IEEE Robotics and Automation Letters (RAL) paper presented at
2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Graph-Propagation-based Kinematic Algorithm for In-pipe Truss Structure Robots

Yu Chen, Jinyun Xu, Yilin Cai, Shuo Yang, Ben Brown, Fujun Ruan,
Yizhu Gu, Howie Choset, and Lu Li

***Abstract*—Robots designed for in-pipe navigation, inspection, and repair require flexibility for intricate pipeline traversal and the strength to carry payloads. However, conventional wheeled in-pipe robots face challenges in simultaneously achieving both substantial flexibility and payload-carrying capacity. A superior approach involves utilizing truss robots with redundant joints and linkages for pipe shape adaptation and actuation force distribution, providing significant advantages for complex pipeline navigation and heavy payload delivery. However, the kinematics of truss robots is computationally expensive for conventional Jacobian-based algorithms due to their complicated structural constraints. To address this limitation, we propose a novel algorithm for efficient truss-robot-kinematics computation using Graph Propagation (GP) method. Our method computes both forward kinematics and Jacobian in a propagative manner. It also guarantees geometric constraints with the Sigmoid function as the boundary. In simulation experiments, our algorithm accelerates pipe shape adaptation computation by 5.2∼16.4 times compared to finite difference methods. The practical feasibility of our method is assessed through physical in-pipe crawling experiments using a truss robot prototype. Additionally, the prototype's ability to carry heavy payloads is demonstrated through payload-carrying experiments, which results in 2∼4 times heavier payload capacity compared to two-wheeled robot approaches. We also showcase the versatility of proposed method in addressing manipulation tasks, indicating its generalizability across diverse applications. We believe this work could provide a unique algorithmic framework for truss robot kinematics formulation and computation, which will enable next generation of in-pipe robots to be more adaptive to complex environments and formidable toward real-world applications.**

***Index Terms*—Kinematics, Redundant Robots, In-pipe Robots, Truss Robots**

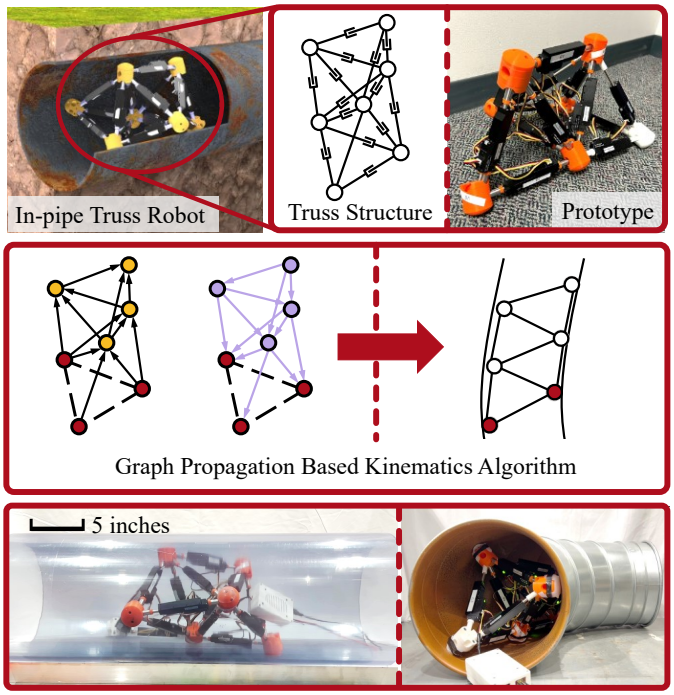

Fig. 1: A truss robot prototype is designed for in-pipe navigation and heavy payload delivery. Leveraging the proposed Graph Propagation based kinematic algorithm, the robot adapts its shape to fit the pipe shape and crawls inside pipelines.

I. Introduction

IN-PIPE robots are anticipated to perform navigation, inspection, and repair tasks in intricate pipelines. This requires the in-pipe robots to be flexible for adapting to pipes with different sizes, pipe elbows, and uneven inner surfaces [1]. Additionally, in-pipe robots need to possess the strength to carry substantial payloads, encompassing pipe-repair materials, mechanisms, various sensors, and tether systems. Existing in-pipe robots, including widely applied wheeled in-pipe robots, frequently struggle to concurrently achieve both of these objectives [2].

We present a novel approach utilizing tetrahedral modular truss robots for in-pipe crawling (shown in Fig. 1). Truss robots consist of repeating tetrahedral or octahedral modular structures with actuated edges and passive joints at the vertices. By adjusting the lengths of the edges, truss robots can change their configurations, which allows for a variety of shapes and an expanded workspace. The shape flexibility of in-pipe truss robots allows them to adapt to various pipe surfaces. Another advantage provided by in-pipe truss robot is the heavy payload carrying capability. Due to their load distribution among the edges for pure tensile or compressive forces, truss robots can withstand significant external loads while remaining a low body weight [3].

However, the intricate structure of truss robots introduces the challenge of addressing complex and time-consuming kinematics problems. Given that the truss robot comprises

Manuscript received: December 2, 2023 ; Revised January 29, 2024; Accepted March 15, 2024.

This paper was recommended for publication by Editor Jaydev P. Desai upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by the US Department of Energy Advanced Research Projects Agency-Energy (ARPA-E) under the Rapid Encapsulation of Pipelines Avoiding Intensive Replacement (REPAIR) program.

Authors are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA. uchen0223@gmail.com, lilu12@andrew.cmu.edu

Digital Object Identifier (DOI): see top of this page.

multiple parallel sub-structures, solving the kinematics of such a configuration necessitates considering the intricate relationships between the edges and vertices. This results in a significant amount of computation when solving the kinematics of truss robots, especially when dealing with a considerable large number of vertices.

To address this challenge, we propose a computationally efficient algorithm for solving the forward and inverse kinematics (IK) of a truss robot using the Graph Propagation (GP) method. GP is a technique that propagates information through a graph structure [4], [5]. In our approach, the robot is modeled as a graph, and the computation of kinematics is propagated from vertex to vertex through this graph. The utilization of this propagation approach effectively reduces the complexity of the kinematics algorithm to a linear scale relative to the number of vertices. Furthermore, our algorithm accounts for the robot's geometric constraints, including edge length, triangle face formation, and tetrahedral module formation constraints.

Our main contributions are summarized as follows:

- An efficient and generalizable Graph Propagation (GP)-based algorithm for solving truss robots' kinematics.
- A full-stack proof-of-concept prototype that proves the feasibility of the proposed algorithm, the flexibility of the truss structure in pipe shape adaptation, and the strength of the truss structure in payload carrying.

