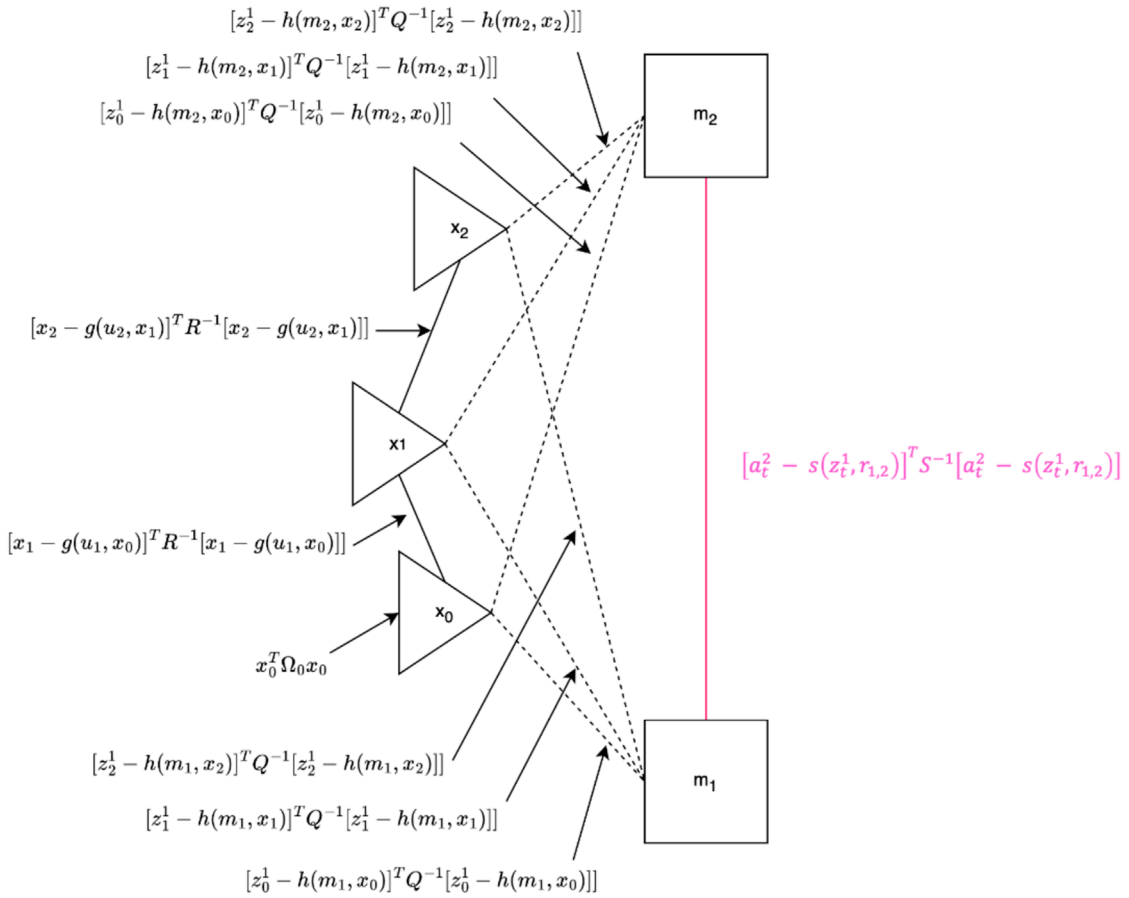


FIGURE 3
SF-GraphSLAM diagram. Black elements represent the existing map based on GraphSLAM, and pink elements represent the additional information utilized in SF-GraphSLAM. There are three robot poses. These also represent the camera poses and two map features, which represent the AprilTag markers. Solid black lines indicate motion between consecutive robot poses, while dashed black lines represent measurements from those robot poses of the map elements in view of the camera. The pink solid line and equation represent the additional semantic information of the desired transformation matrix between the two AprilTag markers. For best results, it is ideal to have at least two AprilTags visible to the camera at any given pose.

This function is defined over all the map m features and poses $x_{1:t}$. The function starts with the anchoring constraint, $x_0^T \Omega_0^{-1} x_0$, which initializes the first robot pose as $(0\ 0\ 0)^T$, therefore constraining the absolute coordinates of the map.

The information matrix Ω is populated with zeros for all the off-diagonal elements except for where either a measurement or pose link was created, between two consecutive poses or between a map element observed at a given pose, respectively. The Ω is sparse with all elements being zero, including between pairs of different features, except for a linear number of constraints generated from the graph. The SLAM measurements only constrain the map features relative to the robot pose, but we never collect information about the features relative to each other.

2.5.7 SF-GraphSLAM: incorporating semantic information into the graph

The SF-GraphSLAM approach builds off of GraphSLAM and adds additional semantic components to the $J_{GraphSLAM}$ function to create $J_{SF-GraphSLAM}$, shown in Equation 27:

$$\begin{aligned} J_{SF-GraphSLAM} = {} & x_0^T \Omega_0^{-1} x_0 + \sum_t \left(x_t - g\left(u_t, x_{t-1}\right)\right)^T R_t^{-1} \left(x_t - g\left(u_t, x_{t-1}\right)\right) \\ & + \sum_t \sum_i \left(z_t^i - h\left(y_t, c_t^i, i\right)\right)^T Q_t^{-1} \left(z_t^i - h\left(y_t, c_t^i, i\right)\right) \\ & + \sum_t \sum_k \sum_i \left(a_t^{i,k} - s\left(z_t^i, r_{i,k}\right)\right)^T S_t^{-1} \left(a_t^{i,k} - s\left(z_t^i, r_{i,k}\right)\right) \end{aligned} \tag{27}$$

This is equivalent to the double negative log of the product of factors, as shown in Equation 28:

$$-2 \log \phi\left(y_{0:t}, z_{1:t}, u_{1:t}, c_{1:t}, a_{1:t}, r_{1:t}\right) = J_{SF-GraphSLAM} \tag{28}$$

Here, $a_t^{i,k}$ and z_t^i are the candidate poses for the two AprilTag markers connected by a single deployable or assembly relation. For all measurement times, t, the number of AprilTags detected from the camera, k, is compared against every other observable tag, i, if they are connected by a single relation. For example, $a_t^{i,k}$ could be on the

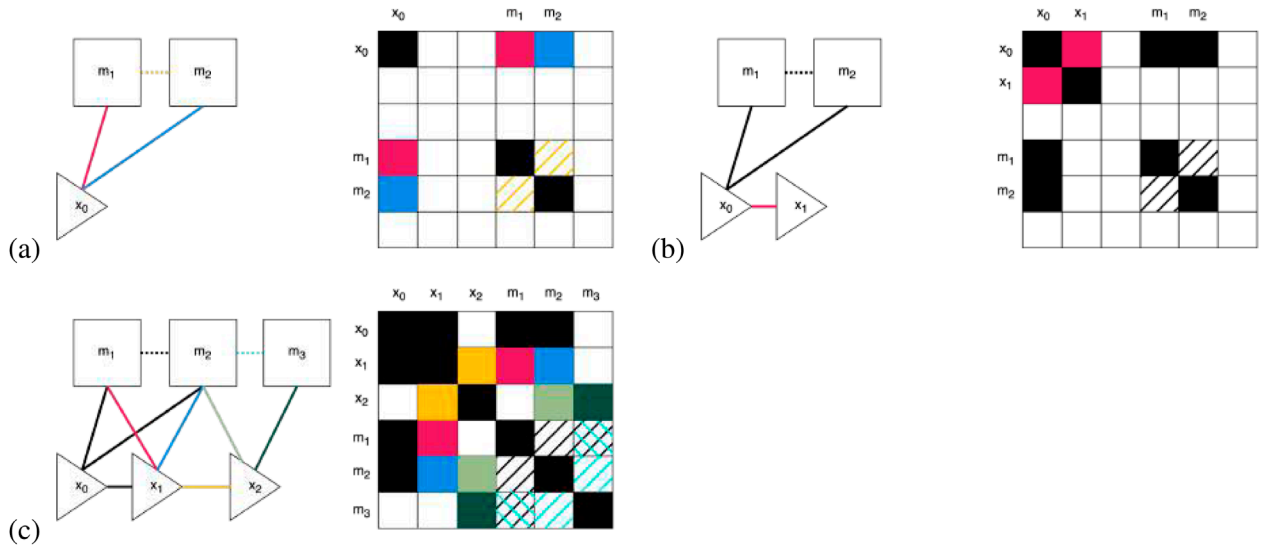


FIGURE 4
Illustration of how the information matrix, on the right, gets built out in SF-GraphSLAM using our example with three robot poses each viewing two map features shown on the left dependence graphs. Note that the information in solid lines/boxes represents what GraphSLAM already establishes, while the dotted lines and hashed boxes represent the added semantic information SF-GraphSLAM utilizes. **(a)** Observation at $t = 0$ of both AprilTags. Additional semantic information of the desired relative pose of two AprilTags. **(b)** Robot moves from x_0 to x_1. **(c)** The robot completes its motion and measurements at each time step, incorporates new semantic information of adjacent AprilTags, and updates the information matrix. Note that using the two sets of semantic knowledge between m_1 and m_2 and m_2 and m_3 also allows us to add to the information matrix between m_1 and m_3. Note that to generate an information matrix with enough overlapping features, the camera must be able to view adjacent markers in at least one robot pose.

bottom plane of a deployable truss, and z_t^i could be on the top plane. Additionally, they could represent markers of adjacent modules within the larger assembled BORG structure. In either case, based on either knowledge of the module structure, in the deployable case, or knowledge of the assembled structure, in the BORG truss case, there is semantic information known about what the desired relative poses of these AprilTags are and what the expected error should be, based on the physical deployment and assembly constraints. The lowercase $s(z_t^i, r_{i,k})$ function is used to determine where the marker m_j should be with respect to m_k based on their relation. The error between the candidate $a_t^{i,k}$ and where the model predicts it should be, s, is represented by $a_t^{i,k} - s(z_t^i, r_{i,k})$. Uppercase function S is the covariance matrix.

The covariance matrix S can be described by Equation 29:

$$S_t = \begin{bmatrix} \sigma_x & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_y & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_z & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{r_1} & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{r_2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_{r_3} \end{bmatrix} \tag{29}$$

2.5.8 Ideal relation for deployable and assembled modules

The following explains how the ideal s semantic function relation is determined using the deployable module as an example. The truss is first stowed in a compressed state where the AprilTags are closer together, 0.1575 m, and then, when deployed, they should ideally be 0.5 m apart. Throughout, the deployment path of the second AprilTag is constricted by the physical constraints of the deployable module.

We considered adding time indices to the map marker features m in order to account for the different states of the module, such as stowed or deployed. For simplicity, we decided to conduct the SF-GraphSLAM after each deployment step or assembly step is fully complete to remove the need to add the additional time element because GraphSLAM maps are time-invariant.

Therefore, we focus on the semantic information we know about the two AprilTags only in their fully deployed state. We know that if the deployment was successful, we would expect the transformation matrix between m_1 and m_2 to be what is shown in Equation 30, where there is a perfect 0.5 m transform along the z-axis and no other positional or rotational differences.

$$r_{1,2} = [0,\ 0,\ 0.5,\ 0,\ 0,\ 0] \tag{30}$$

Similar transforms can be specified for all adjacent AprilTags because the order of assembly steps is known, the final desired location of modules within the structure is known, and their desired relative positions are known. This is why being able to use AprilTags that also have identifying numbers is crucial to be able to properly keep track of which modules are being measured to query the desired relationships of AprilTags as they are being viewed by the camera. This ideal relationship is one reference, but the next section describes more specific relations for the deployable, close-out strut, and close-out square mechanism and assembly relationships.

