Locomotion of Linear Actuator Robots Through Kinematic Planning and Nonlinear Optimization

Nathan S. Usevitch, Zachary M. Hammond, and Mac Schwager

***Abstract*—In this article consider a class of robotic systems composed of high-elongation linear actuators connected at universal joints. We derive the differential kinematics of such robots, and show that any instantaneous velocity of the nodes can be achieved through actuator motions if the graph describing the robot's configuration is infinitesimally rigid. We formulate physical constraints that constrain the maximum and minimum length of each actuator, the minimum distance between unconnected actuators, the minimum angle between connected actuators, and constraints that ensure the robot avoids singular configurations. We present two planning algorithms that allow a linear actuator robot to locomote. The first algorithm repeatedly solves a nonlinear optimization problem online to move the robot's center of mass in a desired direction for one time step. This algorithm can be used for an arbitrary linear actuator robot but does not guarantee persistent feasibility. The second method ensures persistent feasibility with a hierarchical coarse-fine planning decomposition, and applies to linear actuator robots with a certain symmetry property. We compare these two planning methods in simulation studies.**

***Index Terms*—Kinematics, motion and path planning, optimization and optimal control, truss robots.**

I. Introduction

In this article, we present a control methodology for robots made up of high-elongation linear actuators connected together at universal joints, which we call linear actuator robots (LARs). Such robots can change their shape dramatically through the coordinated actuation of their linear members. The control of robots with a large number of degrees of freedom is important to allow robots to become increasingly flexible to a wide variety of tasks. In the case of an LAR, the robot can change shape to be better suited for a multitude of tasks, including locomotion, manipulation, and matching of 3-D target shapes (shape morphing). Such a robot would be valuable in search and rescue missions, where an LAR could flexibly maneuver over uneven terrain and morph into a custom manipulator to clear debris. LARs can also serve as a type of "programmable matter," changing shape to represent 3-D objects and responding to a human designer's digital manipulations in real time.

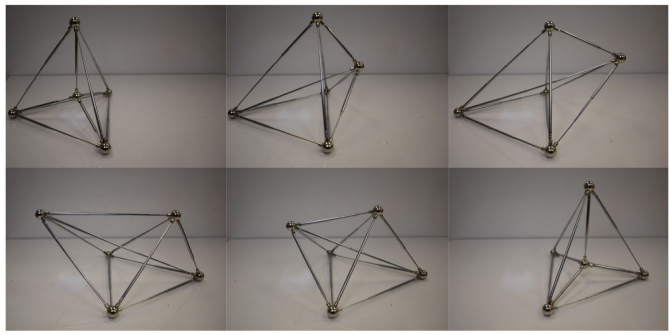

Fig. 1. Two algorithms are presented for LARs to locomote. This figure shows snapshots of an optimized everting gait for an LAR with ten edges and five vertices, computed with one of our algorithms. The LAR shown is a passive mockup made from ten car antennas, and hand-positioned to illustrate the gate.

In this work, we present nonlinear optimization techniques to enable an LAR to locomote. We present a differential kinematic analysis of LARs, relating the velocities of the nodes in the structure to the rate of change of the actuator lengths. This allows us to link concepts from graph rigidity to the control of the robot structure. We use this kinematic analysis to derive two on-line planning algorithms for locomotion, which are both based on the same underlying nonlinear optimization algorithm tailored to the kinematics and constraints of LARs. A passive mock-up of an LAR executing one of the optimized locomotion trajectories presented in this article is shown in Fig. 1. In this case, the robot is composed of ten actuators (passive car antenna elements) and five nodes (with spherical joints formed by magnets attached to steel balls).

A. Related Work

To best understand the related work, we first present an overview of different implementations and applications of LARs, and then discuss in detail the specific control methodologies. Early work on TETROBOTs proposed robots composed of

Manuscript received September 11, 2019; revised February 10, 2020; accepted April 5, 2020. Date of publication June 2, 2020; date of current version October 1, 2020. This work was supported in part by DARPA YFA under Award D18AP00064 and in part by the National Science Foundation under Award 1637446. This article was recommended for publication by Associate Editor R. Hatton and Editor M. Yim upon evaluation of the reviewers' comments. *(Corresponding author: Nathan S. Usevitch.)*

Nathan S. Usevitch and Zachary M. Hammond are with the Department of Mechanical Engineering, Stanford University, Stanford, CA 94305 USA (e-mail: usevitch@stanford.edu; zhammond@stanford.edu).

Mac Schwager is with the Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305 USA (e-mail: schwager@stanford.edu).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. The material consists of a video, with no audio track, demonstrating animations of the motion of linear actuator robots. The size of the video is 18.5 MB. Contact usevitch@stanford.edu for further questions about this work. Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2020.2995067

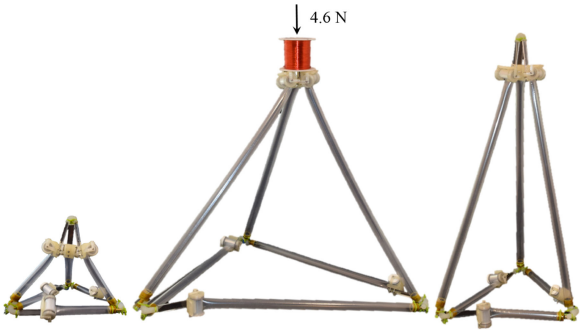

Fig. 2. Tetrahedral robot with pneumatic reel actuators described in [17]. Future work will implement our algorithms on a robot with similar construction.

linear actuators arranged in repeated graphical motifs of tetrahedrons or octahedrons to facilitate kinematic computations [1]–[3]. Robots of this type have been referred to as variable geometry trusses [4], and have been proposed as manipulators [5], as platforms that allows locomotion over various terrains, and as robots with the ability to change shape to adapt to tasks that may be unknown *a priori* [6]. Other physical variants of shape changing robots based on linear actuators include the modular linear actuator system presented in [7], an octahedron designed for burrowing tasks made of high-extension actuators presented in [8], and an active-surface type device that uses prismatic joints to deform a surface into arbitrary shapes while respecting some constraints [9]. In [10], a user interface is presented that allows a novice user to create a large-scale truss structure, and then animate its motion by inserting a few linear actuators. In [11], a 2-D structure is built from a collection of triangles with linear actuators as their edges, allowing the overall structure to change shape. Some recent work has focused on an LAR where the edges can also actively reconfigure their connectivity as well as their lengths, which have been called variable topology trusses, with particular focus on the application of shoring up rubble in disaster sites [12], [13]. Other work has also focused specifically on the mechanical design of linear actuator robotic structures [14], [15]. Recent work has produced compact linear actuators that can extend up to ten times their nominal length [16], [17], making large-scale LARs with significant shape changing capabilities technologically feasible. In future work, we plan to implement the proposed algorithms on a system composed of pneumatic reel actuators developed by Hammond *et al.* [17], as shown in Fig. 2.