II. Related Work

We discuss previous kinematics algorithms of truss robots and review various in-pipe robot solutions in this section.

A. Truss Robot and its Kinematics Algorithms

Previous research on truss robots has primarily focused on manipulation tasks and ground crawling applications. Most truss robot manipulation research tends to focus on controlling the robot's end-point for path following [6]. Some studies have also explored the application of truss robots in object grasping, where the robot wraps itself around the object. In ground crawling scenarios, truss robots have been used as serpentine robots to traverse flat ground or steps [7], [8]. Alternatively, a pyramid-shaped truss robot [9] was designed to crawl on flat ground and a six-legged truss walker was used for locomotion on rough terrain [3].

The forward kinematics of truss robots can be computed analytically, while the IK is more challenging to solve. Tetrobot has proposed pseudo-inverse and virtual force methods to solve the IK of truss robots [3], [10]. In the pseudo-inverse method, [10] has employed a propagative approach for computing the robot's Jacobian. Their propagative approach differs from our GP method in two key aspects: First, Tetrobot computes the Jacobian using the same propagation order as forward kinematics, which requires assumptions about the motion of the 'starting face'. To remove the Jacobian computation errors resulting from these assumptions, Tetrobot further introduced an adjustments approach to fine-tune Jacobian matrix. In contrast, our method propagates from end-effectors along with the derivative chain, eliminating the need for such assumptions. Second, Tetrobot computes an analytic Jacobian that describes the relationship between edge and vertex velocities. In the context of in-pipe crawling, our method prioritizes the position of the vertices relative to the pipe walls over velocities. Therefore, our method employs a geometric Jacobian that directly relates the end-effector's position to the edge length. Although flexible and generalizable, the pseudo-inverse method is computationally expensive and does not account for edge length limits, which require post-processing to enforce these limits. The virtual force method, while better suited for handling vertex constraints, requires that all edges are extractable.

The truss robots IK can also be computed using iteratively univariate search [11]. In such approach, one edge is selected to extend by a given step size, and the process is iteratively repeated until the position error falls below a predetermined threshold. Additionally, truss robot kinematics can be solved using neural networks [12] or mesh optimization methods [9].

B. In-pipe Robots

Over the past years, various mechanisms and different types of locomotion have been employed in the design of in-pipe robots. These innovations aim to enhance pipeline traversing, payload capacity, speed, and lightweight design. The payload carrying ability and traversability are critical in many crucial tasks such as large-scale pipeline maintenance shown in [13], since the robot needs to traverse through intricate pipelines carrying heavy pipe repairing materials.

Wheeled in-pipe robots can apply pressure using their wheels to push against the inner wall of the pipe, allowing them to crawl in vertical pipes and adapt to varying pipe shapes. However, wheeled in-pipe robots have limited ability to carry heavy payloads [2]. Walking-type robots utilize articulated mechanisms for pipe navigation [14]. These walking mechanisms enable robots to effectively navigate through vertical pipes, overcome obstacles, and adapt to pipes with varying diameters. Nevertheless, the complex structure of walking mechanisms often results in bulky shapes, limiting their ability to navigate small and medium-sized pipes [15].

Several prior works have utilized truss structures to achieve inchworm locomotion while adapting to various pipe diameters. [16], [17] have employed tensegrity-based structures for in-pipe crawling. [18] employed a hydraulic octahedral burrowing robot for pipe elbow and branch crawling. These designs primarily focus on the scalability of the robot with limited payload-carrying capacity. For instance, the payload in Liu et al.'s work[16] is 0.5 kg, with body weight of 1.3kg.

III. GP-based Kinematics Algorithm

Consider a truss robot constructed by 4 serially-connected tetrahedral modules with 7 vertices as an example (shown in Fig. 2). Neighboring tetrahedral modules share the same triangle face. We will use $V^{(i)}$ to represent the ith vertex out of a total of N vertices. In our kinematics model, we define the first 3 vertices, $V^{(1)}$, $V^{(2)}$, and $V^{(3)}$, as the base vertices, which establish the reference frame for measuring the movements and positions of the robot's other components. In the truss robot,

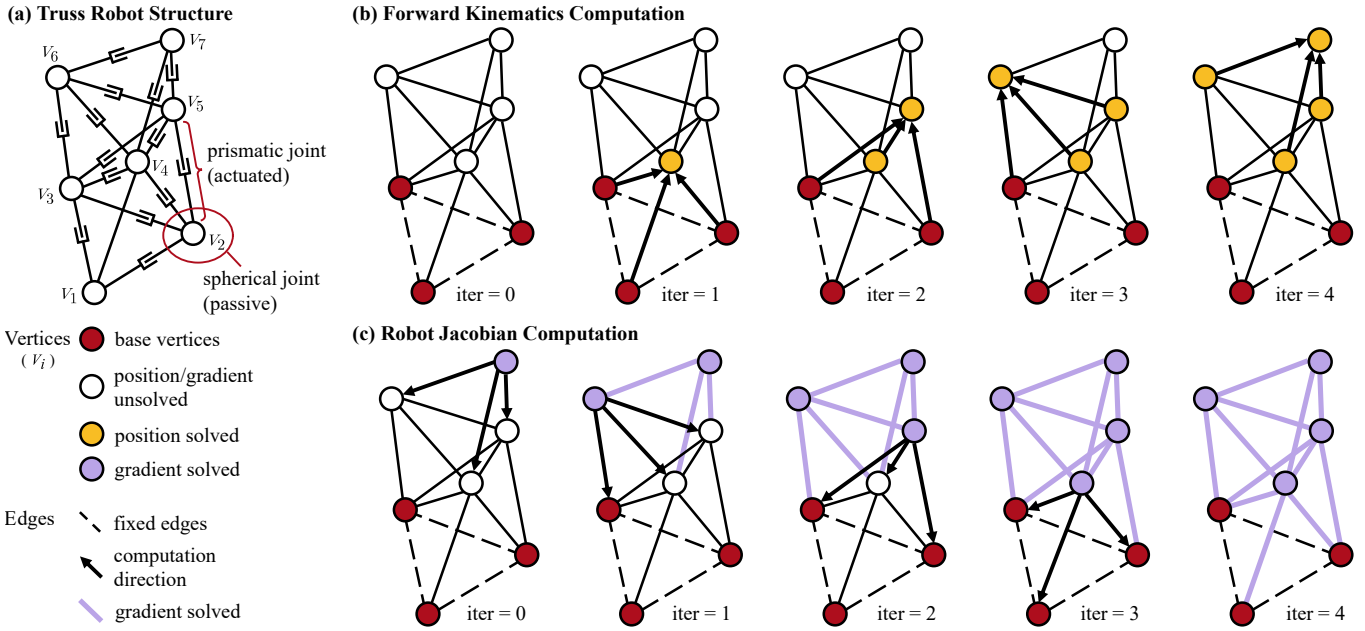

Fig. 2: Truss robot mechanism and the overall framework of the proposed kinematic algorithm. (a) The simplified diagram of the proposed truss robot, which can be modeled as the graph depicted in: (b) The iterative propagation process of forward kinematic computation, and (c) the iterative propagation process of robot Jacobian computation.