2.5.9 Flexible relationship based on deployable kinematics

We must consider the kinematic model of the deployable to be able to compare the AprilTag measured positions with all possible deployable states, including stowed and partially and fully deployed.

Grübler's formula [Lynch and Park, (2017); Equation 2.4], shown in Equation 31, can be used to calculate the degrees of freedom (DOF) of the deployable truss because it is a mechanism based on joints and links. If we look at half of the deployable truss, it can be characterized as an 8-bar linkage. The number of links, N, is eight, including the ground, which in the case of the deployable truss is the bottom strut. The number of joints, J, is also eight, and each joint has one degree of freedom, f_i because they are all revolute joints. Finally, the DOF of the rigid body, m, in this case, is 3 because it is a planar mechanism. Therefore, the DOF of the deployable truss is 5.

$$dof = m(N-1-J) + \sum_{i=1}^{J} f_i = 3(8-1-8) + \sum_{i=1}^{8} 1 = 5 \tag{31}$$

If all joints are cylindrical with an f of 2, and we consider it a spatial mechanism with an m of 6, the equation calculates the DOF to be 10.

$$dof = m(N-1-J) + \sum_{i=1}^{J} f_i = 6(8-1-8) + \sum_{i=1}^{8} 2 = 10 \tag{32}$$

While Equation 32 is more accurate to the error possible in real-world hardware due to wiggle in the rotational shafts, the equation result of 5 DOF is used to simplify the analysis.

The s function can incorporate the kinematic model of whatever deployable truss is in use. For the example, we can define the corners of the truss using A, B, D, and E, respectively, and the mid nodes, C and F. There are measurable thetas between each deployable strut and the bottom and top, respectively, θ_1, θ_2, θ_3, and θ_4. θ_5 could represent any of the opposite side's angles. This is shown in Figures 5a,b.

The following set of equations in Equation 33 represents the locations of the nodes based on the kinematic model. This is set up similarly to another deployable structure described by Qi et al. (2016).

$$\begin{cases} A = (0 \quad 0 \quad 0) \\ B = (0 \quad 0 \quad -L_{strut}) \\ C = (0 \quad L_{halfstrut}\sin(\theta_2) \quad L_{halfstrut}\cos(\theta_2) - L_{strut}) \\ D = (0 \quad L_{halfstrut}(\sin(\theta_2) + \sin(\theta_3)) \quad L_{halfstrut}(\cos(\theta_2) + \cos(\theta_3)) - L_{strut}) \\ E = (0 \quad L_{halfstrut}(\sin(\theta_1) + \sin(\theta_4)) \quad -L_{halfstrut}(\cos(\theta_1) + \cos(\theta_4))) \\ F = (0 \quad L_{halfstrut}\sin(\theta_1) \quad -L_{halfstrut}\cos(\theta_1)) \end{cases} \tag{33}$$

From measuring the positions of the lower AprilTag, AT1, and the upper AprilTag, m_1, we can get a transform for m_2 relative to m_1, T_{m_1,m_2}. Therefore, we can focus on node E's position because it is adjacent to m_2 while node A is adjacent to m_1, which we can make the origin of the local coordinate system to analyze T_{m_1,m_2}. There is a minimum and maximum allowable angle of 4.69° and 90° respectively, for all θs based on the minimum and maximum height of the deployable truss. Therefore, in the algorithm to check whether the T_{m_1,m_2} is a valid configuration, we can solve the equations in Equation 33 and see whether they reach valid θ values. We can focus on checking node E's validity, and to simplify the equations, we can assume that $\theta_1 = \theta_4$ because a possible valid configuration is their being equal. We do not need to determine the exact intermediate state; we only need to determine whether the final measurement is valid.

2.5.10 Relation for assembled close-out struts

An image of a close-out strut fully inserted, along with the geometry of the node's interface, is shown in Figures 5c,d. In both cases, the strut is inserted into capturing features on two adjacent nodes. This information about the example truss node and strut geometry can be used to create bounds for whether the strut is considered "captured," within the physical geometric bounds of the node, or "final assembly," when the ball plungers internal to the node deploy into the strut end hole feature to constrain the DOF of the strut. The relation condition zones for "captured" and "fully assembled" are outlined in Figure 5g.

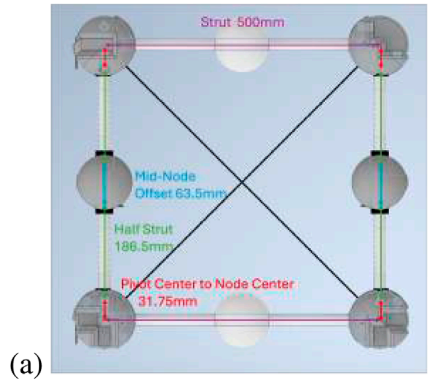
2.5.11 Relation for assembled close-out squares

The assembled close-out square relation is similar to that for the strut. Figure 5e shows an example of the close-out square inserted between the top four deployable corners, as well as a (Figure 5f) top view of the node geometry that interfaces with the close-out square. This information about the example truss node and close-out square geometry can be used to create bounds for whether the close-out square is considered "captured" or "final assembly," as shown in Figure 5g.

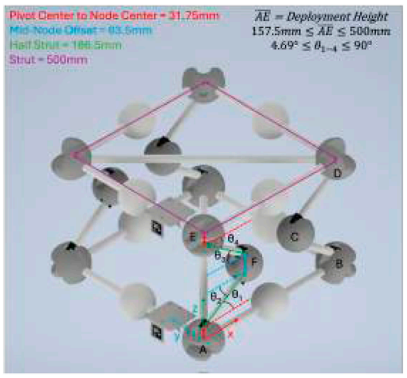
2.6 Generating BORG truss ideal model

The BORG truss can be simplified to four nodes along each axis connected by 0.5 m struts. For easy transition from simulation to hardware testing, each module in the 3 × 3 × 3 BORG truss example was given an identification number with respect to the order of assembly. Figure 6a below shows this module numbering scheme. In addition, each module has unique AprilTag identification, two tags for deployables, and a single tag for close-out struts and close-out squares. Those are also numbered in ascending order, shown for the four sides of the truss in Figures 6b–e. Each side has the AprilTags grouped on the right edge in order for easy camera panning for measurements. There is also a turntable, which has 12 tags spaced around to help connect the grouped edges of AprilTags.

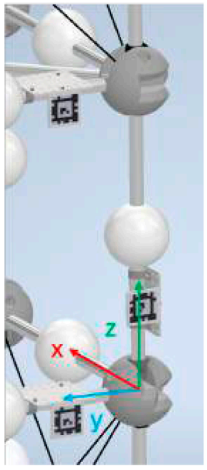
The offset of the AprilTag to the rightmost adjacent node center is uniform, excluding the special case of the vertical close-out struts, which have the same transform simply rotated around the x-axis. In addition, all AprilTags are positioned facing outward from the face they are on, which adjusts the local transform within the global coordinate frame, but each AprilTag maintains the same coordinate frame of x-axis to the right, y-axis up, and z-axis pointing outward from the truss face. Table 1 lists the AprilTag numbers, their respective module, and the location of the node that the AprilTag is adjacent to for the case of the ideal BORG truss structure, where each node is 0.5 m away in each direction.



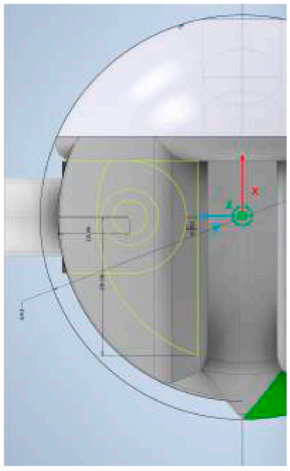
(a)



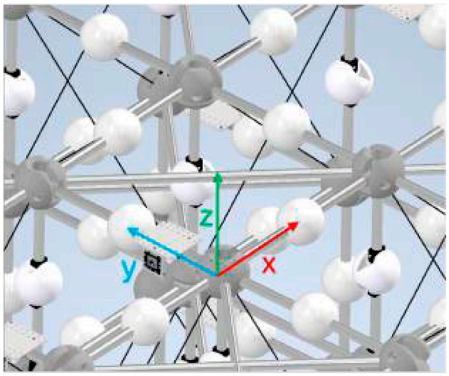
(b)



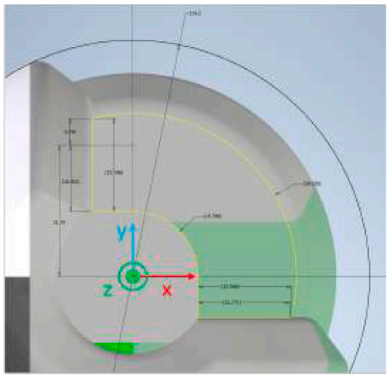
(c)



(d)



(e)



(f)

(g)

Module	Zone	Limit	x (m)	y (m)	z (m)	rx (deg)	ry (deg)	rz (deg)
Close-out-Strut	Capture	Min	-39.138	-3.302	-0.127	-0.4584	-0.4584	-90
		Max	0	20.24	0.127	0.4584	0.4584	0
	Final Assembly	Min	-0.679	-0.679	-0.127	-0.4584	-0.4584	Single Side: -90, If Both: -0.4584
		Max	0	0	0.127	0.4584	0.4584	Single Side/If Both: +0
Close-out-Square	Capture	Min	-16.002	-16.002	0	0	0	0
		Max	6.546	6.546	37.972	0	0	0
	Final Assembly	Min	0	0	-0.127	0	0	0
		Max	0.679	0.679	0.679	0	0	0

FIGURE 5
(a) Deployable truss side view with lengths of main elements labeled. **(b)** Deployable truss kinematic model with nodes and angles labeled. **(c)** A close-out strut is inserted in the vertical orientation. **(d)** Node geometry of the close-out strut to calculate capture and final assembly zones. **(e)** A close-out square is inserted in the horizontal orientation. **(f)** Top view of node close-out square geometry to calculate capture and final assembly zones. **(g)** Close-out strut and close-out square capture vs. assembly zone definitions.