Tensegrity robots are conceptually similar to LARs and have been proposed for similar applications. Tensegrity robots consist of a number of rigid bars under compressive loading suspended in a network of compliant cables in tension. Tensegrity robots move by changing the lengths of a subset of the members, typically by spooling in or out the compliant cables, and have been proposed for a number of applications, with a particular focus on use as a planetary rover [18], [19]. Tensegrity robots have the additional constraint that some members (the cables) can only experience tension loads, imposing some limits on their ability to change their overall shape.

B. Control and Planning Approaches

A variety of different control strategies have been used for past implementations of LARs and tensegrity robots. These methods differ in their use of a model, treatment of constraints, and whether or not they consider dynamic effects or assume that the motion of the robot leads to only quasi-static motions. The key challenge is that the large number of independent actuators create a high-dimensional input space that can be challenging to explore. Existing work on controlling TETROBOTs focuses on algorithms for propagating kinematic chains of tetrahedrons or octahedrons [1]. When the motion of certain nodes are specified, Lee and Sanderson [2] and [3] provide centralized and decentralized algorithms for finding dynamically consistent motions for systems with the requisite chain architecture. In [20] and [21], hand-designed gaits are presented for use on a particular tetrahedral robot. These gaits are presented as quasi-static paths (trajectories of how the edge lengths change with time, with no accounting for the requisite forces), and they discuss how the target edge lengths would be used as inputs to a PID controller that would realize these quasi-static gaits. Wang *et al.* [22] present a dynamic model for a tetrahedral robot, but states that the dynamics are too complicated for a model-based controller, so they use the kinematic gait presented in [21], but use the dynamics model to better track the trajectory. Zagal *et al.* [8] also specify motion of an octahedral robot in terms of kinematic motion of the nodes, and in simulation tests, a large family of possible deformations. In [23], a rapidly-exploring random tree algorithm (RRT) is used to plan for a kinematic model of the robot by planning directly in the space of node positions. This method is generally applicable to arbitrary graph structures, but the large potential space of motions makes it computationally challenging to find a solution, and heuristics must be used to bias the sampling during the RRT algorithm.

Control of tensegrity robots has also received substantial work. Some approaches use geometric form finding in conjunction with Monte Carlo simulations to determine useful shapes [24], [25]. If desired trajectories for the node positions are known, the force density method provides the control inputs to move the equilibrium state of the robot along a certain path, neglecting dynamics effects [26], [27]. Under a quasi-static assumption, sampling-based planning has been used to find a feasible, but not optimal, path that avoids collisions [28]. Due to the large state space and input space for the dynamic system, randomized kinodynamic planning presents a potential approach [29]. In [30], a sampling-based planning technique is used in conjunction with a dynamic tensegrity simulator, which requires parallelization to be able to operate in a reasonable timescale. In [31], full kinodynamic planning is used, but in order to make the problem tractable, Littlefield *et al.* first introduce an approximate quasi-static solution and use that as a starting point for the kinodynamic sampling. A common approach in tensegrity robotics is to consider the dynamics but to do so with a learning or adaptive based approach, including evolutionary style algorithms [25], [32], [33] or reinforcement learning [34], [35]. The gaits and motions resulting from these approaches leverage the dynamics, but tend to not fully utilize available

information on the known models of these systems, and it is challenging to adapt these methods to a specific, prescribed task (moving the center of mass along a trajectory). Other approaches reduce the problem to a small number of parameters for periodic trajectories [36], [37] using central pattern generators and Bayesian optimization. In [38], a dynamic model is used for offline model predictive control simulations that provide a training set for a deep learning system. In [39], guided policy search and a reduction of the search space due to the symmetry of the SUPERball tensegrity robot is used to enable tensegrity locomotion over nonsmooth surfaces. Also leveraging symmetry, Vespignani *et al.* [40] propose a method to extend a single motion primitive into longer trajectories. A variety of control approaches to tensegrity robots are reviewed in [41].

We note that while many of the tensegrity control approaches do leverage the dynamics of the tensegrity system, they do not directly consider a full model based approach of the dynamics, either treating the dynamics through a data-driven approach or employing a method to reduce computation (leveraging symmetry, or using a kinematic solution as a guide). For this reason, we propose that better quasi-static planning methods are valuable steps toward improved system performance. We also note that whereas the dynamics of tensegrity systems often have a significant effect on the robots response, the linear actuators used in LARs are often relatively slow, leading to less emphasis on leveraging the dynamics. In this work, we follow the precedent of the prior work on LARs and utilize a kinematic model.

In this article, we present the kinematics of LARs with arbitrary graphical structure, including overconstrained structures, and utilize an optimization-based approach for planning directly over the position of nodes of the robot. This optimization approach allows our method to be customized to different tasks and cost functions. We consider actuator constraints (to enforce min–max elongation), physical constraints (to prevent self-intersection and enforce a minimum angle between connected actuators), and constraints to avoid kinematic singularities in arbitrary robots in designing locomotion algorithms. The key contributions of this work are two algorithms to solve the following problem.

Problem 1: Move the center of mass of the robot in a prescribed direction v_{cm}, or along a prescribed trajectory $x_{\text{cm}}(\tau)$, ensuring that the robot is always physically feasible and that the robot does not pass through any singular configurations.

The first algorithm we propose solves an online optimization to minimize an objective function that considers only the current state and motion of the robot while ensuring physical feasibility. This method applies to any robot that is in an infinitesimally rigid configuration. However, this method does not guarantee the persistent feasibility of the robot's motion, meaning it is possible that the robot will reach a configuration from which it cannot continue without violating physical constraints (i.e., it might get tangled up). To ensure persistent feasibility, we present another method where we solve an offline optimization that generates periodic motion primitives to move a robot from a starting configuration to an equivalent configuration centered on a new support polygon. This motion primitive is then used by a high-level planner to plan paths from an initial configuration to a goal. We refer to this method as the two-tiered planning approach. This method guarantees persistent feasibility of the trajectory, but requires that the initial configuration of the robot satisfy certain symmetry requirements. The performance of the two algorithms is compared in simulation study in which we find that the two-tiered planning approach gives better performance in terms of cost.

This article builds upon our past work by Usevitch *et al.* [42] on modeling and providing an algorithmic foundation for this class of robots. This work adds singularity constraints and angle constraints to our past work, and the solution method for the online optimization method has been improved. The two-tiered planning algorithm using motion primitives is also new in this work. We also note that the work presented in the conference paper has served as the foundation of the work in [23] and [43].

The rest of this article is organized as follows. Section II formalizes a model for LARs and derives the forward and inverse kinematics relating the change in actuator lengths to node positions. Section III describes the physical constraints imposed to ensure the robot motion is feasible. Our single-step locomotion algorithm is given in Section IV, and our two-tiered approach is presented in Section V. These methods are compared in Section VI. In Section VII, we discuss the performance of the kinematic plan in the presence of dynamic effects. Finally, Section VIII concludes this article.