vertex $V^{(i)}$ ($i>4$) can form a tetrahedral module together with $V^{(i-1)}$, $V^{(i-2)}$, and $V^{(i-3)}$. The global position of vertex $V^{(i)}$ is represented as $\mathbf{v}^{(i)} = [x^{(i)}, y^{(i)}, z^{(i)}]^\mathsf{T}$. The edge connecting vertices $V^{(i)}$ and $V^{(j)}$ is denoted as $L^{(i,j)}$, with a length of $l^{(i,j)}$.

The joint space of the robot, denoted as $\mathbf{q}$, is defined as a vector containing the lengths of the robot's edges:

$$\mathbf{q} = [\, l^{(1,2)} \;\; l^{(1,3)} \;\; l^{(1,4)} \;\; \cdots \;\; l^{(N-2,N-1)} \;\; l^{(N-2,N)} \;\; l^{(N-1,N)} \,]^\mathsf{T}. \quad (1)$$

The task space $\mathbf{P}$ is defined as the positions that the robot vertices can reach. We can represent the task space as a matrix that contains the positions of all the vertices:

$$\mathbf{P} = [\, \mathbf{v}^{(1)} \;\; \cdots \;\; \mathbf{v}^{(i)} \;\; \cdots \;\; \mathbf{v}^{(N)} \,]. \quad (2)$$

To establish the relationship between the joint space $\mathbf{q}$ and the task space $\mathbf{P}$, we represent it as a graph in our method. In this graph, each vertex is associated with a vertex position $\mathbf{v}^{(i)}$, and each edge connecting $\mathbf{v}^{(i)}$ and $\mathbf{v}^{(j)}$ is weighted by the length $l^{(i,j)}$ (shown in Fig. 2). The elements in the joint space and task space can then be represented using the graph's edge set and vertex set, respectively.

A. Computation within One Module

We first consider the computation within one tetrahedral module, whose vertices are $V^{(1)}$, $V^{(2)}$, $V^{(3)}$, and $V^{(4)}$. The position of the last vertex $V^{(4)}$ is constrained with:

$$(\mathbf{v}^{(4)} - \mathbf{v}^{(i)})^\mathsf{T} (\mathbf{v}^{(4)} - \mathbf{v}^{(i)}) = {l^{(i,4)}}^2, \; i \in \{1,2,3\}, \quad (3)$$

$$((\mathbf{v}^{(2)} - \mathbf{v}^{(1)}) \times (\mathbf{v}^{(3)} - \mathbf{v}^{(1)})) \cdot (\mathbf{v}^{(4)} - \mathbf{v}^{(1)}) > 0. \quad (4)$$

Eq. (4) defines the surface normal of face $\mathbf{v}^{(1)}\mathbf{v}^{(2)}\mathbf{v}^{(3)}$ and ensures that $\mathbf{v}^{(4)}$ lies on the positive half-space defined by face $\mathbf{v}^{(1)}\mathbf{v}^{(2)}\mathbf{v}^{(3)}$ and the normal direction.

Eq. (3) and Eq. (4) can be transferred to an explicit expression that computes $\mathbf{v}^{(4)}$ given the $\mathbf{v}^{(i)}$ and $l^{(i,4)}$ ($i = 1,2,3$). We define this explicit expression as f. Except for the base vertices, the position of any other vertex $\mathbf{v}^{(i)}$ in the truss robot could be represented in a similar manner using f:

$$\mathbf{v}^{(i)} = f(\mathbf{v}^{(i-1)}, \mathbf{v}^{(i-2)}, \mathbf{v}^{(i-3)}, l^{(i-1,i)}, l^{(i-2,i)}, l^{(i-3,i)}), \quad (5)$$

where $i > 3$.

B. Forward Kinematics

Forward kinematics determines the positions of vertices $\mathbf{v}^{(i)}$ given joint space $\mathbf{q}$, which is defined by the lengths of the edges. The positions of the base vertices, $\mathbf{v}^{(1)}$, $\mathbf{v}^{(2)}$, and $\mathbf{v}^{(3)}$, are predetermined either manually or by the robot gait.

Algorithm 1 Forward Kinematics Computation

```
Given joint space q
Predetermine v(1), v(2), and v(3)
for i = 4,5,6,...,N do                    ▷ GP method
    Compute v(i) via Eq. (5)
end for
return v(1), v(2), ..., v(N)
```

The computation of the positions for the remaining vertices follows a topological order: starting from the base vertices and propagating towards the endpoint. This order flows from $\mathbf{v}^{(4)}$ to $\mathbf{v}^{(N)}$ (shown in Fig. 2(b)). As explained in Section III-A, $\mathbf{v}^{(4)}$ can be determined using Eq. (5), which relies on $\mathbf{v}^{(1)}$, $\mathbf{v}^{(2)}$, and $\mathbf{v}^{(3)}$. Once $\mathbf{v}^{(4)}$ is calculated, the next vertex, $\mathbf{v}^{(5)}$, can be determined by utilizing $\mathbf{v}^{(2)}$, $\mathbf{v}^{(3)}$, and $\mathbf{v}^{(4)}$. This iterative process is applied repeatedly to compute $\mathbf{v}^{(6)}$, $\mathbf{v}^{(7)}$, and the remaining vertex positions.

The pseudo code of the forward kinematics is shown in Algorithm 1.

C. Inverse Kinematics

IK aims to determine the optimal joint space $\mathbf{q}^*$ for the robot given the desired position of the vertices $\mathbf{v}^*_{\text{end}}$. In this case, we employ the Jacobian inverse technique to solve the problem. The IK of the truss robot can be viewed as an optimization problem. First, we quantify the deviation between the current vertices position $\hat{\mathbf{v}}_{\text{end}}$ and the goal position $\mathbf{v}^*_{\text{end}}$ as the vertices positioning error E. Then, the configuration of the truss robot can then be iteratively optimized by minimizing this error E.

During the optimization process of the robot's configuration, the joint space must take into account specific constraints to ensure the practicality of the robot's mechanical and topological limits. Three types of constraints need to be considered: edge length constraint, triangle face formation constraint, and tetrahedron module formation constraint.