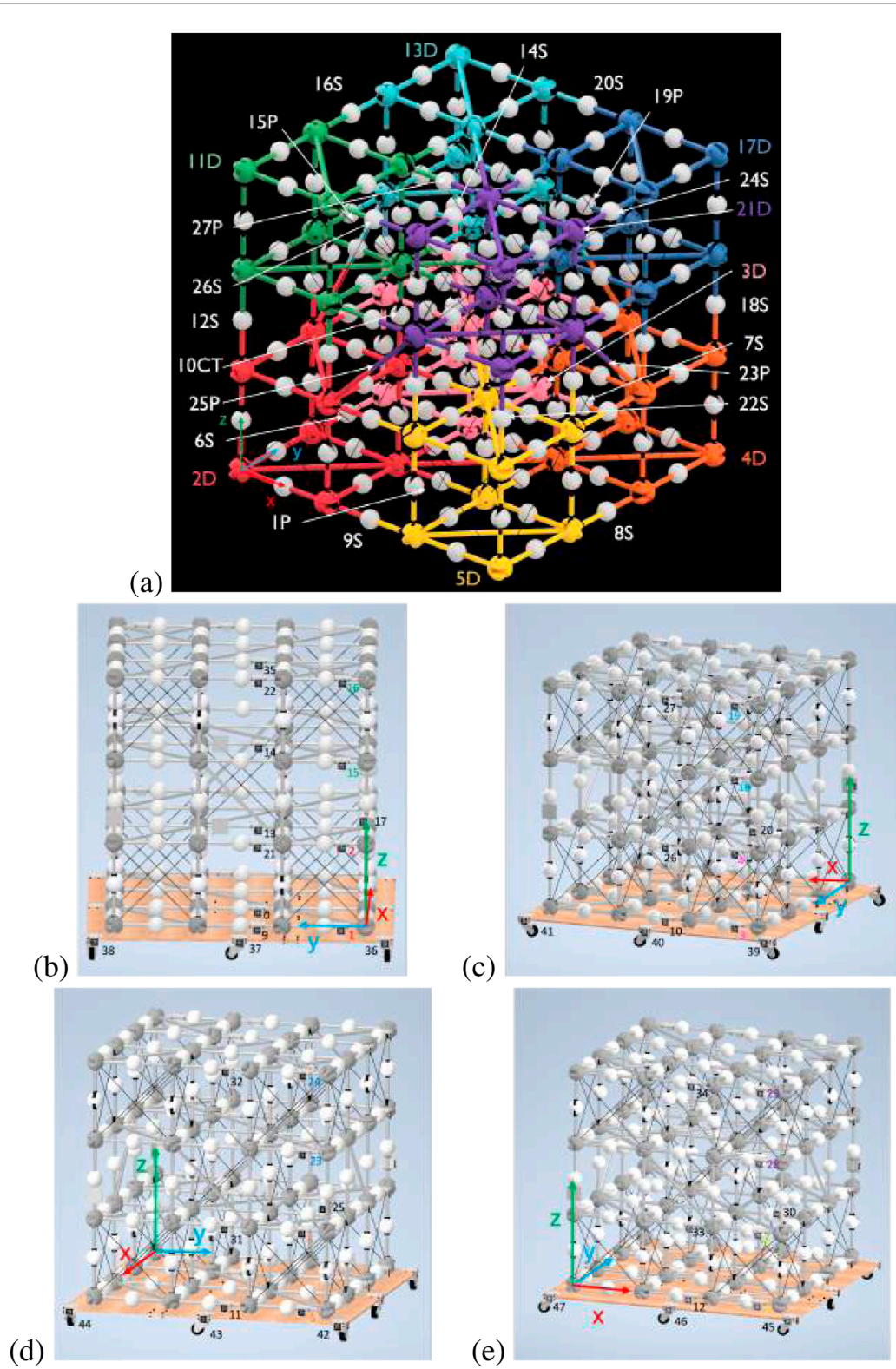


FIGURE 6
(a) BORG truss with labeled modules and axis at the outermost corner node on the first deployable truss, 2D (red). Qualifiers are used after the assembly number order to indicate whether the module is a close-out square (P), deployable (D), close-out strut (S), or the center truss deployable (CT). In addition, the corner deployable modules are color-coded in rainbow order to aid in quick identification during hardware trials. **(b)** Side 1 of the BORG truss. **(c)** Side 2 of the BORG truss. **(d)** Side 3 of the BORG truss. **(e)** Side 4 of the BORG truss.

TABLE 1 AprilTag relationship map: red, deployable; blue, close-out strut; green, close-out square.

Tag number	Opti name	Tags w/single mechanical relationship				Tag number	Opti name	Tags w/single mechanical relationship			
0	1P	1	37			**24**	17D-T	23	32		
1	2D-B	0	2	9	36	**25**	18S	6	23		
2	2D-T	1	17	21		**26**	19P	4	18		
3	3D-B	4	10	39		**27**	20S	19			
4	3D-T	3	20	26		**28**	21D-B	29	30	33	
5	4D-B	6	11	42		**29**	21D-T	28	34		
6	4D-T	5	25	31		**30**	22S	8	28		
7	5D-B	8	12	45		**31**	23P	6	23		
8	5D-T	7	30	33		**32**	24S	24			
9	6S	1	37			**33**	25P	8	28		
10	7S	3	40			**34**	26S	29			
11	8S	5	43			**35**	27P	16			
12	9S	7	46			**36**	TT1	1	37	47	
13	10CT-B	14	2			**37**	TT2	0	9	36	38
14	10CT-T	13	15			**38**	TT3	37	39		
15	11D-B	16	17	21		**39**	TT3	3	38	40	
16	11D-T	15	22			**40**	TT4	10	39	41	
17	12S	2	15			**41**	TT5	40	42		
18	13D-B	19	20	26		**42**	TT5	5	41	43	
19	13D-T	18	27			**43**	TT6	11	42	44	
20	14S	4	18			**44**	TT7	43	45		
21	15P	2	15			**45**	TT7	7	44	46	
22	16S	16				**46**	TT8	12	45	47	
23	17D-B	24	25	31		**47**	TT1	36	46		

2.7 Creating a map of the deployable mechanism and assembled joint relationships

In order for the relationships of all the deployable and assembled modules to be generated and accessible for the SF-GraphSLAM approach, a map was created that records the numbers of adjacent tags that share a single type of relationship to each listed AprilTag. To clarify, there are more adjacent tags for each number, which is observable by the camera, but to be able to reduce the map to connections with only one type of relationship, the following map was generated in Table 1. The deployable relationships are highlighted in red and are governed by the relationship described in Section 2.5.9. The close-out strut relationships are highlighted in blue, and their relationship is dictated in Section 2.5.10. The close-out square relationships are highlighted in green and outlined in Section 2.5.11.

2.8 Simulation structure

The hardware used in the simulation trials consisted of a desktop PC running an AMD Ryzen 9 5900 × 12 core processor

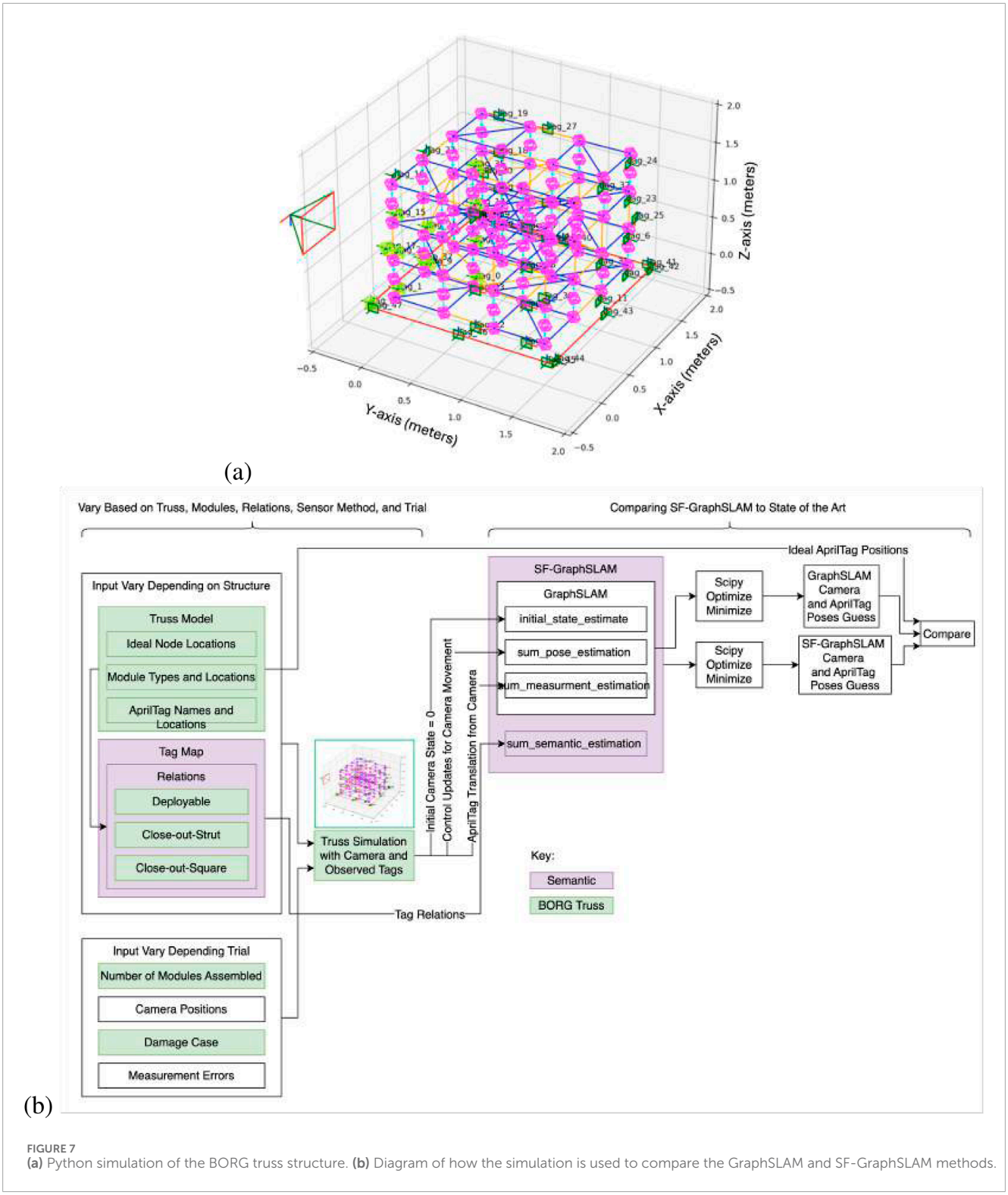


FIGURE 7
(a) Python simulation of the BORG truss structure. **(b)** Diagram of how the simulation is used to compare the GraphSLAM and SF-GraphSLAM methods.

at 3.7 GHz, coupled with 64 GB of DDR4 RAM. The operating environment for the simulation was Python 3.11, with basic-robotics 1.0.2. A Python-plotted simulation was created for the BORG truss example shown in Figure 7a.

The simulation utilized the basic-robotics Python library heavily on the basic-robotics infrastructure Chapin (2023). The simulation has spheres centered at the nodes, in pink, with line elements to represent the struts. There are three types of modules: deployables (shown with blue struts), close-out struts (shown as a single yellow line), and close-out squares (shown as four yellow struts with a diagonal). The turntable is shown with red lines. AprilTags are drawn as green squares and labeled with the tag name and coordinate

Input: State Vector y
Output: Summation $J_{GraphSLAM}$
$J_{GraphSLAM} = initial\_state\_estimate() + sum\_pose\_estimate(y) + sum\_measurement\_estimate(y)$

Algorithm 1. $J_{GraphSLAM}$.

frame. The camera is shown with its own coordinate frame off to the left, and the tags that are viewable from its position are highlighted in bright green. The axis is in units of meters.

The diagram in Figure 7b shows what is input into the simulation by the user, what components are used for the GraphSLAM and SF-GraphSLAM, and the process that generates results to compare.