II. Kinematics

Formally, we model an LAR as a framework, which consists of a graph G and vertex positions $p_i \in \mathbb{R}^d$. Our methods are applicable to LARs embedded in Euclidean space of arbitrary dimension d, but we focus on the embeddings in 3-D ($d = 3$). The graph is denoted as $G = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \{1, \ldots, N\}$ are the vertices of the graph, and $\mathcal{E} = \{\ldots, \{i, j\}, \ldots\}$ are the undirected edges of the graph. The geometry of the robot is fully represented by the concatenation of all vertex positions $x = [p_1^T, p_2^T, \ldots, p_n^T]^T$. In this article,, we consider a quasi-static model as opposed to a dynamic model, implicitly assuming that the robot's motion is slow enough that inertial effects are negligible, an assumption that we will further address in Section VII. We define a length vector L, which is a concatenated vector of the lengths of all edges in the graph

$$L_k = \|p_i - p_j\| \quad \forall \{i, j\} \in \mathcal{E}. \tag{1}$$

The vector L is of length n_L equal to the number of edges of the graph, and can be directly computed from the framework (G, x). We use the notations $L(x)$ to indicate the length vector induced by a set of node positions x. We note that the relationship in (1) is the constraint on the node positions created by an edge. Note that $L(x)$ represents the "inverse kinematics" for LAR robots since it is a function that maps from the vertex positions (analogous to the end effector position in a serial manipulator) to the lengths of the linear actuators (analogous to the joint positions in a serial manipulator), and it is trivial to obtain (as also noted by Hamlin and Sanderson [1]).

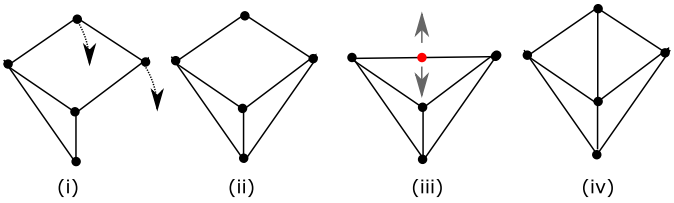

Fig. 3. (a) Nonrigid framework. Arrows show the direction nodes can be moved with no change to lengths. (b) Minimally infinitesimally rigid network. (c) Network has the same topology as (b) and is rigid but not infinitesimally rigid and, hence, not controllable. No controlled motion is possible in the direction of the arrows. (d) Additional edge is added to (b), meaning the structure is no longer minimally rigid. Motions of the actuators must be coordinated and cannot always be made independently.

A. Rigidity

While it is trivial to obtain the edge lengths from the node positions, our task is to invert this relationship and control the node positions by changing the edge lengths. In a network of linear actuators, each link length imposes one constraint on the node positions, as given in (1). Finding the vertex positions from the link lengths means finding node positions that satisfy all of the constraint equations up to translation and rotation of the entire network. Several classes of solutions exist based on the rigidity of the underlying graph. Examples of a few of these classes are shown in Fig. 3, and our analysis of the device kinematics in the following section will depend on the rigidity of the underlying graph of the robot. If the system of equations has infinite solutions, the framework is not rigid, as it is possible to move the system relative to itself without violating length constraints as in Fig. 3(a). A framework is rigid if there are a discrete number of solutions to the constraint equations, and all deflections of the system relative to itself violate the length constraints.

Of particular use to our analysis are graphs that are infinitesimally rigid, meaning that all infinitesimal deflections of the system relative to itself violate the length constraints. Infinitesimal rigidity is dependent on the configuration x and is not an inherent characteristic of the graph G. Infinitesimally rigid frameworks are a subset of rigid frameworks, meaning a framework can be rigid but not infinitesimally rigid, but all infinitesimally rigid frameworks are also rigid. Fig. 3(b) shows an infinitesimally rigid framework, whereas the one in Fig. 3(c) is rigid but not infinitesimally rigid.

Of particular interest in the design of LARs are minimally rigid graphs. A minimally rigid graph is a rigid graph where the removal of any link causes the graph to lose rigidity. These minimally rigid graphs provide a lower bound on the number of links necessary to constrain a certain number of nodes. For a graph in three dimensions, at least $3n-6$ edges are necessary for minimal rigidity, which can be understood intuitively based on a degree of freedom argument. Each node in $\mathbb{R}^3$ has 3 DOF, and each edge imposes a constraint that removes at most one degree of freedom. The final structure has 6 DOF in its rigid body motion (three translational and three rotational). An infinitesimally rigid graph in $\mathbb{R}^3$ with $3n-6$ edges is minimally rigid, although $3n-6$ edges does not necessarily imply rigidity. In Fig. 3, the framework in (b) is infinitesimally minimally rigid, framework (c) is minimally rigid but not infinitesimally rigid, and framework (d) is infinitesimally rigid and overconstrained.

B. Differential Kinematics

As opposed to reconstructing node positions from edge lengths, we instead determine the kinematic relationship of how nodes move from a given start point with changing link lengths. To find this relationship between $\dot{L}$ and $\dot{x}$, we first square (1) and take its derivative with respect to time to obtain

$$\frac{dL_k^2}{dt} = 2L_k\dot{L}_k = 2(p_i - p_j)^T\dot{p}_i + 2(p_j - p_i)^T\dot{p}_j. \quad (2)$$

Rewriting in matrix form

$$\begin{bmatrix} \dot{L}_1 \\ \dot{L}_2 \\ \cdots \\ \dot{L}_{n_L} \end{bmatrix} = R(x)\dot{x}. \quad (3)$$

In this equation, $R(x)$ is a scaled version of the well-known rigidity matrix in the study of rigidity [44], [45], or the kinematic matrix in the study of kinematically indeterminate frameworks [46]. Each row of $R(x)$ represents a link L_k. For example, let row k represent the link between nodes i and j. The only nonzero values of row k are $R(x)_{k,(3i-2,3i-1,3i)} = \frac{(p_i-p_j)}{\|L_k\|}$ and $R(x)_{k,(3j-2,3j-1,3j)} = \frac{(p_j-p_i)}{\|L_k\|}$. Note that in the standard rigidity matrix, the entries are of the form $(p_j - p_i)$ and not $\frac{(p_j-p_i)}{\|L_k\|}$, hence we refer to $R(x)$ as the scaled rigidity matrix. For a graph with n vertices, N_L edges, and the positions of the vertices given in $\mathbb{R}^d$, then $R \in \mathbb{R}^{N_L,nd}$. For $d=3$, the maximum rank of R is $3n-6$. If the matrix $R(x)$ is of maximum rank, the framework is infinitesimally rigid [45].

C. Contact With the Ground

In addition to the relationship between actuator lengths and vertex positions, we must capture the robot's interaction with the environment. Sufficient constraints between the robot and the environment must be used to ensure that the location of the structure is fully defined (six independent relationships when the structure is in $\mathbb{R}^3$). For this work, we assume that three of the robot's nodes on the ground form a support polygon, and that the nodes that make up the support polygon do not slide across the ground. If after applying some control the center of mass leaves the support polygon, the structure rolls about the edge of the support polygon closest to the center of mass until the next point comes into contact with the ground. This process is repeated until the center of mass is inside the support polygon. This assumption is valid for many cases, but it does neglect the dynamic nature of the rolling transition. The decision that the support feet do not move along the ground is restrictive, but means that the gaits are somewhat robust to changes in the ground properties, and do not depend on friction models of the ground.