The edge length constraint ensures that the lengths of the edges fall within a certain range of values, thereby ensuring the match with the actual robot actuator characteristics. In our approach, we enforce this constraint using a barrier function:

$$l^{(i,j)} = l^{(i,j)}_{\min} + (l^{(i,j)}_{\max} - l^{(i,j)}_{\min})\sigma(\omega^{(i,j)}), \tag{6}$$

where $l^{(i,j)}_{\min}$ and $l^{(i,j)}_{\max}$ represent the minimum and maximum lengths of the edge, respectively, and slack variable $\omega^{(i,j)} \in \mathbb{R}$. σ is the Sigmoid function that maps the slack variable into the range of $(0,1)$:

$$\sigma(\omega^{(i,j)}) = 1/(1+e^{-\omega^{(i,j)}}) \in (0,1). \tag{7}$$

Therefore, the range of $l^{(i,j)}$ is constrained within $(l^{(i,j)}_{\min}, l^{(i,j)}_{\max})$. In the subsequent calculations, we will substitute $l^{(i,j)}$ with $\omega^{(i,j)}$, resulting in the representation of the joint space as ω instead of $\mathbf{q}$:

$$\omega = [\omega^{(1,2)}\, \omega^{(1,3)} \cdots \omega^{(N-2,N-1)}\, \omega^{(N-2,N)}\, \omega^{(N-1,N)}]^{\mathsf{T}}. \tag{8}$$

Since the joint space $\mathbf{q}$ is already bounded within the maximum and minimum edge lengths using the barrier function (6), we only need to focus on the constraints related to triangle and tetrahedron formation. In our proposed technique, these constraints are satisfied by constraining the values of $l^{(i,j)}_{\min}$ and $l^{(i,j)}_{\max}$ (proof shown in Appendix):

$$l^{(i,j)}_{\min} \le l^{(i,j)}_{\max} \le \sqrt{3}\, l^{(i,j)}_{\min}. \tag{9}$$

After substituting $\mathbf{q}$ with ω and imposing the constraints on the parameters $l^{(i,j)}_{\min}$ and $l^{(i,j)}_{\max}$, the IK problem is then transformed into an unconstrained optimization problem, which can be considered as finding the optimal values of $\omega \in \mathbb{R}$ to minimize the vertices positioning error E:

$$\min_{\omega \in \mathbb{R}} \quad E(\hat{\mathbf{v}}_{\text{end}}, \mathbf{v}^*_{\text{end}}). \tag{10}$$

To numerically solve for the minimum value of E, we shall obtain the Jacobian of the robot and then iteratively compute an approximation of the difference between the optimized joint space and the initial joint space. In our approach, we solve the Jacobian by implementing the GP method from endpoint to base vertices (shown in Fig. 2(c)). The gradient of $\omega^{(i,j)}$ with respect to E is calculated as follows:

$$\frac{\partial E}{\partial \omega^{(i,j)}} = \frac{\partial E}{\partial \mathbf{v}^{(j)}} \frac{\partial \mathbf{v}^{(j)}}{\partial l^{(i,j)}} \frac{\partial l^{(i,j)}}{\partial \omega^{(i,j)}}, \tag{11}$$

where

$$\frac{\partial \mathbf{v}^{(j)}}{\partial l^{(i,j)}} = \frac{\partial f}{\partial l^{(i,j)}}, \tag{12}$$

$$\frac{\partial l^{(i,j)}}{\partial \omega^{(i,j)}} = (l^{(i,j)}_{\max} - l^{(i,j)}_{\min})\,(1-\sigma(\omega^{(i,j)}))\,\sigma(\omega^{(i,j)}). \tag{13}$$

$\partial E/\partial \mathbf{v}^{(j)}$ is the gradient of the later endpoint of $l^{(i,j)}$ with respect to E, which could be calculated as follows:

$$\frac{\partial E}{\partial \mathbf{v}^{(j)}} = \sum_{k=0,1,2,3; j+k\le N} \frac{\partial E}{\partial \mathbf{v}^{(j+k)}} \frac{\partial \mathbf{v}^{(j+k)}}{\partial \mathbf{v}^{(j)}}. \tag{14}$$

Our GP method calculates $\partial E/\partial \mathbf{v}^{(N)}$ in a dynamic programming manner, which restores and reuses the computed gradients. After solving $\partial E/\partial \mathbf{v}^{(N)}$, we can compute $\partial E/\partial \omega^{(N-q,N)}$ $(q = 1,2,3)$ using Eq. (11) and $\partial E/\partial \mathbf{v}^{(N-1)}$ using Eq. (14). With $\partial E/\partial \mathbf{v}^{(N-1)}$ solved, $\partial E/\partial \omega^{(N-1-q,N-1)}$ $(q = 1,2,3)$ and $\partial E/\partial \mathbf{v}^{(N-2)}$ can also be solved. This process is repeated until all the gradients of $\omega^{(i,j)}$ with respect to E are computed (see Algorithm 2). The Jacobian computation only needs to traverse each module once, and the algorithm, therefore, exhibits a complexity of $O(N)$.

Algorithm 2 Robot Jacobian Computation

```
Given vertices positioning error E
for j = N, N−1, ...3, 2, 1 do                    ▷ GP method
    Compute ∂E/∂v^(j) via Eq. (14).
    for i = j−1, j−2, j−3 do
        if i > 0 then
            Compute ∂E/∂ω^(i,j) via Eq. (11).
        end if
    end for
end for
return ∂E/∂ω
```

IV. Shape Optimization and In-Pipe Crawling Gait

In the scenario of in-pipe crawling, the entire robot body is capable of free movement, and the base vertices are considered floating bases. We use three virtual base vertices as the reference of the robot frame, which are connected to the floating base vertices by virtual edges (shown in Fig. 3).

We first randomly initialize or predetermine the positions of virtual base vertices. Then we optimize the positions of other vertices using the algorithm outlined in section III to situate all vertices on the pipe's inner surface. The vertices positioning error E with respect to vertex i is defined as:

$$E^{(i)} = |\, s(x^{(i)}, y^{(i)}, z^{(i)})\,|, \tag{15}$$

where $[x^{(i)}, y^{(i)}, z^{(i)}]^{\mathsf{T}}$ denotes the position of vertex i and the pipe inner surface is defined by $s(x,y,z) = 0$.

Subsequently, we allow the virtual base vertices to move forward along the pipe centerline for a small constant distance. Afterward, we again optimize the robot's configuration to adapt to the pipe's inner surface. Following this manner, the entire robot moves forward along the pipe for one step (shown in Fig. 3).