Depending on the structure, the truss model and tag map need to be updated. For the BORG truss structure, there are ideal node locations, 0.5 m displacement between nodes in an array of four nodes along each axis. The BORG truss has three types of modules, nine deployables, 12 close-out struts, and six close-out squares, and those locations are all stored in the truss model. In addition, the truss model has the positions and orientations of the AprilTags for each module and their unique tag names. The tag map is another file that defines the relationships between tags with a single relationship type of deployable mechanism, close-out strut assembly, or close-out strut assembly, as shown in Table 1. The relationship definitions are also stored in this file and are accessed by the "sum_semantic_estimation" function within SF-GraphSLAM, which is what differentiates it from the state of the art. The deployable close-out strut and close-out square relationships are detailed in Sections 2.5.9–2.5.11, respectively.

Depending on the simulation trial, the number of modules assembled, camera positions and trajectories, truss damage or non-fully deployed or assembled cases, and measurement errors can be adjusted.

The truss and trial inputs listed above influence the simulation run, and the results can then be used to perform GraphSLAM and SF-GraphSLAM calculations. "Scipy.optimize.minimize" is the optimizing function selected for both SLAM cases, using the "Powell" method option and inputting an array of zeros the length of the state vector, the 6-DOF pose of all the camera positions, and a 6DOF pose for each AprilTag being analyzed in the trial. The state vector is then adjusted from zero by the optimizer to minimize the sum of the functions within the SLAM variants to produce its best guess of the locations of all the camera poses and AprilTag poses. The GraphSLAM and SF-GraphSLAM optimizations are run separately and then compared with each other and the ground truth of the truss structure's state in that trial's case.

Within the GraphSLAM (Thrun and Montemerlo, 2006) function are three sub-functions: *initial_state_estimate*, *sum_pose_estimate*, and *sum_measurement_estimate*, which mirror the summations shown in the $J_GraphSLAM$ Equation 26, shown in Algorithm 1.

The *initial_state_estimate* function is shown in Algorithm 2.

The *sum_pose_estimate* function is shown in Algorithm 3.

Algorithm 4 represents the *g* function, a sub-function for *sum_pose_estimate*.

Input: N/A
Output: Summation $init\_state$
$init\_state = x_0^T \Omega_0^{-1} x_0$

Algorithm 2. Initial_state_estimate.

Input: State Vector y
Output: Summation sum_pose_est
Set degree of freedom (dof) value based on data type, 6 for Euler.
$dof = 6$
Set value for state covariance matrix, R [dof, dof].
Check the state vector guess for every camera position.
for t *in range(len(camera_positions))* **do**
    Extract the current guess camera pose from the state vector.
    $x_t = y[t * dof : t * dof + dof]$
    if $t = 0$ **then**
        If it is the first position guess compare it to an array of zeros the length of dof since the camera is supposed to have an initial state of zero.
        $pose\_diff = x_t - [0,0,0,0,0,0]$
    else
        Extract the previous guess camera pose from the state vector.
        $x_{t-1} = y[(t-1) * dof : (t-1) * dof + dof]$
        Get the known controls of the camera between $t-1$ and t, index i
        $u_t = camera\_controls[i]$
        Compare the guess of the current pose with the g function guess based on the previous guess pose and known camera control.
        $pose\_diff = x_t - g(u_t, x_{t-1})$
    end if
    At each time step add the new pose_diff to the previous sum_pose_est.
    $sum\_pose\_est += pose\_diff^T * R^{-1} * pose\_diff$
end for

Algorithm 3. Sum_pose_estimate

Input: New Control u_t and Previous Guess Pose x_{t-1}
Output: Current Pose Guess Based on Previous Pose and Control g
$g = x_{t-1} * u_t$

Algorithm 4. Pose Guess Based on Previous *t* Guess and Movement Since Then: g.

Input: State Vector y
Output: Summation *sum_meas_est*
Set degree of freedom (dof) value based on data type, 6 for Euler.
$dof = 6$
Set value for measurement covariance matrix, Q [dof, dof].
Check the state vector guess for every measurement from each camera position.
for t *in range(len(camera_positions))* **do**
  Extract the current guess camera pose from the state vector.
  $x_t = y[t * dof : t * dof + dof]$
  Create a list of AprilTags that are observed, AprilTags_Observed, by the camera at this time's position.
  for j *in range(len(AprilTags_Desired))* **do**
    Cycle through all the AprilTags_Desired and check if they are within the AprilTags_Observed list.
    if *AprilTags_Desired[j] is within AprilTags_Observed* **then**
      Extract the current guess AprilTag pose from the state vector, y_{TagJ}.
      Calculate the difference between the guess of *TagJ* pose and where the predicted pose of *TagJ* would be based on the current position and measurement at that time, calculated using function h.
      $meas\_diff[j][t] = y_{TagJ} - h(x_t, z_t^j)$
    else
      Do not update sum_meas_est for that tag.
    end if
    At each time step add the new meas_diff to the previous sum_meas_est.
    $sum\_meas\_est += meas\_diff[j][t]^T * Q^{-1} * meas\_diff[j][t]$
  end for
end for

Algorithm 5. Sum_measurement_estimate.

The *sum_measurement_estimate* function is shown in Algorithm 5.

Algorithm 6 represents the *h* function, a sub-function for *sum_measurement_estimate*.

Within the SF-GraphSLAM function are four sub-functions: the first three are the same as the GraphSLAM summation, and a final function called *sum_semantic_estimate*. This mirrors the summation shown in the $J_{SF-GraphSLAM}$ Equation 27, shown in Algorithm 7.

The *sum_semantic_estimate* function is shown in Algorithm 8.

Input: Guess Pose x_t and Measurement of AprilTag j at Time t, z_t^j
Output: Guess Pose of AprilTag j
$h = x_t * z_t^j$

Algorithm 6. AprilTag *j* Guess Pose Based on Camera Guess Pose and Measurment: h.

Input: State Vector y
Output: Summation $J_{SF-GraphSLAM}$
$J_{SF-GraphSLAM} = initial\_state\_estimate() + sum\_pose\_estimate(y) + sum\_measurement\_estimate(y) + sum\_semantic\_estimate(y)$
return $J_{SF-GraphSLAM}$

Algorithm 7. $J_{SF-GraphSLAM}$.

Algorithm 9 represents the *s* function, a sub-function for *sum_semantic_estimate*.

2.9 Simulation implementation

The simulation is set up to be able to focus on any number of desired AprilTags based on the stage of the assembly process at which this analysis is completed and how much of the structure has been assembled. Ideally, this SF-GraphSLAM would be run between assembly steps to verify that the previous deployment or assembly step was completed within the acceptable bounds before continuing assembly to avoid stacking up errors over time. This system can also be run at the end of a full assembly to get the state of each AprilTag and, by relationship, the truss nodes.

3 Results

3.1 Testing tag relationship types with simulation

To test the SF-GraphSLAM approach and compare it against the SOA GraphSLAM approach, we first provided a single example of the camera moving between three positions and observing two AprilTags representing the bottom and top markers of a single deployable module. The results of the GraphSLAM and SF-GraphSLAM of this simulation are shown below.

In this case, two tags are being compared, "Tag_1″ and "Tag_2,″ from the first deployable module to be assembled. This simulation has 0.01 m of translational camera view noise and $0.01 * \pi/180$ radian rotational noise per axis. In addition, there was a 0.1 camera distance noise multiplier, measured as a percentage increase in measured noise per meter from the camera. Finally, there was a 0.05 m camera-reported translation and $0.01 * \pi/180$ radian rotational noise per axis, also known as camera pose error. Random Gaussian noise was included in the pose control update and the

```
Input: State Vector y
Output: Summation sum_sem_est
Set degree of freedom (dof) value based on data type, 6 for Euler.
dof = 6
Set value for measurement covariance matrix, S [dof, dof].
Check the state vector guess for every measurement from each camera position and compare with other
  observed AprilTags with known relations.
for t in range(len(camera_positions)) do
    Extract the current guess camera pose from the state vector.
    x_t = y[t * dof : t * dof + dof]
    Create a list of AprilTags that are observed, AprilTags_Observed, by the camera at this
      time's position.
    for j in range(len(AprilTags_Desired)) do
        Cycle through all the AprilTags_Desired and check if they are within the AprilTags_Observed list.
        for k in range(len(AprilTags_Desired)) do
            Check tag map to see if TagJ has any known relations, TagK and then check if they are also
              within the AprilTags_Observed list.
            if AprilTags_Desired[j] is within AprilTags_Observed then
                if AprilTags_Desired[k] is within AprilTags_Observed then
                    Extract the current guess AprilTag pose from the state vector, y_TagJ.
                    Calculate the difference between the guess of TagJ pose and where the predicted pose
                      of TagJ would be based on the current guess of related TagK pose, y_TagK, and the known
                      relationship between the tags, calculated using function s.
                    sem_diff[j][t] = y_TagJ − s(TagJ,TagK)
                else
                    Do not update sum_meas_est for that tag.
                end if
            else
                Do not update sum_meas_est for that tag.
            end if
        end for
        sum_sem_diff[j] += sem_diff[j][t]^T * S^-1 * sem_diff[j][t]
    end for
    At each time step add the new sum_sem_diff for each j to the previous sum_sem_est.
    for j in range(len(AprilTags_Desired)) do
        sum_sem_est += sum_sem_diff[j]
    end for
end for
```

Algorithm 8. Sum_semantic_estimate.

AprilTag measurement for the measurement function. The covariance matrices Q, R, S are diagonal matrices with the diagonal values given as squared standard deviations in meters and radians and are shown in Equations 34–36. This data set was generated with 50 trial runs, which had their data and plots saved for analysis. The mean square error, root mean squared error, mean, standard deviation, and maximum error are plotted in Table 2, and one of the trial plots is shown below in Figure 8. These data show that the SF-GraphSLAM, on average, has a lower MSE than GraphSLAM for all translation and rotation categories. Therefore, for the deployable example with camera view and reported noise error, the SF-GraphSLAM consistently produces results closer to the ground truth.

$$Q = \mathrm{diag}\left[(1\,\mathrm{m})^2, (1\,\mathrm{m})^2, (1\,\mathrm{m})^2, (0.1\,\mathrm{rad})^2, (0.1\,\mathrm{rad})^2, (0.1\,\mathrm{rad})^2\right] \tag{34}$$

$$R = \mathrm{diag}\left[(0.1\,\mathrm{m})^2, (0.1\,\mathrm{m})^2, (0.1\,\mathrm{m})^2, (0.01\,\mathrm{rad})^2, (0.01\,\mathrm{rad})^2, (0.01\,\mathrm{rad})^2\right] \tag{35}$$

$$S = \mathrm{diag}\left[(0.1\,\mathrm{m})^2, (0.1\,\mathrm{m})^2, (0.1\,\mathrm{m})^2, (0.1\,\mathrm{rad})^2, (0.1\,\mathrm{rad})^2, (0.1\,\mathrm{rad})^2\right] \tag{36}$$

Input: Tag J and K
Output: Guess Pose of AprilTag j with Respect to AprilTag k and Known Relation
1 This function relies on the tag relationship map and relationship designations for deployable mechanisms and close-out strut and close-out square assembly relations.
2 $s = y_{TagK} + relationship(TagJ, TagK)$

Algorithm 9. Semantic Association: Update ApriTag Guess Pose Based on Adjacent Tag and Known Relation Type: s.