We encode these relationships in terms of the equation $C\dot{x} = 0$ where each row of C has one nonzero entry that is equal to 1, such that each row of C makes one node of the robot stationary in one coordinate of the environment. For a minimal set of constraints, $C \in \mathbb{R}^{6,\,3n}$. We choose this minimum set of constraints such that one of the support feet is fixed in all three dimensions, another support foot is fixed in the vertical direction and one of the lateral directions, and the last support foot is fixed only in the vertical direction. If we constrain the three nodes of the support polygon to be unable to move, $C \in \mathbb{R}^{9,\,3n}$. In the future, we could expand the contact constraints to the form $C\dot{x} = \dot{F}$ where $\dot{F}$ represents some motion of the environment and the C matrix potentially captures a different type of interaction with the environment. For example, this framework could enforce an interaction where the robot's feet could slide on the ground or be supported against moving obstacles.

D. Kinematic Model

We combine the kinematic model with the contact model to obtain the following differential kinematics:

$$\begin{bmatrix} \dot{L} \\ 0 \end{bmatrix} = \begin{bmatrix} R(x) \\ C \end{bmatrix} \begin{bmatrix} \dot{x} \end{bmatrix} = H(x)\dot{x}. \tag{4}$$

If the system is infinitesimally minimally rigid and a minimal set of constraints is applied that is linearly independent of the link constraints, the combined matrix $H = [R^T\ C^T]^T$ is full rank and square, and hence invertible, allowing us to write

$$\dot{x} = H(x)^{-1} \begin{bmatrix} \dot{L} \\ 0 \end{bmatrix}. \tag{5}$$

Note that this is the form of a driftless dynamical system, and that $H(x)^{-1}$ is the Jacobian matrix relating the motion of the actuators to the motion of the nodes. The vector $\dot{L}$ describes the rates of change of the linear actuators, and hence is the input to the system. Equation (5) shows that when the $H(x)$ is invertible, each input channel $\dot{L}_k$ can be commanded independently of the others. The fact that this matrix is invertible means that the input space is all possible length velocities, allowing us to make the following proposition.

Proposition 1: Given an infinitesimally minimally rigid framework with the minimum number of constraints to the environment, the length of each edge can change independently.

This means that it is not necessary to coordinate movements between lengths as long as the graph remains minimally infinitesimally rigid.

E. Controlling Overconstrained Networks

If the system is infinitesimally rigid but not minimally rigid, it is overconstrained and some motions of the linear actuators must be coordinated. In this case, the H matrix is skinny, with more rows than columns. Taking the singular value decomposition of the combined H matrix

$$U^T \begin{bmatrix} \dot{L} \\ 0 \end{bmatrix} = \Sigma V^T \dot{x} \tag{6}$$

$$\begin{bmatrix} U_1^T \\ U_2^T \end{bmatrix} \begin{bmatrix} \dot{L} \\ 0 \end{bmatrix} = \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} V^T \dot{x}. \tag{7}$$

The bottom rows of this expression can be expressed as a constraint, which encoding how certain lengths must move in a coordinated fashion

$$U_2^T \begin{bmatrix} \dot{L} \\ 0 \end{bmatrix} = 0. \tag{8}$$

By utilizing this constraint, redundant rows of the H matrix and their corresponding elements in the vector $[\dot{L}^T\ 0]^T$ can be removed until it is square and full rank and, hence, invertible. We call the reduced H matrix and L vector the master group, and we denote them as H_m and L_m, respectively. We refer to the removed rows as the slave group, and denote as H_s, and the removed actuator inputs as $\dot{L}_s$. We note that the actuators chosen for the master and slave groups are partially up to the user's discretion, and could potentially change based on configuration. As an example procedure, an algorithm could initialize $H_m = H$, and H_s as an empty matrix, and then iterate through each row of the H_m matrix. If it finds a row linearly dependent on the previous rows from the H_m matrix and places it in H_s, and removes the corresponding element of L_m and places it in L_s. This allows us to express the system as follows:

$$\dot{x} = \begin{bmatrix} H_m(x) \end{bmatrix}^{-1} \begin{bmatrix} \dot{L}_m \\ 0 \end{bmatrix} \tag{9}$$

$$\text{s.t.} \quad \dot{L}_s = H_s(x) H_m(x)^{-1} \begin{bmatrix} \dot{L}_m \\ 0 \end{bmatrix}. \tag{10}$$

Due to (10), the input space is restricted such that only combinations of link velocities that satisfy the constraint can be physically realized. The master inputs $\dot{L}_m$ can be picked arbitrarily, but $\dot{L}_s$ must be chosen to satisfy the constraint equation.

This system can be expressed in the standard form of a linear dynamical system, $\dot{x} = Ax + Bu$ where $A = 0, u = [\dot{L}^T\ 0^T]^T$, and $B = H_m(x)^{-1}$. We now make the following proposition.

Proposition 2: A framework that is infinitesimally rigid is fully actuated.

This means that for an infinitesimally rigid system, control of every degree of freedom can be achieved given control of the rate of change of the actuator lengths and the motion of the contact points. This has the key advantage of allowing us to plan our motion in terms of node positions, and then use the $[R^T\ C^T]^T$ matrix to determine what input to apply to the actuators.

III. Physical Constraints

Locomotion requires finding a method to actuate the robot to move while it maintains physical feasibility. We define feasibility as follows.

Definition 1: A framework (G, x) is feasible if it meets following three types of physical constraints.

1) The lengths of all actuators fall within a fixed maximum and minimum length range.

2) The actuators do not physically intersect (except at the endpoints of two connected actuators).
3) The angles defined by two actuators connected at a joint remain above a minimum value.

To ensure that all motions of the robot are physically feasible, we detail the form of the constraints and quantify how many of each type of constraint occurs in the optimization based on the characteristics of the underlying graph. In addition to these physical constraints, we also present constraints to prevent the robot from crossing configurations where infinitesimally rigidity is lost, which correspond to the singular configurations of the robot.

A. Length Constraints

For physical feasibility to be preserved, all actuators must be maintained between a maximum and minimum actuator length. The squared length of actuator k that connects nodes $\{i, j\}$ is quadratic in x, and the constraint that it remain within the set maximum and minimum length can be expressed as

$$L_{\min}^2 \leq x^T \left[I_d \otimes A_k \right] x \leq L_{\max}^2 \tag{11}$$

where A_k is a matrix where only nonzero entries are $A_{k,ii} = A_{k,jj} = 1$ and $A_{k,ij} = A_{k,ji} = -1$. We note that constraints of the quadratic form $x^T Q x \leq c$, where c is a positive constant, are convex if and only if Q is positive semidefinite. We note that A_k is the Laplacian matrix of a graph that contains only edge k. As the Laplacian matrix is always positive semidefinite, the maximum length constraint is convex in the node positions while the minimum length constraint is not. Thus, our algorithms will handle nonconvex and nonlinear constraints.