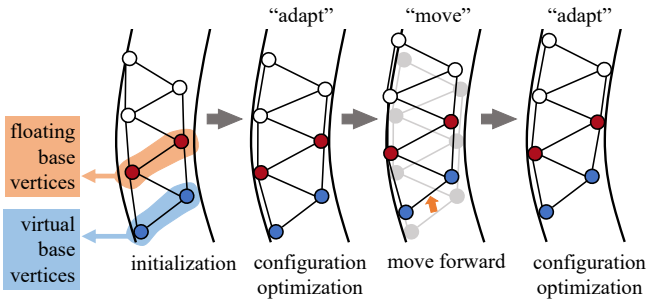

Fig. 3: Illustration of the procedure of robot gait computation. The robot is a 2D abstraction of the actual 3D robot graph for better visualization. The robot crawls in pipes following a worm-like adapt-move-adapt motion. In practical applications, the positions of the vertices are updated in a sequential manner.

However, it is not feasible for the robot to move all of its vertices simultaneously in practical scenarios. Therefore, building upon the optimized robot configuration, we update the position of each vertex one by one in a sequential manner. Given that the step size is small, we neglect the risk of breaking the constraints in (9) when sequentially updating the vertex positions.

V. Experiments

This section encompasses software simulations and real-world physical experiments. We first evaluate the pipe adaptation ability of the structure and algorithm through simulations. Then, we benchmarked the efficiency of the GP-based algorithm in end-point placement simulations. Additionally, two examples of end-point path following simulations are presented to show the generalizability of the proposed algorithm to manipulation scenarios. To further validate our approach, a physical prototype of the truss robot system is introduced, which enables physical experiments to evaluate the truss robot's mobility in straight pipes and pipe elbow structures, all facilitated by the proposed algorithm. Payload-carrying experiments are finally conducted to highlight the robot's ability to provide high pull force. Detail about our algorithm and video demonstration are available at `https://github.com/Neuling-jpg/Earthworm`.

A. Simulation for Pipe Adaptation Ability Evaluation

In the pipe shape adaptation simulation, the truss structure's ability to fit into different pipe shapes is evaluated by assessing the vertices' positioning error E with respect to varying pipe diameters. For simplicity, the robot is abstracted to a 3D point-and-line structure in our simulations: The edges are represented as line segments with length $l^{(i,j)}$, and the volume occupied by vertices is neglected. The minimum and maximum lengths of the edges are set to 9 and 11 inches, respectively. The pipe diameters vary between 8.8 and 13.2 inches to align with the potential diameter range of a natural gas distribution pipes, mainly with a 12-inch pipe diameter. The pipe turning radius R is sampled from $\{10, 13, 15, 20, \infty\}$ inches, where $R = \infty$ corresponds to straight pipes.

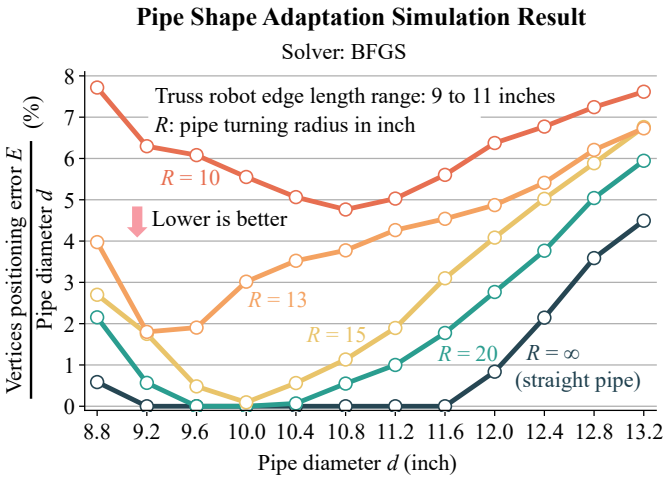

Fig. 4: Pipe shape adaptation simulation result. y-axis indicates the ratio of vertices positioning error E over pipe diameter d. x-axis indicates the pipe diameter d ranging from 8.8 to 13.2 inches. The relative errors are obtained under different conditions of the pipe turning radius R (10 to ∞ inches), which indicates the pipe shape adaptation ability.

The inner surface of the straight pipe, where $R = \infty$, is represented as follows:

$$s_{\text{straight}} = \sqrt{x^2 + y^2} - \frac{d}{2}, \tag{16}$$

and the inner surface of the pipe elbow is modeled as a torus, whose equation is:

$$s_{\text{elbow}} = \sqrt{\left(\sqrt{x^2 + y^2} - R\right)^2 + z^2} - \frac{d}{2}, \tag{17}$$

where d is the diameter of the pipe and R is the turning radius of the elbow.

TABLE I: Pipe Shape Adaptation Runtime Result

R		10	13	15	20	∞
Runtime (ms)	w/o GP	1098.4	573.3	695.2	1233.8	393.1
	w/ GP	67.0	66.0	51.0	118.0	76.0
Speed improvement ratio		16.4	8.7	13.6	10.5	5.2

$*$ Speed improvement ratio: the ratio of runtime w/o GP to runtime w/ GP

Given Eq. (16) and (17), the vertices positioning error E is computed using Eq. (15). We use GP method to compute Jacobian, and Broyden–Fletcher–Goldfarb–Shanno (BFGS) [19] to optimize the configuration of the robot.

The results of the pipe shape adaptation simulation are depicted in Fig. 4. The positioning error value E, which indicates the proposed method's ability to adapt to the pipe shape, exhibits notable variations corresponding to different pipe turning radii R and pipe diameters d. The results of E, although vary significantly under different simulation setups, still demonstrate a small positioning error for the vertices, which is less than 8% of the pipe diameter. These results prove that the proposed structure and algorithm are capable of adapting to pipes with different diameters and turning radius.