3.2 Testing the partially deployed module simulation case

Because the deployable truss modules could have the potential to not be fully deployed before assembly, as shown in Figure 9 we tested a case where this happened to show how we can identify that it is not a fully deployed case and not assume the ideal transformation. In this scenario, the SF-GraphSLAM reverts to GraphSLAM when the AprilTags are outside the bounds of an expected deployed case. Table 3 below shows that both perform equally. This result would be flagged during an assembly step as it is not a complete deployment, and it should be re-deployed or another module swapped out before continuing assembly. Figure 10a shows the simulated partially deployed truss, and Figure 10b shows the results of running GraphSLAM and SF-GraphSLAM, with Figure 10c showing the offset of tag 1 relative to tag 2, and Figure 10d showing both tag 1 and tag 2 together.

3.3 Testing a larger BORG truss simulation case

After verifying all three relational types worked as intended, a larger AprilTag set test was performed with the BORG structure. This example analyzes the first face of the BORG truss structure. Figure 11 shows the GraphSLAM and SF-GraphSLAM results for this experiment. The pose estimate errors for SF-GraphSLAM are listed in Table 4. This simulation has 0.01 m of translational camera view noise and 0.01*pi/180 rotational noise. In addition, there was a 0.1 camera distance noise multiplier, measured as a percentage increase in measured noise per meter from the camera. There was a 0.05 m camera-reported translation and $0.01 * \pi/180$ rotational noise, also known as camera pose error. Finally, random Gaussian noise was included in the pose control update and the AprilTag measurement for the measurement function.

Due to the computational time involved in processing all the tags for the BORG cube, only a single face was analyzed. The concept of implementing SF-GraphSLAM is to run it often between assembly steps with smaller sets of AprilTags and then update the simulated truss reference, which is carried over into the next inspection task. Therefore, an ideal and guess state vector for all the tags can be maintained locally and referenced instead of having to re-calculate it from guesses of zero each implementation.

3.4 Testing tag elimination

This test case is used to show that if an AprilTag is incorrectly placed, a verification step can be used to determine that this tag result is erroneous and can be eliminated if the rest of the assembly is valid. The standard concept of operations entails running SF-GraphSLAM at the end of each assembly step and ensuring that the deployable and assembled modules are placed properly. Therefore, this verification step is only for checking whether a tag has been moved or obscured later, causing bad results. The process entails taking the output of SF-GraphSLAM, x, and attempting to best fit all the AprilTag values to the ideal truss. This test was performed with tags on the first face of the BORG structure. A base tag was selected to be the first tag, "Tag_1," to use its pose as a guess and try to minimize the other tag guess error with respect to it, assuming an ideal truss structure. An error of a 0.3 m tag displacement was simulated on "Tag_17" in the $y-axis$. This test runs through the guess positions of the observed tags based on the base tag and ideal transforms and then calculates the distance between the guessed location and the measured location. Then, the distances are sorted in descending order, with the worst fit tags (with the largest distances) at the beginning. A removal cutoff, maximum distance allowable, and a maximum number of tags to remove can be specified. Each tag distance is evaluated, and if it is above the allowable cutoff, the tag is thrown out. This test found that Tag_17 was outside the cutoff, and it was removed from the tag list. This is allowable because, since the surrounding structure is compared against and is within expected bounds, a deduction can be made that the tag's position would be impossible to return for a properly placed AprilTag, while the rest of the structure does not also show cascading damage error. This can be done with a minimum of three tags and up to as many tags as desired.

3.5 Quantifying measurement accuracy requirements for space structures and robustness against sensor and measurement error

Based on the introduced error in the trials above, we can quantify the robustness against sensor and measurement error of SF-GraphSLAM compared to the SOA GraphSLAM due to its higher accuracy. Structures developed for in-space assembly by NASA Langley Research Center (Dorsey et al., 2021; Hedgepeth, 2012) were used for reference of root mean squared error (RMSE) and compared against. In a critical requirements document for the design of large space structures (Hedgepeth, 2012), it was noted that an accuracy of 0.1 mm would be required for a 10-m-long member. A 102-member tetrahedral truss structure example with 0.14 mm RMSE and a 14-m diameter truss with a surface precision of 0.0719 mm RMSE (Dorsey et al., 2021; Hedgepeth, 2012) was used as a reference, and an average goal RMSE was calculated. Then, the RMSE values from various trial

10.3389/frobt.2025.1426676

TABLE 2 Deployable example with camera view and reported noise error: mean squared error, root mean squared error, mean, standard deviation, and maximum error for GraphSLAM offset, SF-GraphSLAM offset, and the comparison of the two with the SF-GraphSLAM offset divided by the GraphSLAM offset.

Method	Evaluation	X_Trans (m)	Y_Trans (m)	Z_Trans (m)	X_Rot (rad)	Y_Rot (rad)	Z_Rot (rad)
GraphSLAM	Mean squared error (MSE)	5.02E−04	3.13E−03	1.28E−04	5.97E−07	1.55E−06	3.73E−07
	Root mean squared error (RMSE)	2.241 E−02	5.59E−02	1.13E−02	7.73E−04	1.24E−03	6.11E−04
	Mean	2.24E−02	5.59E−02	1.12E−02	7.61E−04	1.24E−03	6.00E−04
	Standard deviation	1.251 E−03	1.19E−03	1.31E−03	1.34E−04	1.30E−04	1.161 E−04
	Max error	−2.26E−02	−5.62E−02	−1.07E−02	−1.03E−03	1.57E−03	−4.25E−04
SF-GraphSLAM	Mean squared error (MSE)	4.21E−07	1.16E−06	5.65E−08	3.851 E−07	9.53E−07	2.92E−07
	Root mean squared error (RMSE)	6.49E−04	1.08E−03	2.38E−04	6.21E−04	9.76E−04	5.41E−04
	Mean	6.26E−04	1.06E−03	1.84E−04	6.03E−04	9.71E−04	5.32E−04
	Standard deviation	1.71E−04	2.03E−04	2.38E−04	1.46E−04	1.041 E−04	9.71E−05
	Max error	−7.63E−04	−9.65E−04	7.57E−04	−5.45E−04	1.16E−03	−5.66E−04
Ratio Between SF-GraphSLAM and GraphSLAM	Mean squared error (MSE)	8.37E−04	3.72E−04	4.42E−04	6.45E−01	6.16E−01	7.83E−01
	Root mean squared error (RMSE)	2.89E−02	1.93E−02	2.10E−02	8.03E−01	7.85E−01	8.85E−01
	Mean	2.80E−02	1.89E−02	1.64E−02	7.93E−01	7.85E−01	8.87E−01
	Standard deviation	1.37E−01	1.70E−01	1.82E−01	1.09E+00	7.98E−01	8.36E−01
	Max error	3.38E−02	1.72E−02	−7.07E−02	5.30E−01	7.38E−01	1.33E+00

runs, with different levels of introduced error, were averaged for translation (m) and rotation (rad) error and compared against the reference average to see whether they were higher or lower. These results are shown in Table 5. RMSE values that are above the reference are highlighted in red, while values below are highlighted in green. For these trials, both the GraphSLAM and SF-GraphSLAM rotational values are above the average, but more trials could be done with less introduced rotational error. In terms of translation error, the SF-GraphSLAM performs better and has all values below the reference's average. This is significant because it shows SF-GraphSLAM's increased accuracy allows for robustness against sensor and measurement errors. This is because even though there is a variety of introduced errors, SF-GraphSLAM can still estimate the positions of the tags within error margins that are smaller than the error of the reference truss. This is required to be able to measure anomalies in the truss structure itself.

4 Discussion

SF-GraphSLAM was shown to reduce the mean squared error of fiducial pose estimation attached to an example modular space truss structure compared to the state-of-the-art GraphSLAM. The SF-GraphSLAM method successfully combined the fast detection and measurement of fiducials, AprilTags, with semantic information about the truss structure being assembled to aid in estimating module poses, even when there was considerable noise from both the camera's simulated pose and measurement.

The previous work in the BORG mixed assembly trade study (Chapin et al., 2023) explained the mixed assembly method, reducing the number of unique assembly components by using a mixture of deployable and assembled modules. It then proves how the mixed assembly method can be used to minimize the state space due to the reduction of components tracked. This benefit is only increased by using sparsely placed fiducials on the modules to

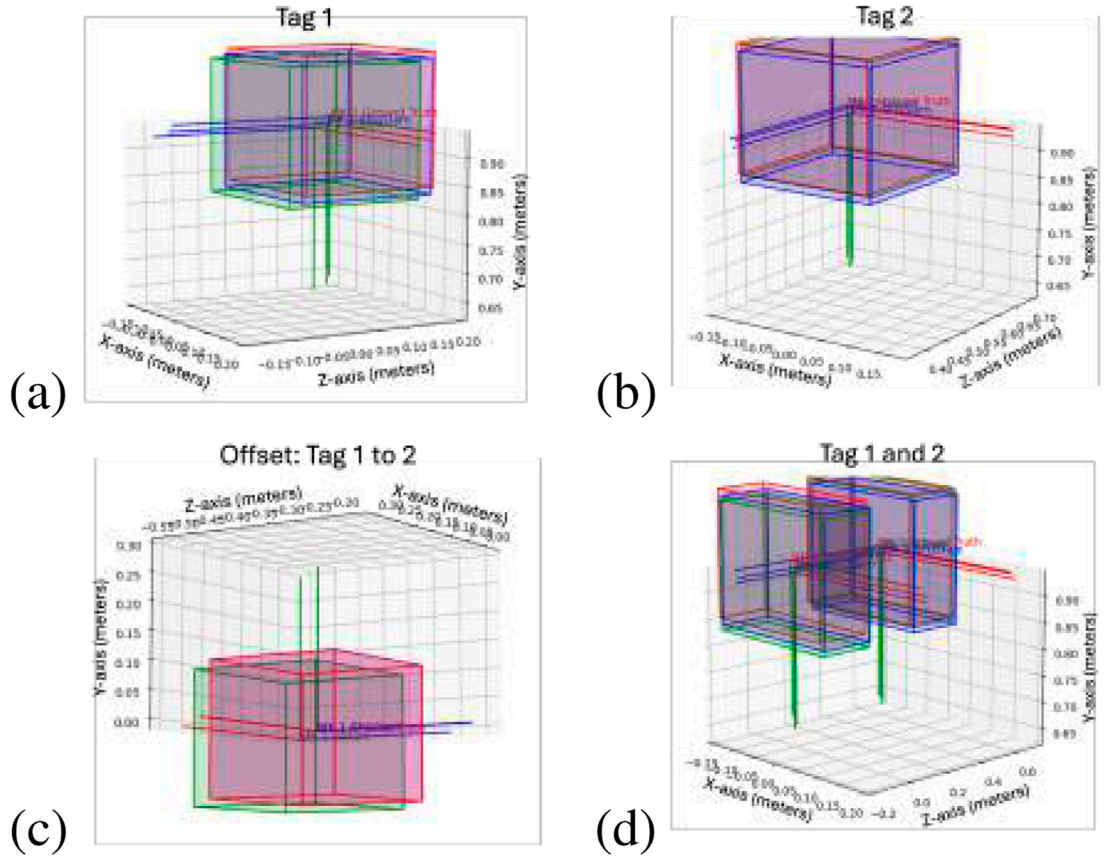


FIGURE 8
Deployable truss simulation test example. This plots the results of one of the trial runs. Red indicates the ideal marker positions, blue indicates the SF-GraphSLAM guess poses, and green indicates the GraphSLAM guess tag poses. The top two plots are AprilTag positions with respect to the camera; **(a)** tag 1 on the left and **(b)** tag 2 on the right. **(c)** Plots the offset between tag 2 with respect to tag 1 for the respective ideal, SF-GraphSLAM, and GraphSLAM values. **(d)** Plotting both tags in the camera frame.