B. Distance Between Actuator Constraints

We also enforce the constraint that actuators do not collide physically, except for at the vertices where they are joined. To determine if two actuators cross, the minimum distance between them must be greater than $d_{\min}$, a positive diameter of the actuator assuming that the actuator can be represented as a cylinder. The minimum distance between actuators connecting vertices i, j and k, l is denoted as d_{ij}^{kl}, and can be expressed as follows:

$$d_{ij}^{kl} = \min \| (p_i + \alpha(p_j - p_i)) - (p_k + \gamma(p_l - p_k)) \| \times \alpha, \gamma \in (0, 1). \tag{12}$$

These links are not in collision if $d_{ij}^{kl} > d_{\min}$. Efficient algorithms for this computation have been explored previously [47]. Checking the pairwise distances between all edges in a graph requires checking $\frac{N_L^2 - N_L}{2}$ constraints of the type expressed in (12). As we do not compute the distance between actuators that are connected at a node, the number of constraints is reduced by the number of pairwise distances between edges that meet at a node, which for node i is given by $\frac{g_i^2 - g_i}{2}$ where g_i is the degree of the node. Thus, the total number of constraints to avoid collisions between actuators is

$$\frac{N_L^2 - N_L}{2} - \sum_{i=1}^{n} \frac{g_i^2 - g_i}{2}. \tag{13}$$

C. Angle Constraints

Another key physical constraint is that the angle between connected actuators remain above a certain value, which is especially important when the actuators have a high elongation ratio. An angle constraint between two edges is a function of three vertices. We define p_i as the position of the shared node between two edges, and p_j and p_k as the other vertices of the two edges. The angle constraint is

$$\cos(\theta_{\min}) \leq \frac{(p_j - p_i)^T (p_k - p_i)}{\|p_j - p_i\| \|p_k - p_i\|}. \tag{14}$$

The number of angle constraints can also be expressed in terms of the degree of the nodes of the graph

$$-N_L + \frac{1}{2} \sum_{i=1}^{n} g_i^2. \tag{15}$$

D. Rigidity Maintenance Constraint

Our proposed optimization approach is based on the observation that if the robot is infinitesimally rigid, we can directly optimize a path for the node positions and recreate the needed actuator trajectories. For this assumption to remain valid, the robot must maintain its infinitesimal rigidity, meaning the rigidity matrix R must remain of rank $3n - 6$. Designing controllers that maintain infinitesimal rigidity has been a topic in formation control of multiagent systems [48], [49]. In [48], the rigidity eigenvalue for frameworks in $\mathbb{R}^3$ is defined as the seventh smallest eigenvalue of $R(x)^T R(x)$ and the gradient of the rigidity eigenvalue with respect to the node positions is used as part of a controller. In the general case, infinitesimal rigidity can be enforced using the following constraint:

$$\lambda_7 > \lambda_{\text{crit}} \tag{16}$$

where λ_7 is the seventh smallest eigenvalue of the $R(x)^T R(x)$ matrix and λ_{crit} is its minimum allowable value.

One problem with (16) is that the magnitude of λ_7 changes quadratically with network size. To provide a constraint that is invariant to network scale, we instead use the worst-case rigidity metric, taken directly from [50], and defined as

$$\frac{\lambda_7}{\sum_{i=1}^{3n} \lambda_i} = \frac{\lambda_7}{\text{tr}(R(x)^T R(x))} = \frac{\lambda_7}{\sum_{i=1}^{N_L} (L(x)_i)^2} \geq \lambda_{\text{crit}}. \tag{17}$$

It has been noted that if a framework is infinitesimally rigid in one configuration it is infinitesimally rigid almost everywhere, meaning that for a graph with one infinitesimally rigid configuration, the set of nonrigid configurations is a set of zero measure [51]. We make the observation that the configurations where the robot loses rigidity often divide the state space into disconnected regions. We define each of these regions as a rigidity equivalence class as follows.

Definition 2 (Rigidity Equivalence Class): The framework $F_1 = (G, X_1)$ and the framework $F_2 = (G, X_2)$ are in the same

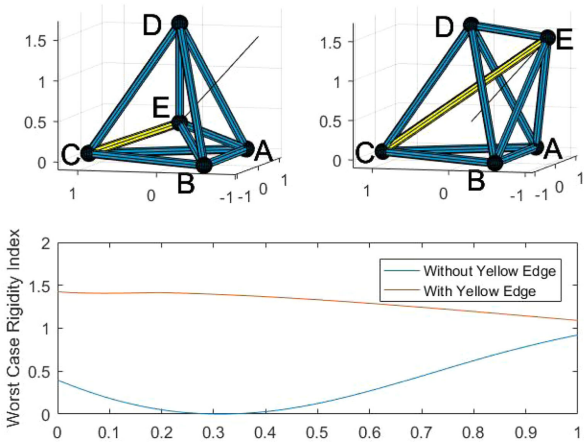

Fig. 4. Values of the worst-case rigidity index (17) as the position of node E changes linearly between the left and right configurations. Without edge CE (shown in yellow), the worst-case rigidity index goes to 0 when E is coplanar with DAB, whereas with the yellow edge, the worst-case rigidity index remains greater than 1.

rigidity equivalence class if a continuous path $x(t)$ exists such that $x(0) = X_1, x(T) = X_2$, and the rigidity matrix $R(G, x(t))$ is maximal rank for all $t \in (0, T)$.

Analytically characterizing these rigidity equivalence classes for an arbitrary graph has proved challenging. However, we are able to make a statement for the case of graphs that contains three-simplex (a complete tetrahedron) as a subgraph. For each complete tetrahedron, we define its orientation as the sign of the signed volume, which is computed as

$$V = (p_4 - p_1)^T (p_3 - p_1) \times (p_2 - p_1). \tag{18}$$

These preliminaries allow us to make the following statement.

Theorem 1: Let $F_1 = (G, X_1)$ and $F_2 = (G, X_2)$ be two minimally rigid frameworks in $\mathbb{R}^3$. If there exists a subgraph of G that is a three-simplex and F_1 and F_2 contain the simplex with opposite orientation, the two frameworks lie in different equivalence classes.

Proof: Finding a smooth path $x(t)$ for the vertices of a simplex from one orientation to the other requires the signed volume to smoothly change signs, passing a configuration where $V = 0$. When $V = 0$ for a simplex, one of the edges of the simplex is a linear combination of the others, meaning there is a redundant edge in the R matrix. For a minimally rigid graph, the R matrix has $3n - 6$ rows, so any linearly dependent edges indicate the matrix is not maximal rank and, hence, not infinitesimally rigid. ■

One general question in the design of LARs is if an overconstrained network is necessary, or if a minimally rigid network is sufficient. We give an example where an overconstrained robot can achieve motion through a configuration that would represent a singularity were the robot minimally rigid (see Fig. 4). In this example, we first consider the robot to be only composed of the blue edges (edge CE is not present). In this case, both the left and right configurations are infinitesimally rigid and simplex ABED has different orientation in each configuration, meaning that the two configurations lie in different rigidity equivalence classes by Theorem 1. If the node positions are linearly interpolated between the two configurations, the rigidity index in (17) goes to 0 when node E is coplanar with nodes ABD. The addition of the yellow edge, which makes the robot overconstrained, allows rigidity to be maintained throughout the transition, as shown by the plot in Fig. 4. We note that with the yellow edge, this graph is the fully connected five-node graph, known as the K5 graph. The K5 graph displays another interesting property.