We then set the pipe diameter to 10 to showcase the runtime of solving kinematics for pipe shape adaptation with or without GP (see Table I). The speed ratio represents the ratio of runtime without GP to runtime with GP. The results show that the GP approach accelerates the speed of solving kinematics by a factor ranging from 5.2 to 16.4 times. This indicates

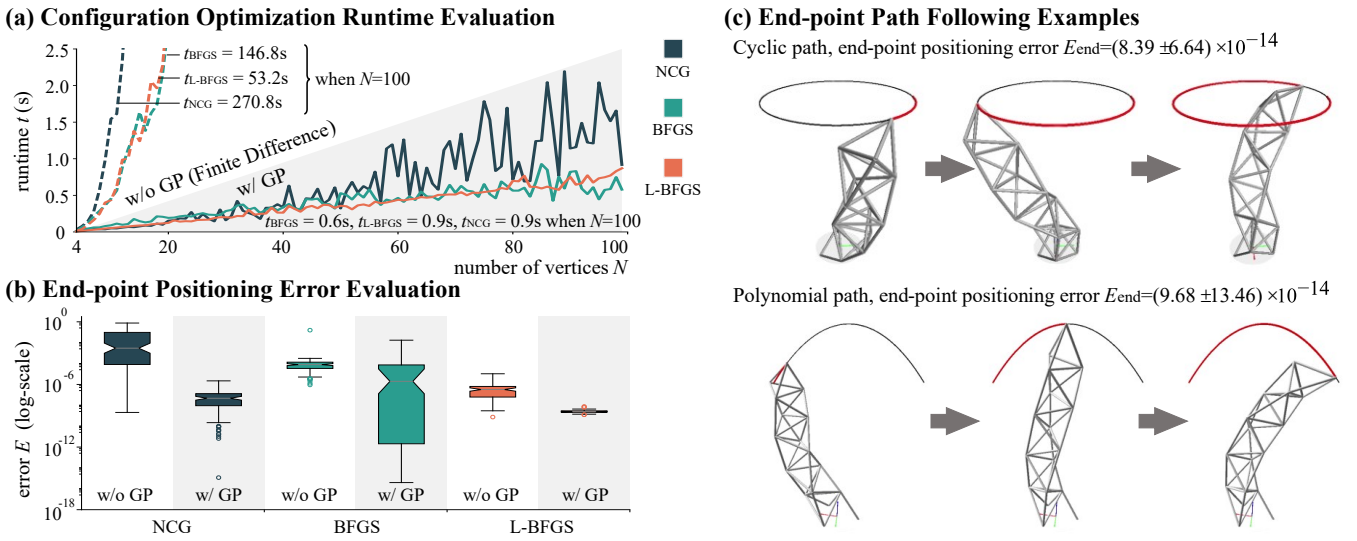

Fig. 5: (a) The runtime t of configuration optimization solved by NCG, BFGS, and L-BFGS with respect to the number of vertices N, with and without GP. (b) Boxplot of the vertices' positioning error E, with and without GP. (c) Two examples of end-point path following tasks using the proposed GP based kinematic algorithm.

that the use of the GP approach also brings a significant acceleration in solving kinematics for pipe shape adaptation. The efficiency of the algorithm will be detailed in the next section.

B. End-point Placement Simulation for Algorithm Efficiency and Generalizability Evaluation

End-point placement simulations are performed in this section to assess the efficiency of the algorithm and demonstrate its generalizability in manipulation scenarios. End-point placement tasks are commonly used in literature on truss robots to evaluate the feasibility of kinematics or dynamics algorithms [20], [21] by requiring the truss robot to position its end vertex at a predetermined location or follow a path while keeping the base vertices fixed.

Given a truss robot with N vertices, we first require the end vertex to move a distance of one unit length along the positive direction of the z-axis. The positioning error of the end vertex E_{end} is defined as:

$$E_{\text{end}} = ||\,\mathbf{v}^{(N)} - (\mathbf{v}^{(N)}_{\text{origin}} + [0,0,1]^{\mathsf{T}})\,||_2, \tag{18}$$

where $\mathbf{v}^{(N)}$ and $\mathbf{v}^{(N)}_{\text{origin}}$ represent the current and original position of vertex $V^{(N)}$ respectively. Three solvers, Newton Conjugate-Gradient (NCG) [22], BFGS, and Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) [23], are utilized to solve IK.

We evaluate the computation time required to reach the target point in the end-point placement task for different numbers of vertices, ranging from 4 (as the minimum number of vertices needed to form a single tetrahedral module) to 100 (as the stress test condition). All tests are conducted on a Windows PC with AMD Ryzen 7 5800H processor.

Both BFGS and L-BFGS, the two fastest solvers in the experiments, exhibit a runtime increase that approximately scales linearly with the number of vertices (shown in Fig. 5(a)). Compared to the methods without GP, which instead approximate Jacobian with finite difference [24], the incorporation of GP for Jacobian computation results in a significant improvement in algorithm speed. For example, the utilization of the GP method increased the computation speed by approximately 5∼31 times when $N = 10$, 59∼300 times when $N = 20$, and 71∼125 times in the extreme case of $N = 100$. This confirmed that the utilization of GP method can efficiently optimize the robot's configuration. Surprisingly, empirical results show that employing GP for Jacobian computation yields comparable errors, or even a reduced error as small as $1/10^4$, compared to results obtained without GP (shown in Fig. 5(b)). One potential explanation could be the proposed GP algorithm prevents the discretization error introduced by gradient approximation using finite difference. It is worth noting that we did not compare our method with the pseudo-inverse method. The reason for this omission is that the pseudo-inverse method cannot guarantee edge length constraints, potentially leading to infeasible solutions.

To further demonstrate the generalizability of the GP-based algorithm, we provide two examples of end-point path following tasks (shown in Figure 5(c)). In these examples, a truss robot with 14 vertices is utilized to track both a cyclic and a polynomial path. The cyclic path is defined as $\{x^2 + y^2 = 15^2,\ z = 40\}$, while the polynomial path is defined as $\{z = -0.04\,x^2,\ |x| \le 18,\ y = 0\}$. For both tests, the end-point positioning errors are small, measuring $(8.39 \pm 6.64) \times 10^{-14}$ and $(9.68 \pm 13.46) \times 10^{-14}$ [simulation unit length], respectively. These two examples in manipulation demonstrate the feasibility of extending the proposed algorithms to other robotic applications using the modular truss structure.

C. Hardware Experiments

To further validate the truss structure concept and evaluate the proposed kinematic algorithm, we designed and built a physical robot prototype. The prototype consists of 4 tetrahedral modules with 7 vertices and 15 edges (shown in Fig.