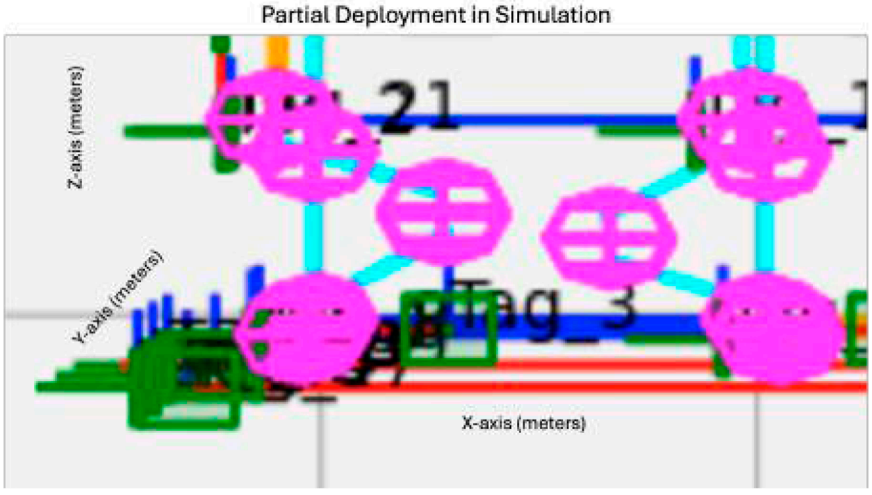


FIGURE 9
Partially deployed truss test simulation setup.

TABLE 3 Partially deployed truss test case: mean squared error, root mean squared error, mean, standard deviation, and maximum error for GraphSLAM offset, SF-GraphSLAM offset, and the comparison of the two with the SF-GraphSLAM offset divided by the GraphSLAM offset.

Quantity	X_Trans (m)	Y_Trans (m)	Z_Trans (m)	X_Rot (rad)	Y_Rot (rad)	Z_Rot (rad)
GraphSLAM offset MSE	0.000823202	4.07E−06	0.000497053	8.88E−08	5.72E−08	1.18E−07
GraphSLAM offset RMSE	0.028691503	0.002016924	0.022294694	0.000297956	0.000239212	0.000343207
GraphSLAM offset mean	0.02843751	0.001515303	0.02219467	0.000256795	0.000201602	0.00032011
GraphSLAM offset stdev	0.003809247	0.001758544	0.002109508	0.000165152	0.00019409	0.000123777
GraphSLAM offset max_error	0.032634787	0.00035341	−0.025219943	0.000524758	0.000452405	0.00053455
SF-GraphSLAM offset MSE	0.000909264	1.27E−05	0.000492483	7.76E−08	3.50E−08	3.40E−07
SF-GraphSLAM offset RMSE	0.030154006	0.003565644	0.022191951	0.000278536	0.00018711	0.000583429
SF-GraphSLAM offset mean	0.030147366	0.003058347	0.02186321	0.00026789	0.00014472	0.000508247
SF-GraphSLAM offset stdev	0.000632779	0.001833122	0.003805622	7.63E−05	0.000185967	0.000286485
SF-GraphSLAM offset max_error	0.031189212	−0.000758613	−0.024521632	0.000397442	0.000430115	0.000921899
SF/G MSE	1.10E+00	3.13E+00	9.91E−01	8.74E−01	6.12E−01	2.89E+00
SF/G RMSE	1.050973386	1.767862025	0.995391594	0.93482168	0.782191311	1.699932567
SF/G mean	1.06E+00	2.02E+00	9.85E−01	1.04E+00	7.18E−01	1.59E+00
SF/G stdev	1.66E−01	1.04E+00	1.80E+00	4.62E−01	9.58E−01	2.31E+00
SF/G max_error	9.56E−01	−2.15E+00	9.72E−01	7.57E−01	9.51E−01	1.72E+00

remove the need to track the states of sub-components other than the deployable top and bottom planes.

The mathematical approach shows how the SF-GraphSLAM approach is derived by adding a semantic element of known relationships between map elements and adding that to the state-of-the-art GraphSLAM approach. Improvements in the completeness of the graph relationships and information matrix are shown.

The simulation results show the construction of a built on-orbit robotically assembled gigatruss (BORG) simulation, based on the mixed assembly method. This allows for the comparison of SF-GraphSLAM and GraphSLAM on a structure with controllable noise. A series of tests were conducted with both methods, attempting to optimize for the best guess of the AprilTag poses based on simulated camera control and measurement input. The trials had a range of introduced camera poses and measurement simulated errors. The maximum camera measurement error used in the simulation was 0.01 m of translational camera view noise, $0.01 * \pi/180$ rotational noise, and a 0.1 camera distance noise multiplier. The maximum camera pose noise was a 0.05 m translation and $0.01 * \pi/180$ rotational noise. In addition, when a series of trial runs was completed for use in finding the mean squared error and other evaluation criteria, there was random Gaussian noise on the pose control update and the AprilTag measurement for the measurement function. In these tests, SF-GraphSLAM consistently performed better at utilizing the semantic relationship map to find better fit AprilTag poses that were able to counteract some of the error and provide a better pose estimate. The simulation environment was also used to test tag rejection; when a single or a small number of tags have large errors, while the surrounding structure fits the model well, the tag can be eliminated and assumed to be a tag placement error. Finally, the robustness of the system was considered and compared well to other space structure examples of required root mean squared error across the large truss surface.

Overall, this comparative experimental data shows increased performance of the SF-GraphSLAM approach compared to the GraphSLAM algorithm on which it was based. The consistency of this improved performance depends on the accuracy of the imputed semantic knowledge of the modules, fiducial locations, fiducial relations, and overall structure assembly. If these inputs are not accurate, then it could lead to the SF-GraphSLAM approach biasing the pose estimates to incorrect locations. Initial hardware testing of SF-GraphSLAM was conducted and documented by Chapin et al. (2024) and Chapin (2024), but the error in the hardware was too great to see the same increased performance over the GraphSLAM approach as was seen in simulation.

4.1 Potential drawbacks

While the use of semantic information is shown to provide an improved estimate of the evolution of the structure, there are

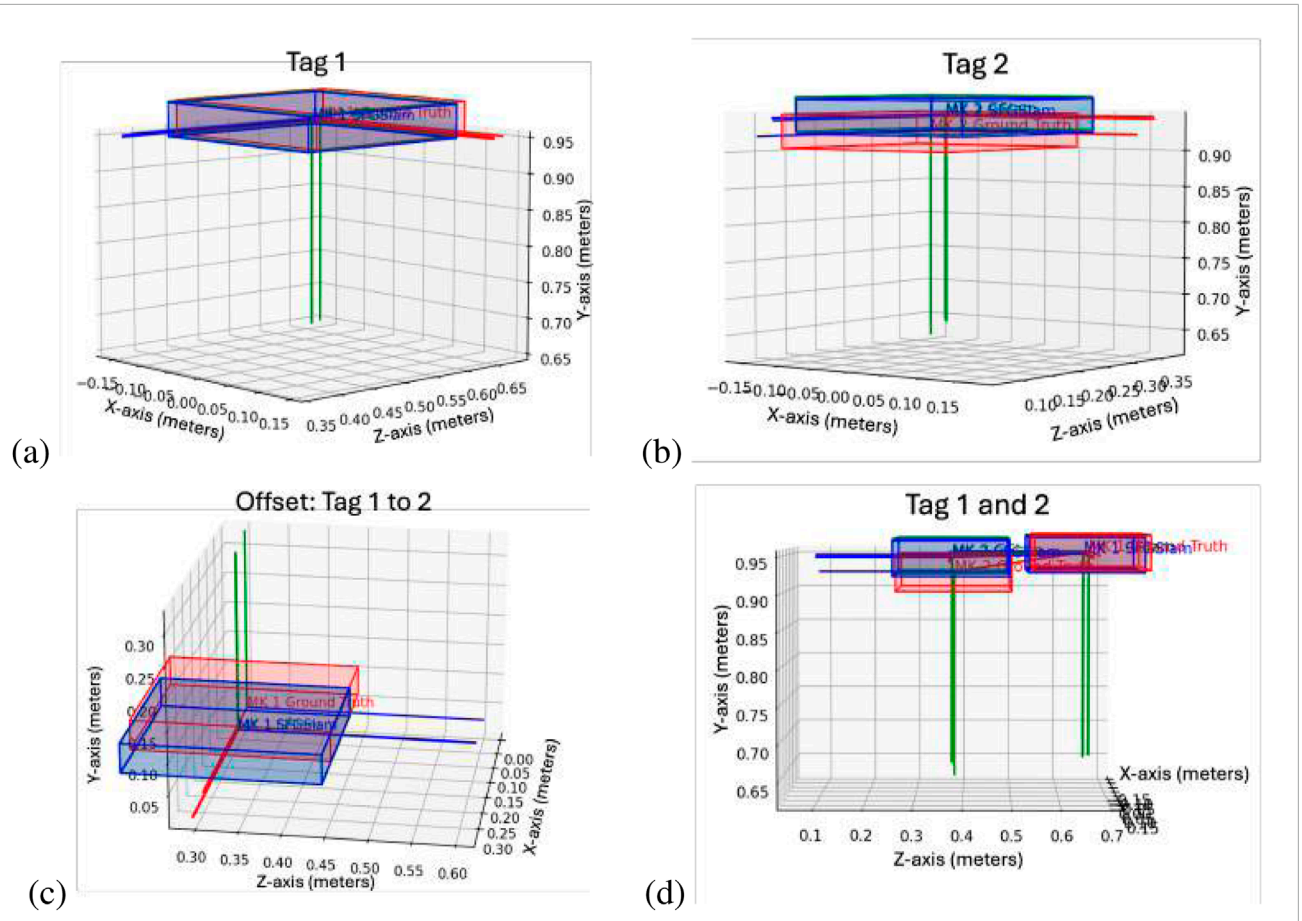


FIGURE 10
Partially deployed truss results. **(a)** Tag 1 plot. **(b)** Tag 2 plot. **(c)** Offset of tag 1 to tag 2 plot. **(d)** Tags 1 and 2 plotted together.

potential areas of concern. Because SF-GraphSLAM is a factor graph algorithm that considers the entire history of the states of the assembly robots and the structure elements, the time required to find a local minimum will increase, consuming more and more energy as the assembly time goes on. This can be countered by using an incremental factor graph method. If time is a critical resource, this can be further mitigated by using a filtering approach that does not retain the full state history (e.g., EKF-SLAM), which further ensures that computation time per step is not dependent on how long the system has been in operation.

This work also assumes that visual markers are not hindered by sunlight, which may not always be possible. Without mitigation, a reduction in sensor information (and sensor quality) will result in a reduction in accuracy and may possibly lead to scenarios where there is not sufficient information to find a single optimal estimate for one or more components. In such cases, SF-GraphSLAM can be augmented with factors representing data from different sensor types, including laser range finders and retro-reflective markers. Furthermore, the strategic orientation of visual sensors, marker placement, and assembly time to mitigate the influence of sunlight will need to be part of the consideration for system design and sequence planning.