Theorem 2: The rigidity matrix $R(x)$ for a robot represented by a complete graph of five or more nodes only loses rank at configurations where the robot has actuators in collision.

Proof: For a node in a complete graph to have an unconstrained infinitesimal motion, its neighboring edges must not span $\mathbb{R}^3$, meaning that all nodes must lie in the plane. Complete graphs with five or more nodes do not have planar noncrossing embeddings. ■

This result means enforcing the constraint that no actuators collide for the K5 graph naturally enforces the graph rigidity constraint. Whenever we evaluate a K5 robot in this article, we leverage this result and do not enforce the rigidity maintenance constraint.

E. Constraint Satisfaction Between Timesteps

Our approach to finding a trajectory for an LAR is to use an optimization to solve for a discretized trajectory containing N_{config} configurations denoted as x^j where $j = 1, 2, \ldots N_{\text{config}}$. The optimization solution guarantees that the configurations x^j satisfy the constraints defined earlier, which we now succinctly express as $f(x^j) \leq 0$. However, the nonconvex nature of the constraints means that it is possible that the intermediate configurations (the configurations between x^j and x^{j+1}) may violate the constraints. To address this, we enforce a constraint that two sequential configurations must be close together in terms of the distance each node travels. We define this constraint as

$$\|p_i^j - p_i^{j-1}\| \leq d_{\text{move}} \quad \forall i. \tag{19}$$

We assume that the intermediate configurations between x_i^j and x_i^{j-1} are given by linear interpolation. From work on sampling-based motion planning [52], the maximum violation of a constraint between two configurations can be bounded by using the Lipschitz constant, K of the constraint as follows:

$$\left|f(x^j) - f(x^k)\right| \leq K\|x^j - x^k\|. \tag{20}$$

Given the Lipschitz constant for each constraint function, it is possible to augment the constraints with a buffer such that satisfying the buffered constraints and the constraint in (19) ensures satisfaction of the true constraint. In our case, we assume that the constraints already include this buffer. In practice, we choose d_{move} to ensure that two edges cannot jump over each other without violating the collision constraint by picking $2d_{\text{move}} \leq d_{\min}$.

IV. Single-Step Locomotion

Our first approach to solving the locomotion problem involves solving an online optimization to move the center of mass to a desired position for one time step. It acts greedily to minimize

an objective function for a time step, and does not account for making and breaking contact with the ground. Our second method, presented in Section V and referred to as a two-tiered planning approach, extends this single-step computation to an optimization over multiple steps. The two-tiered approach directly accounts for the rolling behavior in the computation, but imposes restrictions that the robot must satisfy certain symmetry requirements.

A. Controlling the Velocity of the Center of Mass

The position of the center of mass is defined in terms of the mass matrix of the system, $M \in \mathbb{R}^{3\times 3n}$. Without loss of generality, the quasi-static model allows us to assume that all mass is concentrated at the nodes of the system. In our case, we assume that all actuators are of uniform, evenly distributed mass, and thus half of the mass is assigned to each end of the actuator. The position of the center of mass is given by

$$x_{\text{com}} = Mx = \left[m_{\text{vec}} \otimes I_3,\right] x \tag{21}$$

where $m_{\text{vec},i}$ is the sum of all of the partial masses assigned to node i. In the uniformly distributed case, $m_{\text{vec},i} = \frac{d_i}{2N_L}$. With this mass matrix, we can express the velocity of the center of mass as a function of the actuator velocities

$$\dot{x}_{\text{com}} = M\dot{x} = MH^{-1}\dot{L}. \tag{22}$$

We can now pick any $\dot{L}$ that achieves a desired motion of the center of mass. The maximum rank of M is d, so for a system with many vertices, MH^{-1} will have more columns than rows, and there is freedom in which $\dot{x}$ is selected to move the center of mass. We define an optimization problem to pick a value of $\dot{L}$ that minimizes an objective function.

B. Optimization Setup

The kinematic relationships derived in Section II apply to a continuous time system. To optimize the trajectory, we work in discrete time, denoting the configuration of the robot with the superscript x^j. In practice, we determine the node velocities by linearly interpolating from the current configuration to the configuration that is the result of the optimization, and determine the necessary actuator velocities using the kinematic relationships. The optimization procedure takes as an input the configuration at x^{j-1} and optimizes the next desired configuration x^j. We seek to find a trajectory that maximizes some cost function $J(x)$ while satisfying constraints. As this optimization minimizes the cost function over only one step, we refer to this optimization approach as the greedy method. The complete optimization problem is given as follows:

$$\min_{x^j} J(x^j) \tag{23}$$

subject to

$$Cx^j = b \tag{24}$$

$$Gx^j \geq 0 \tag{25}$$

$$f(x^j) \leq 0 \tag{26}$$

$$\|x_i^j - x_i^{j-1}\| \leq d_{\text{move}} \quad \forall i. \tag{27}$$

The choice of cost function $J(x)$ will be discussed in the following section. Equation (24) fixes the contact points and is the discrete time version of the ground constraint, where b is a vector of locations of the vertices in the support polygon. In the locomotion optimization, we also enforce the linear constraint that no nodes pass through the ground, $Gx > 0$, where $G = I_n \otimes \text{diag}([0\ 0\ 1])$. We denote all of the feasibility constraints, including maximum and minimum actuator length (11), actuator collision constraints (12), angle constraints (14), and singularity avoidance constraints (17) as $f(x^j) \leq 0$ as given in (26).

C. Objective Function

By defining this problem as an optimization problem, the system will take the action that instantaneously optimizes some objective, $J(x)$. One intuitive choice for the cost function is $J(x) = \|\dot{L}(x)\|^2 = \|R(x)\dot{x}\|^2$, which penalizes large actuator velocities. In discrete time, we approximate this cost as

$$J(x) = \|L(x^j) - L(x^{j-1})\|^2. \tag{28}$$

As mentioned earlier, one potential issue with a single-step method is that persistent feasibility is not guaranteed. One heuristic to prevent the robot from getting tangled up in an unfavorable configuration is to try and keep the network as close as possible to a fixed operating point, such as attempting to keep all actuators close to a nominal length l_N. This can be encoded with an objective function

$$J(x) = \|L(x^j) - l_N\|^2. \tag{29}$$

We will quantitatively compare the results of using both of these cost functions in Section VI.