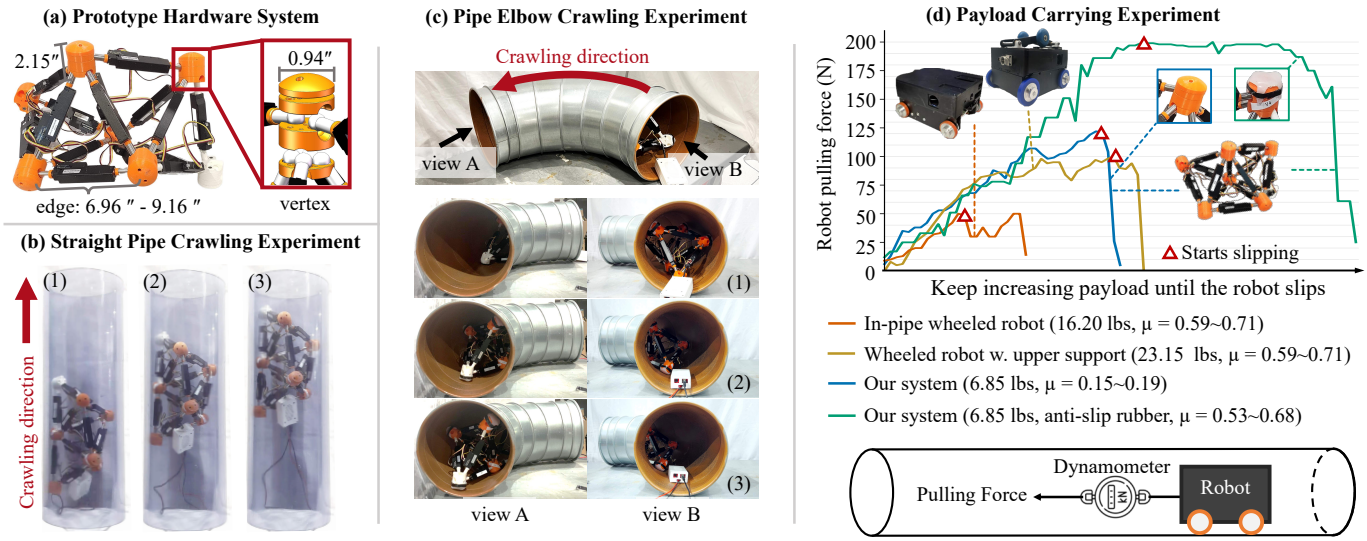

Fig. 6: (a) Picture of the overall structure of the prototype and the exploded view of its vertex. (b) - (c) Example video frames from vertical straight pipe, and pipe elbow crawling experiments. (d) Payload carrying capabilities comparison between our system and an in-pipe wheeled robot. Our system, despite having a lighter weight and a smaller friction coefficient, is able to provide a maximum pull force significantly greater than the wheeled robots, with or without upward pusher wheels support.

6(a)). The prototype weighs 6.85 lbs. Each edge is actuated by a linear actuator (PA-12-R Micro Precision Servo Actuator) with a stroke length of 2.2 inches and a maximum extension speed of 0.3 inch/s. The length of the edges can be adjusted between 6.96 and 9.16 inches, depending on the controlled extension or contraction of the actuators. The rated force of the actuators is 17.53 lbs. The vertices are 3D printed and connect the edges with ball-and-socket joints. The height and diameter of the vertices are 2.15 and 0.94 inches, respectively. Each edge has two ball-shaped tips that can be held by the spherical pits to form a spheroid joint. The rotation space of the edge relative to the spheroid joint lies within a cone with a top angle of 26°. In the current design, each vertex can hold a maximum of 6 edges. The kinematics, in-pipe crawling gait, and corresponding actuator extension/contraction commands are computed offline using Python assuming the pipe is manoeuvrable with a pre-known shape and size. Then, an Arduino Mega 2560 is used for online open-loop robot control.

Experiments on straight pipe and pipe elbow crawling were conducted. For straight pipe crawling, we use a PVC pipe with an inner diameter of 12 inches and a length of 37.4 inches. The selected video frames demonstrate that the robot is capable of crawling in vertical pipes (shown in Fig. 6(b)). The pipe elbow used in the experiment is a 90-degree metal elbow with a turning radius of 18 inches with a pipe diameter of 12 inches. The robot successfully crawled through the pipe elbow from one end to the other (shown in Fig. 6(c)). The average linear speed is approximately 0.3 inch/min. In straight pipe crawling experiments, the actuators exhibit length variations of approximately 6.5% to 15.2% relative to their stroke range. While in pipe elbow crawling experiments, this ratio ranges from 8.7% to 10.8%. Moreover, during straight pipe crawling, 13.3% of the actuators have reached their minimum available length, while none have reached their maximum. During pipe elbow crawling, the percentages are 20% and 6.7% respectively, for actuators reaching their minimum and maximum available lengths. The payload-carrying ability of the truss robot prototype was assessed and compared with a baseline wheeled in-pipe robot. By incrementally adding payload until the robot experienced slipping, we measured the maximum pull force provided by 3 systems: baseline in-pipe wheeled robot with anti-slip rubber wheels, our system without anti-slip mechanisms, and our system with anti-slip rubber glued on the nodes (shown in Fig. 6(d)). The friction coefficients (μ) between each of these systems and the PVC pipe were evaluated using the block-on-block method [25].

TABLE II: Pull Force Comparison Results

System	μ	$F_{\max}$ (N)	$F_{\max}/M$
Wheeled robot w/o upper support	0.59∼0.71	50	0.68
Wheeled robot w/ upper support	0.59∼0.71	107	1.02
Our system w/o anti-slip rubber	0.15∼0.19	122	3.92
Our system w/ anti-slip rubber	0.53∼0.68	**200**	**6.43**

The maximum pull force of the robots $F_{\max}$ and the pull force to weight ratio $F_{\max}/M$ are shown in Table II. The prototype demonstrated the capability to carry a payload of 6.43 times its own weight. Notably, our system's pull force significantly outperforms our two baselines: in-pipe wheeled robot and wheeled robot with upper wheel support, despite our system being 2.36∼3.38 times lighter. Even in the absence of any anti-slip mechanisms, our system achieved a pull force of 122N, which is 3.92 times its own weight, 2.44 times that of the wheeled robot, and 1.14 times that of the upper-supported wheeled robot.

VI. Conclusion & Future Works

We present a computationally efficient algorithm for solving truss robots' kinematics. A physical prototype is designed

and built to demonstrate its crawling capabilities in both straight pipes and pipe elbows, as well as its payload-carrying ability. Additionally, simulation results highlight the significant improvement in Jacobian computation efficiency achieved by the proposed GP-based kinematics algorithm. We further demonstrated that the algorithm can be generalized to other applications, such as manipulation tasks.

Several exciting future research directions have been identified for the presented algorithm and truss robot design. Firstly, additional constraints related to the in-pipe crawling environment can be considered during the gait generation. One specific is the workspace constraint associated with non-vertex contacts in conditions such as pipe bends, in-pipe extrusions, obstacles, or pipe joints. Secondly, the slow extension speed of the existing commercial linear actuators significantly limits the crawling speed of the prototype presented in this paper. We also believe that the robot's navigation ability can be greatly enhanced by designing and utilizing more potent and faster actuators. Thirdly, we believe that our algorithm can be applied to online gait planning with the help of additional perception units and close-loop control scheme. Lastly, testing on more diverse and challenging pipe shapes, including T-shape pipe elbow, is necessary for further applications of our method.