4.2 Contribution to the state of the art

The SF-GraphSLAM method is a new way to focus on a SLAM method that can handle tracking components used for robotic assembly. The concept of the mixed assembly method, using deployable and assembled modules, is novel and is shown to greatly reduce the state vector of the assembly problem. This state vector is further minimized by leveraging scarcely placed fiducials on the truss modules. SF-GraphSLAM also shows a new way to create and store relationships between map elements and integrate them in the SLAM to create more connections in the graph that can generate an updated summation to be optimized to find the most likely poses of the moving camera and map elements. This novel method is shown in mathematical derivation and simulation within this article and will be shown in hardware trials in a subsequent article.

4.3 Future work

SF-GraphSLAM was compared against the GraphSLAM algorithm it was originally derived from. Comparisons of SF-GraphSLAM's performance against other SOA factor graph-based algorithms would allow for better analysis of its relative performance. Similarly, the initial hardware testing (Chapin et al.,

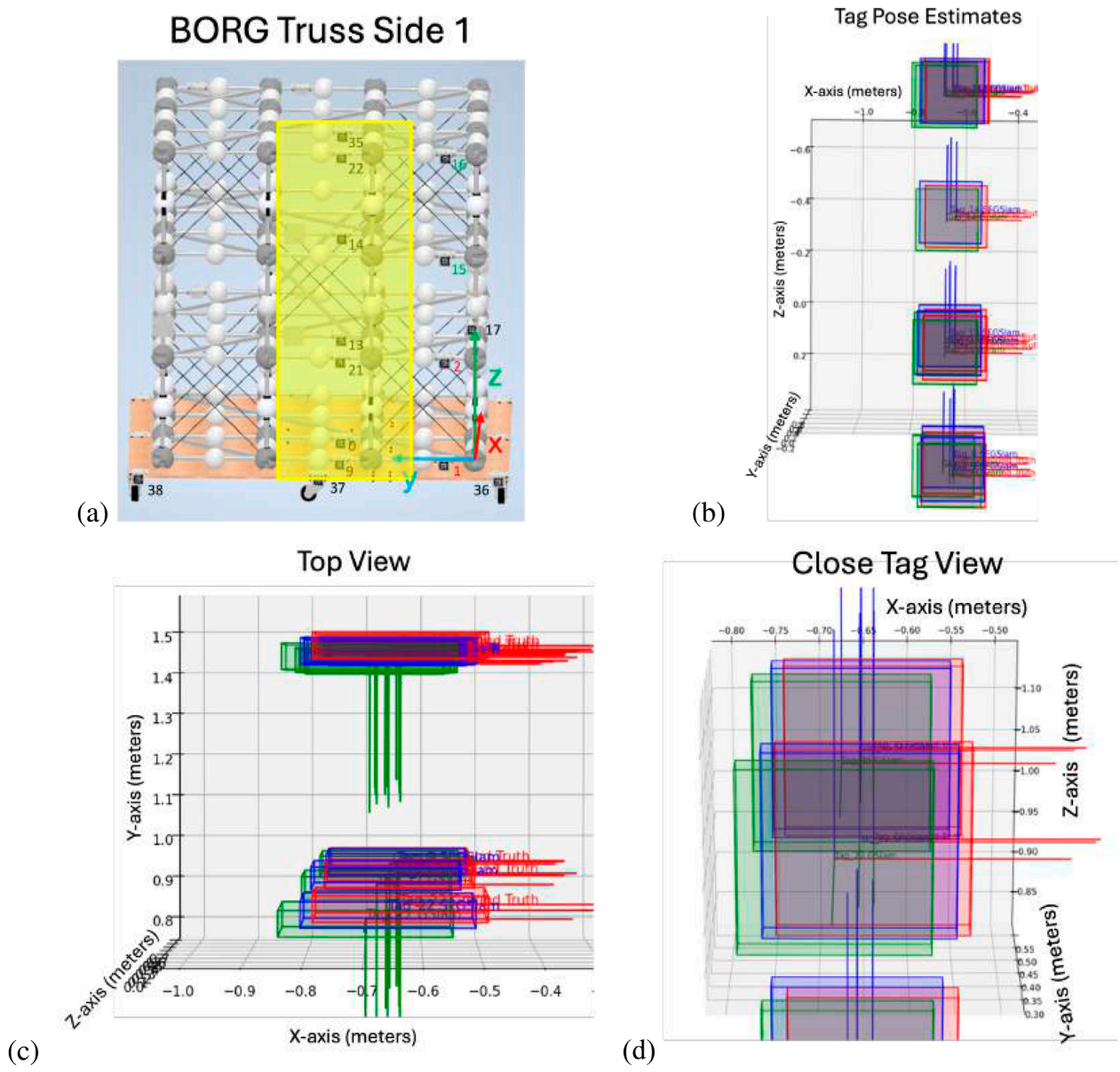


FIGURE 11
The plot of the side of the BORG truss structure. GraphSLAM (green) and SF-GraphSLAM (blue) pose estimates are plotted against the ground truth (red). **(a)** BORG Truss Side 1, with the plotted tags highlighted **(b)** Front view. **(c)** Top view. **(d)** Close-up view.

2024) also compared SF-GraphSLAM against GraphSLAM, so further algorithm comparisons in hardware testing would be beneficial. Additionally, it is crucial that more precise hardware be utilized to allow for better testing so that the semantic inputs for the SF-GraphSLAM are accurate and can lead to higher accuracy in the pose estimates.

AprilTags were proposed in this experimentation because a large amount of experimental data has been collected using them across a wide variety of robotics experimentation, making them a good fiducial candidate. We considered what an ideal fiducial for this type of robotic ISAM-focused SLAM could be. Criteria for an ideal fiducial include being viewable from more orientations and perspectives; AprilTags can only be seen from a point of view that is perpendicular to the tag. With this goal in mind, a concept was generated of a cylindrical fiducial that wraps around the strut cylinder and is viewable from many different vantage points. Conceptually, it could look like a bar code with square notches for angular identification. Additionally, this fiducial could be scaled to be larger and seen from further away to enable easier perception of larger structures. Another idea is to even have some sort of embedded fiducial design that changes as you get closer to the fiducial to allow for perception at varying distances. Finally, integrating the fiducial into the strut itself would also

TABLE 4 Error of SF-GraphSLAM and GraphSLAM pose estimates with respect to ground truth for the test of the first side of the BORG truss structure. Difference of the absolute (abs) errors of SF-GraphSLAM and GraphSLAM; a negative number means less SF-GraphSLAM error.

Tag	Approach	X_Trans (m)	Y_Trans (m)	Z_Trans (m)	X_Rot (rad)	Y_Rot (rad)	Z_Rot (rad)
Tag 22	GraphSLAM	−4.961E−02	−1.799E−02	2.942E−02	5.759E−04	−4.250E−04	−1.407E−03
	SF-GraphSLAM	−1.751E−02	−4.901E−04	1.097E−02	7.929E−04	−1.501E−03	−8.125E−04
	Abs difference	−3.210E−02	−1.750E−02	−1.844E−02	2.170E−04	1.076E−03	−5.943E−04
Tag 35	GraphSLAM	−4.432E−02	−1.754E−02	2.156E−02	−3.262E−04	−1.170E−04	−4.423E−04
	SF-GraphSLAM	−1.688E−02	−5.926E−04	1.108E−02	−6.524E−05	−5.748E−04	3.565E−04
	Abs difference	−2.744E−02	−1.695E−02	−1.049E−02	−2.610E−04	4.577E−04	−8.585E−05
Tag 14	GraphSLAM	−3.664E−02	−1.299E−02	2.390E−02	−7.051E−04	1.368E−04	−3.720E−04
	SF-GraphSLAM	−2.258E−02	1.712E−02	2.336E−03	−8.141E−04	−1.087E−03	5.969E−05
	Abs difference	−1.407E−02	4.130E−03	−2.157E−02	1.090E−04	9.503E−04	−3.123E−04
Tag 21	GraphSLAM	−3.836E−02	−1.558E−02	2.669E−02	−1.489E−04	−3.949E−04	−1.224E−03
	SF-GraphSLAM	−2.181E−02	1.931E−02	3.057E−03	6.408E−04	−1.222E−03	3.722E−03
	Abs difference	−1.656E−02	3.731E−03	−2.364E−02	4.919E−04	8.267E−04	2.498E−03
Tag 13	GraphSLAM	−3.993E−02	−1.139E−02	2.149E−02	−3.851E−04	−7.139E−04	−9.356E−04
	SF-GraphSLAM	1.806E−02	2.474E−03	−1.247E−04	−1.278E−03	4.368E−03	−2.105E−02
	Abs difference	−2.187E−02	−8.916E−03	−2.137E−02	8.929E−04	3.655E−03	2.012E−02
Tag 9	GraphSLAM	−1.632E−02	1.008E−03	7.035E−03	−7.723E−04	2.526E−04	6.063E−04
	SF-GraphSLAM	−4.086E−03	2.498E−02	3.459E−04	−6.323E−05	1.492E−04	4.321E−03
	Abs difference	−1.224E−02	2.397E−02	−6.689E−03	−7.091E−04	−1.033E−04	3.715E−03
Tag 0	GraphSLAM	−4.377E−02	−9.908E−03	2.190E−02	1.554E−04	1.362E−04	−6.266E−04
	SF-GraphSLAM	−5.089E−03	2.477E−02	6.166E−04	−2.209E−04	−4.127E−04	1.078E−03
	Abs difference	−3.868E−02	1.486E−02	−2.128E−02	6.556E−05	2.765E−04	4.518E−04

eliminate concerns about the extra mass that fiducials add to the structure.

Data availability statement

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

Author contributions

SC: Writing – original draft, Writing – review and editing. WC: Writing – review and editing. EK: Writing – review and editing.

Funding

The author(s) declare that financial support was received for the research and/or publication of this article. This research was supported by NASA Langley Research Center through a cooperative agreement with the National Institute of Aerospace (C15-2B00-VT sub-award 202101-VT) and a contract with Analytical Mechanics Associates, Inc. (RSES.C2.09.00108.001).

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

TABLE 5 This table shows the root mean square average for translation (m) and rotational (rad) error for a series of trials. Settings for the error in the trials are specified on the left. The results are compared against an average of RMSE from other space structure research as a reference. RMSE values that are above the example are highlighted in red, and values below the example are highlighted in green.