D. One-Step Optimization Results

We find a feasible solution to the optimization problem using the sequential quadratic programming algorithm available in the MATLAB fmincon toolbox. The most computationally expensive part of this algorithm is repeatedly checking to see if the nonlinear constraints are violated, a process that could be parallelized in a future implementation. To speed computation, $\frac{\partial f(x)}{\partial x}$ is computed analytically before operation. We demonstrate the character of the solutions that result from this optimization, we will show the types of trajectories generated when it is applied to different robots in the following sections.

1) Randomly Generated Robot: To show the generality of the algorithm to a wide variety of robots, the locomotion of a randomly generated minimally rigid seven node robot is shown in Fig. 5. The initial configuration of the robot is obtained by starting with a triangular base and iteratively adding one node and connecting it with three randomly selected existing nodes. For each node, the node position is randomly regenerated until all constraints are satisfied. The objective function presented in (28) is used. For these simulations (and for simulations throughout this article), the actuator lengths were constrained to remain between 0.5 and 4 units, the minimum angle between connected actuators was 10°, and the minimum distance between actuators

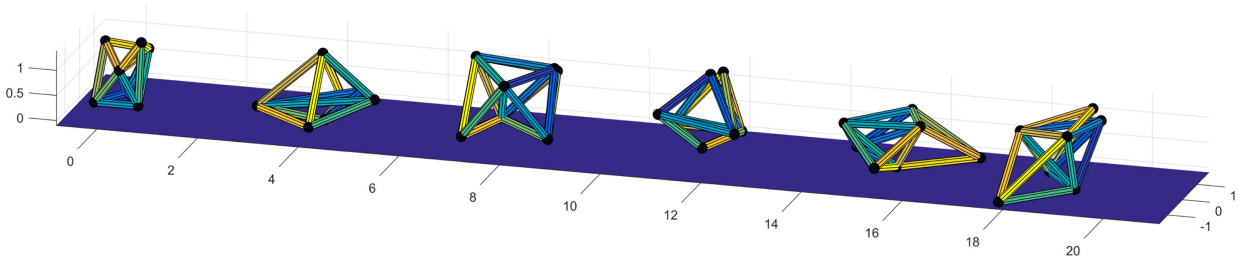

Fig. 5. Movement of an LAR using the objective function given in (28). The robot is minimally rigid with 7 nodes and 15 actuators.

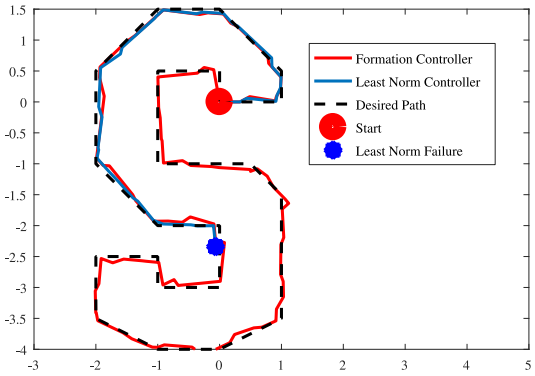

Fig. 6. Path of the center of mass as it travels to each waypoint of an "S." Note that when using the objective in (28), the LAR reaches a point from which it cannot continue. With (29) as the objective, the robot completes the trajectory.

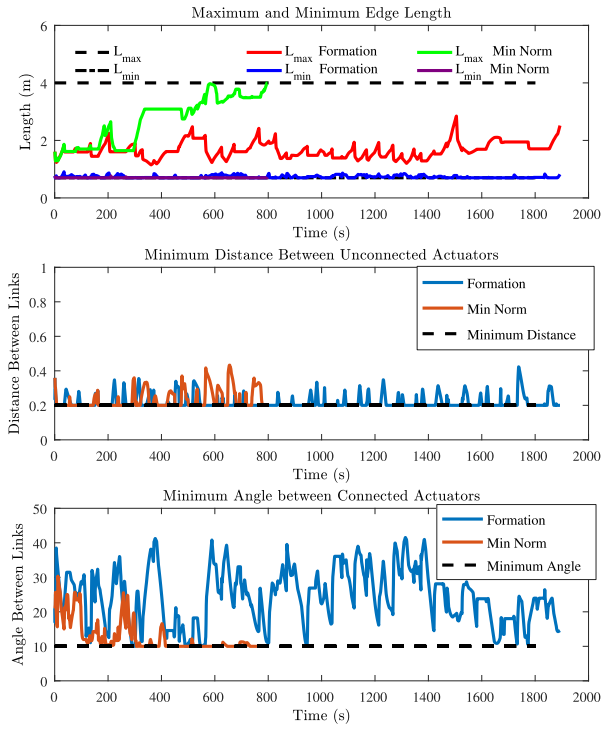

Fig. 7. Plots showing the lengths of the longest and shortest actuators, the minimum distance between any two links that do not share a joint, and the smallest angles throughout the trajectories shown in Fig. 6. While constraints are active, they are never violated.

was set at 0.15 units. The minimum value of the worst-case rigidity index was set at 0.005. The robot has an emergent, almost amoebalike gait as it moves. Videos of this motion are available in the supplementary materials.

2) Trajectory Tracking: In order to demonstrate the ability of the system to follow a trajectory, the K5 robot was controlled to move its center of mass toward waypoints that make up the corners of a predefined trajectory. The resulting trajectories when both (28) and (29) are used as the objective are shown in Fig. 6. We use the same values for the physical constraints as for the random robot, but do not enforce the rigidity maintenance constraint for the K5 robot due to Theorem 2. The variance from the prescribed trajectory occurs because of the rolling motion when the center of mass leaves the support polygon. In order to illustrate the effectiveness of this method in preventing constraints from being violated, Fig. 7 shows that during the trajectories shown in Fig. 6, the various physical constraints on the robot are often active but are not violated.

This trajectory tracking test also gives a sense of the robustness of the control algorithm. A downside to the approach of repeatedly solving the optimization is that persistent feasibility is not guaranteed, meaning it is possible that the network reaches a configuration where it cannot continue without violating some constraint. In the case where the objective function was (28), a configuration was reached where the device could not continue to match the desired center of mass motion without violating constraints (the blue trajectory in Fig. 6). With the objective presented in (29), the planner finds a feasible path that completes the trajectory (the red trajectory in Fig. 6). Simulation results on a variety of trajectories show that a common failure mode of the system is if the robot rolls onto a very large support polygon, it may not have the ability to extend its center of mass and roll again. This failure mode may become less significant if future work included a frictional model of the ground and allowed the support nodes to slide along the ground. Interestingly, relaxing constraints does not necessarily guarantee the robot will be able to travel further before reaching a configuration with no feasible solution. Often, relaxed constraints, such as a higher upper limit on actuator length, lead to failure sooner, as the robot tends to reach more jumbled configurations early in the trajectory. Computation for completing the "S" trajectory involved solving the one-step optimization 1893 times, which took approximately 120 s on a laptop computer (Intel Core i7 Processor, 4 cores, 2.80 GHz, 16 GB RAM). The average time to solve each

optimization was 63 ms, with a standard deviation of 10 ms and a maximum time of 153 ms.