In conclusion, the presented work signifies a fundamental phase in the development of truss-structure in-pipe robots and can be further applied to address other robotic challenges in complex and confined spaces, including hazardous environment exploration and research-and-rescue operations.

APPENDIX

We prove that the constraint in Eq. (9) sufficiently satisfy both the triangle and the tetrahedron formation constraint.

Triangle faces formation constraints: All triangles comprising the robot should satisfy the triangle inequality. For a triangle with 3 sides l_1, l_2, and l_3, assume $l_1 \leq l_2 \leq l_3$. Then l_3 should be no longer than $2l_1$. Given Eq. (9), stating that l_3 is less or equal to $\sqrt{3}l_1$, this constraint is satisfied.

Tetrahedron modules formation constraints: For each tetrahedral module in the robot, the sum of the areas of any three triangle faces to be greater than the area of the fourth face. For a tetrahedron whose size of its 4 faces are s_1, s_2, s_3, and s_4, assume $s_1 \leq s_2 \leq s_3 \leq s_4$. The size of its faces should meet the following constraint that $s_1 + s_2 + s_3 \geq s_4$.

Consider a triangle whose maximum value of three sides are no more than $\sqrt{3}$ times the minimum value. Then, the maximum size of the triangle $s_{\max}$ is less or equal to 3 times its minimum size $s_{\min}$. Given Eq. (9), we have

$$\min_{\forall s_{\min} \in \mathbf{S}_{\min}} 3s_{\min} \geq \max_{\forall s_{\max} \in \mathbf{S}_{\max}} s_{\max} \tag{19}$$

where $\mathbf{S}_{\max}$ and $\mathbf{S}_{\min}$ are the set of all the possible minimum and maximum size of the faces in the tetrahedron. Since $s_1 + s_2 + s_3 \geq \min_{\forall s_{\min} \in \mathbf{S}_{\min}}(3 \times s_{\min})$ and $s_4 \leq \max_{\forall s_{\max} \in \mathbf{S}_{\max}} s_{\max}$, we meet the constraints $s_1 + s_2 + s_3 \geq s_4$.

REFERENCES

[1] S.-g. Roh and H. R. Choi, "Differential-drive in-pipe robot for moving inside urban gas pipelines," *IEEE transactions on robotics*, vol. 21, no. 1, pp. 1–17, 2005.

[2] J. T. Kahnamouei and M. Moallem, "A comprehensive review of in-pipe robots," *Ocean Engineering*, vol. 277, p. 114260, 2023.

[3] G. J. Hamlin and A. C. Sanderson, "Tetrobot: A modular approach to parallel robotics," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 42–50, 1997.

[4] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

[5] M. Kampffmeyer, Y. Chen, X. Liang, H. Wang, Y. Zhang, and E. P. Xing, "Rethinking knowledge graph propagation for zero-shot learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11 487–11 496.

[6] S. Utku, S. Das, B. Wada, G. Chen, and A. Ramesh, "Control of a slow-moving space crane as an adaptive structure," *AIAA journal*, vol. 29, no. 6, pp. 961–967, 1991.

[7] G. S. Chirikjian and J. W. Burdick, "The kinematics of hyper-redundant robot locomotion," *IEEE transactions on robotics and automation*, vol. 11, no. 6, pp. 781–793, 1995.

[8] A. Spinos, D. Carroll, T. Kientz, and M. Yim, "Variable topology truss: Design and analysis," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 2717–2722.

[9] N. Usevitch, Z. Hammond, S. Follmer, and M. Schwager, "Linear actuator robots: Differential kinematics, controllability, and algorithms for locomotion and shape morphing," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 5361–5367.

[10] G. J. Hamlin and A. C. Sanderson, "Tetrobot: a modular system for hyper-redundant parallel robotics," in *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, vol. 1. IEEE, 1995, pp. 154–159.

[11] S. Jain and S. Kramer, "Forward and inverse kinematic solution of the variable geometry truss robot based on an n-celled tetrahedron-tetrahedron truss," 1990.

[12] K. E. Zanganeh and P. C. Hughes, "Fast estimation of the kinematics of the parallel modules of a variable-geometry-truss manipulator using neural networks," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*, vol. 2. IEEE, 1998, pp. 1665–1670.

[13] "Arpa-e repair," Aug 2021. [Online]. Available: https://www.ri.cmu.edu/project/arpa-e-repair/

[14] S. Savin, S. Jatsun, and L. Vorochaeva, "Trajectory generation for a walking in-pipe robot moving through spatially curved pipes," in *MATEC Web of Conferences*, vol. 113. EDP Sciences, 2017, p. 02016.

[15] A. Verma, A. Kaiwart, N. D. Dubey, F. Naseer, and S. Pradhan, "A review on various types of in-pipe inspection robot," *Materials Today: Proceedings*, vol. 50, pp. 1425–1434, 2022.

[16] Y. Liu, X. Dai, Z. Wang, Q. Bi, R. Song, J. Zhao, and Y. Li, "A tensegrity-based inchworm-like robot for crawling in pipes with varying diameters," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 553–11 560, 2022.

[17] Y. Liu, Q. Bi, X. Dai, R. Song, X. Zang, and Y. Li, "An in-pipe crawling robot based on tensegrity structures," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 13 248–13 253.

[18] J. C. Zagal, C. Armstrong, and S. Li, "Deformable octahedron burrowing robot," in *Artificial Life Conference Proceedings*. Citeseer, 2012, pp. 431–438.

[19] R. Fletcher, *Practical methods of optimization*. John Wiley & Sons, 2013.

[20] W. H. Lee and A. C. Sanderson, "Dynamic analysis and distributed control of the tetrobot modular reconfigurable robotic system," *Autonomous Robots*, vol. 10, pp. 67–82, 2001.

[21] G. S. Chirikjian, *Theory and applications of hyper-redundant robotic manipulators*. California Institute of Technology, 1992.

[22] R. S. Dembo and T. Steihaug, "Truncated-newtono algorithms for large-scale unconstrained optimization," *Mathematical Programming*, vol. 26, no. 2, pp. 190–212, 1983.

[23] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, vol. 45, no. 1-3, pp. 503–528, 1989.

[24] J. C. Strikwerda, *Finite difference schemes and partial differential equations*. SIAM, 2004.

[25] E. Rabinowicz, "The nature of the static and kinetic coefficients of friction," *Journal of applied physics*, vol. 22, no. 11, pp. 1373–1379, 1951.