Dataset	Random Gaussian noise		Camera view noise - AKA at means error			Camera reported noise - AKA pose error	
	Pose (m)	Means (m)	Multiplier	Trans (m)	Rot (rad)	Trans (m)	MultiplierRot (rad)
Deployable test - 10 runs	0.00001	0.00001	0	0	0	0	0
Deployable test - x0 Guess - 50 runs	0.00001	0.00001	0.01	0.001745329	0.01	0.05	0.001745329
Deployable test - x0 = 0–50 runs	0.00001	0.00001	0.01	0.001745329	0.01	0.05	0.001745329
Close-out square test - 10 runs	0.00001	0.00001	0.01	0.001745329	0.01	0	0
Close-out strut test - 10 runs	0.00001	0.00001	0	0	0	0.05	0.001745329

Dataset	GraphSLAM RMSE		SF-GraphSLAM RMSE	
	Trans (m)	Rot (rad)	Trans (m)	Rot (rad)
Deployable test - 10 runs	0.029943841	0.000883736	0.000656134	0.00063248
Deployable test - x0 Guess - 50 runs	0.02988171	0.000876039	0.000654889	0.000712574
Deployable test - x0 = 0–50 runs	0.164772152	0.029242353	0.000677433	0.000736049
Close-out square test - 10 runs	0.007224853	0.00084527	0.000307993	0.000687931
Close-out strut test - 10 runs	0.00468295	0.000411439	0.000249976	0.000160106
Average RMSE	**0.047301101**	**0.006451768**	**0.000509285**	**0.000585828**

10.3389/frobt.2025.1426676

Publisher's note

References

Abaspur Kazerouni, I., Fitzgerald, L., Dooly, G., and Toal, D. (2022). A survey of state-of-the-art on visual slam. *Expert Syst. Appl.* 205, 117734. doi:10.1016/j.eswa.2022.117734

Abawi, D., Bienwald, J., and Dorner, R. (2004). Accuracy in optical tracking with fiducial markers: an accuracy function for artoolkit. *Third IEEE ACM Int. Symposium Mix. Augmented Real.*, 260–261. doi:10.1109/ISMAR.2004.8

Asril, E. G., and Zhu, Z. H. (2025). Shape reconstruction of unknown tumbling target using factor graph-based dynamic slam. *AIAA SciTech 2025, SAR-01 In-Space On-Orbit Serv. Robotics Sess.* doi:10.2514/6.2025-0180

Bettens, A., Morrell, B., Coen, M., Wu, X., Gibbens, P., and Chamitoff, G. (2024). Simultaneous localization and mapping architecture for small bodies and space exploration. *Adv. Space Res.* 73, 1185–1197. doi:10.1016/j.asr.2023.10.048

Cavaciuti, A. J., Heying, J. H., and Davis, J. (2022). In-space servicing, assembly, and manufacturing for the new space economy. *Aerosp. Cent. Space Policy Strategy*, 2022–07.

Chapin, S. (2024). *Semantic and fiducial aided graph simultaneous localization and mapping for robotic in-space assembly and servicing of large truss structures*. Virginia Tech.

Chapin, S., Everson, H., Chapin, W., and Komendera, E. (2024). Built on-orbit robotically assembled gigatruss (Borg): ground robotic demonstration. *Aerospace* 11, 447. doi:10.3390/aerospace11060447

Chapin, S., Everson, H., Chapin, W., Quartaro, A., and Komendera, E. (2023). Built on-orbit robotically assembled gigatruss (Borg): a mixed assembly architecture trade study. *Front. Robotics AI* 10, 1109131. doi:10.3389/frobt.2023.1109131

Chapin, W. (2023). Gazebo garden. Available online at: https://github.com/64-B1T/basic_robotics.

Chen, W., Shang, G., Ji, A., Zhou, C., Wang, X., Xu, C., et al. (2022). An overview on visual slam: from tradition to semantic. *Remote Sens.* 14, 3010. doi:10.3390/rs14133010

Chen, Z. (2003). Bayesian filtering: from kalman filters to particle filters, and beyond. *Statistics* 182. doi:10.1080/02331880309257

Dale Arney, Dr., and Mulvaney, J. (2023). *Space servicing, assembly, and manufacturing (ISAM) state of play*. 2023 Edition. Available online at: https://www.nasa.gov/wp-content/uploads/2023/10/isam-state-of-play-2023.pdf.

DeGol, J., Bretl, T., and Hoiem, D. (2018). "Improved structure from motion using fiducial marker matching," in *European conference on computer vision*.

Dellaert, F., and Kaess, M. (2017). Factor graphs for robot perception. *Found. Trends® Robotics* 6, 1–139. doi:10.1561/2300000043

Dorsey, J., Doggett, W., McGlothin, G., Alexandrov, N., Allen, B., Chandarana, M., et al. (2021). State of the profession: Nasa langley research center capabilities/technologies for autonomous in-space assembly and modular persistent assets. *Bull. AAS* 53. doi:10.3847/25c2cfeb.7707c9bf

Fiala, M. (2010). Designing highly reliable fiducial markers. *IEEE Trans. Pattern Analysis Mach. Intell.* 32, 1317–1324. doi:10.1109/TPAMI.2009.146

Garcia, M. (2022). International space station. Available online at: https://www.nasa.gov/mission_pages/station/main/index.html.

Garner, R. (2018). Hubble servicing missions overview. Available online at: https://www.nasa.gov/mission_pages/hubble/servicing/index.html.

Gregg, C. E., and Cheung, K. C. (2024). "Precision in assembled discrete lattice space structures for next-generation isam applications," in *2024 IEEE aerospace conference* (IEEE), 1–9.

Hedgepeth, J. (2012). Critical requirements for the design of large space structures. doi:10.2514/6.1981-443

Henshaw, C. G., Glassner, S., Naasz, B., and Roberts, B. (2022). Grappling spacecraft. *Annu. Rev. Control, Robotics, Aut. Syst.* 5, 137–159. doi:10.1146/annurev-control-042920-011106

Jamieson, K. G., Nowak, R., and Recht, B. (2012). Query complexity of derivative-free optimization. *Adv. Neural Inf. Process. Syst.* 25. doi:10.48550/arXiv.1209.2434

Komendera, E. E., Adhikari, S., Glassner, S., Kishen, A., and Quartaro, A. (2017). *Structure assembly by a heterogeneous team of robots using state estimation, generalized joints, and Mobile parallel manipulators*. IEEE/RSJ International Conference on Intelligent Robots and Systems IROS, 4672–4679. doi:10.1109/IROS.2017.8206338

Leinz, M. R., Chen, C.-T., Scott, P., Gaumer, W., Sabasteanski, P., and Beaven, M. (2008). "Modeling, simulation, testing, and verification of the orbital express autonomous rendezvous and capture sensor system (ARCSS),". *Sensors and systems for space applications II*. Editors R. T. Howard, and P. Motaghedi (Bellingham, WA: SPIE), 6958. doi:10.1117/12.779599695 80C

Lightbody, P., Krajník, T., and Hanheide, M. (2017). "A versatile high-performance visual fiducial marker detection system with scalable identity encoding," in *Proceedings of the symposium on applied computing* (New York, NY, USA: Association for Computing Machinery, SAC '17), 276–282. doi:10.1145/3019612.3019709

Lynch, K. M., and Park, F. C. (2017). *Modern robotics: mechanics, planning, and control*. 1st edn. USA: Cambridge University Press.

Mahmoud, A., and Atia, M. (2022). Improved visual slam using semantic segmentation and layout estimation. *Robotics* 11, 91. doi:10.3390/robotics11050091

Moser, J. N., Wolf, J. H., Cresta, C. J., Guo, R., Rajaram, R., and Cooper, J. R. (2024). "Pose estimation for autonomous in-space assembly," in *Aiaa aviation forum and ascend 2024*, 4909.

Mukherjee, R. (2023). "Survey of select recent in-space servicing assembly and manufacturing related robotics projects at the jet propulsion laboratory," in *Ascend 2023*, 4700.

Obermark, J., Creamer, G., Kelm, B. E., Wagner, W., and Henshaw, C. G. (2007). "SUMO/FREND: vision system for autonomous satellite grapple,". *Sensors and systems for space applications*. Editors R. T. Howard, and R. D. Richards (Bellingham, WA: SPIE), 6555, 65550Y. doi:10.1117/12.720284

Ogilvie, A., Allport, J., Hannah, M., and Lymer, J. (2008). "Autonomous satellite servicing using the orbital express demonstration manipulator system," in *Proceedings of the 9th international symposium on artificial intelligence, robotics and automation in space*. Bellingham, WA: iSAIRAS.

Olson, E. (2011). "Apriltag: a robust and flexible visual fiducial system," in *2011 IEEE international conference on robotics and automation*, 3400–3407. doi:10.1109/ICRA.2011.5979561

Pfrommer, B., and Daniilidis, K. (2019). Tagslam: robust slam with fiducial markers

Post, M. A., Yan, X.-T., and Letier, P. (2021). Modularity for the future in space robotics: a review. *Acta Astronaut.* 189, 530–547. doi:10.1016/j.actaastro.2021.09.007

Qi, X., Huang, H., Li, B., and Deng, Z. (2016). A large ring deployable mechanism for space satellite antenna. *Aerosp. Sci. Technol.* 58, 498–510. doi:10.1016/j.ast.2016.09.014

Reed, B. B., Bacon, C., and Naasz, B. J. (2017). Designing spacecraft to enable robotic servicing. doi:10.2514/6.2017-5255

Rodríguez, I., Bauer, A. S., Nottensteiner, K., Leidner, D., Grunwald, G., and Roa, M. A. (2021). "Autonomous robot planning system for in-space assembly of reconfigurable structures,"*IEEE Aerosp. Conf.* 50100. IEEE, 1–17. doi:10.1109/aero50100.2021.9438257

Schlenker, L., Moretto, M., Gaylor, D., and Linares, R. (2019). "Simultaneous localization and mapping for satellite rendezvous and proximity operations using random finite sets," in *AAS/AIAA space flight mechanics meeting*.

Thrun, S., and Montemerlo, M. (2006). The graph slam algorithm with applications to large-scale mapping of urban structures. *Int. J. Robotics Res.* 25, 403–429. doi:10.1177/0278364906065387

Ticozzi, L., and Tsiotras, P. (2025). Factor graph-based active slam for spacecraft proximity operations

Ting Goh, S., and Abdelkhalik, O. (2019). An introduction to kalman filtering implementation for localization and tracking applications. 143–195. doi:10.1002/9781119434610.ch5

Tweddle, B. E., Saenz-Otero, A., Leonard, J. J., and Miller, D. W. (2015). Factor graph modeling of rigid-body dynamics for localization, mapping, and parameter estimation of a spinning object in space. *J. Field Robotics* 32, 897–933. doi:10.1002/rob.21548

Xia, L., Cui, J., Shen, R., Xu, X., Gao, Y., and Li, X. (2020). A survey of image semantics-based visual simultaneous localization and mapping: application-Oriented solutions to autonomous navigation of Mobile robots. *Int. J. Adv. Robotic Syst.* 17, 1729881420919185. doi:10.1177/1729881420919185

Zhang, R., Cao, Z., Yang, S., Si, L., Sun, H., Xu, L., et al. (2025). Cognition-driven structural prior for instance-dependent label transition matrix estimation. *IEEE Trans. Neural Netw. Learn. Syst.* 36, 3730–3743. doi:10.1109/TNNLS.2023.3347633

Zhang, R., Tan, J., Cao, Z., Xu, L., Liu, Y., Si, L., et al. (2024). Part-aware correlation networks for few-shot learning. *IEEE Trans. Multimedia* 26, 9527–9538. doi:10.1109/TMM.2024.3394681

Zhang, R., Xu, L., Yu, Z., Shi, Y., Mu, C., and Xu, M. (2022). Deep-irtarget: an automatic target detector in infrared imagery using dual-domain feature extraction and allocation. *IEEE Trans. Multimedia* 24, 1735–1749. doi:10.1109/TMM.2021.3070138