Note that at each time step, these methods instantaneously minimize an objective while a desired motion of the center of mass is obtained. The algorithm can be thought of as greedily trying to move the center of mass. However, motion is not optimal for the entirety of the trajectory. The algorithm does not explicitly take into account making and breaking of contact with the surface, which would be required to discuss the optimality of an entire trajectory. To enable discussion of optimality over several steps as well as to directly consider the rolling behavior of the robot, we extend this method to a tiered planning approach.

V. Two-Tiered Planning Approach

In this section, we extend the one-step optimization of the previous section to an optimization over many configurations of the robot. This multistep optimization directly accounts for the rolling behavior, whereas the previous method moved the center of mass without consideration for the rolling motion. We use an offline optimization to compute trajectories from a predefined configuration centered on one support polygon to the same predefined configuration centered at the next support polygon, but with the node correspondences changed. This precomputed trajectory serves as a motion primitive for a high-level planner that computes a series of support polygons that lead from the robot's initial position to a goal region. When deviations occur from the preplanned trajectory, the only computation that occurs online is using the high-level planner to adjust the path of support polygons. As the final path of the robot is composed entirely of feasible motion primitives, the resulting path is guaranteed to be feasible. In this section, we discuss the necessary symmetry requirements for such a trajectory to exist, present the optimization setup to solve for the motion primitive and our use of a high-level planner to combine the motion primitives. We then discuss methods of smoothing the motion primitives during trajectories over a series of support polygons.

A. Symmetry Requirements

We now detail the symmetry requirements that allow a motion primitive to be optimized offline and then stitched together online into long trajectories. This means that the robot must finish a motion primitive in a configuration identical to the starting configuration, but with different node correspondences. If the robot begins in an arbitrary configuration, a path between the initial configuration and the symmetric configuration must be computed and executed. For the symmetric configuration, we restrict the shape of the support polygon of the robot to be an equilateral triangle, meaning that the robot's motion will be over a grid of equilateral triangles. The symmetry requirements are illustrated in Fig. 8. To enable the same primitive to be reused repeatedly, we constrain the starting configuration to have mirror symmetry about the three lines that originate at the vertices of the support polygon and bisect the opposite edge of the triangle, as shown by the red lines in Fig. 8(a). The final configuration of the robot must be identical to the initial configuration reflected across line 23, but with different nodes occupying different

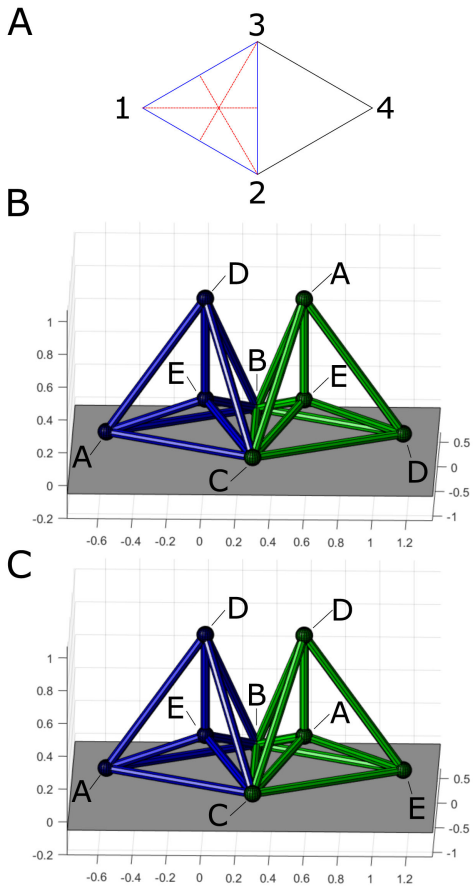

Fig. 8. Illustration of the required symmetry for the starting and ending configurations for the robot. (a) Initial support polygon is shown, which must have mirror symmetry about the three red lines. The ending configuration, which has the support polygon 234, must appear as the mirrored version. (b) and (c) Two configurations of the K5 graph are shown that satisfy the symmetry requirements, but with different node correspondences.

locations in the graph. Note that the nodes at locations 2 and 3 are identical between the two configurations. Fig. 8(b) and (c) shows two examples of starting and ending configurations for the K5 graph that satisfy all symmetry constraints. While these configurations in Fig. 8(b) and (c) look identical, the node correspondences between the two are different. This demonstrates that for some graphs, more than one permutation of the nodes between the starting and ending configuration is possible.

We also note that having a starting and ending configuration that satisfies the symmetry requirements does not guarantee that a valid motion primitive can be found that moves between the configurations without violating constraints. The simplest shape that satisfies the symmetry criteria is the tetrahedron, but a tetrahedron cannot roll from one face to an equal-sized face without having all four of its nodes in a plane—a configuration where actuators are in collision and the graph loses infinitesimal rigidity. In this work, we will analyze the K5 graph with both node correspondences given in Fig. 8, as well as an octahedron robot. We note that for the K5 and octahedron graph, a variety of different configurations of these graphs exist that satisfy the symmetry requirements, for example, the height of the nodes not

on the ground can be uniformly increased to create a different nominal configuration that still satisfies the symmetry requirements. For simplicity, we utilize graphs such that the longest actuators are of unit length.

B. Optimizing a Motion Primitive

Our optimization approach for finding the motion primitives begins with choosing a starting and ending configuration x_0 and x_f that satisfy the symmetry requirements and have support polygons that share a common edge. We discretize the trajectory between the starting and ending configurations into N_{steps} different configurations. We denote each of the configurations j with the superscript x^j, and the variable being optimized is the concatenation of all of these configurations $x_{\text{tot}} = [x^1, x^2, \ldots x^{N_{\text{steps}}}]$. We preassign a tipping configuration x^{j*}, where the center of mass of the robot is exactly on the edge of the support polygon, to be midway through the configuration of steps. The robot tips from one predefined support polygon to the next between the configurations x^{j^*} and x^{j^*+1}. The total optimization to be solved is as follows, with the details of the different constraints discussed in the following sections:

$$\min_{x_{\text{tot}}} \sum_{j=1}^{N_{\text{steps}}} \|L(x^j) - L(x^{j-1})\|^2 \tag{30}$$

subject to

All Configurations

$$C^j x^j = b_j \tag{31}$$

$$g_i^j(Mx^j) + h_i^j \leq 0 \tag{32}$$

$$Gx^j \geq 0$$

$$f(x^j|) \leq 0. \tag{33}$$

Nontipping Configurations

$$\|x_i^{j+1} - x_i^j\| \leq d_{\text{move}} \forall i.$$

Additional Constraints at Tipping Configurations

$$k_1^{j^*}(Mx^{j^*}) + b_1 = 0 \tag{34}$$

$$\|x_s^{j^*} - x_{c1}^{j^*}\| = \|x_s^{j^*+1} - x_{c1}^{j^*+1}\| \tag{35}$$

$$\|x_s^{j^*} - x_{c2}^{j^*}\| = \|x_s^{j^*+1} - x_{c2}^{j^*+1}\| \tag{36}$$

$$\|L(x^{j^*+1})_i - L(x^{j^*})_i\| < c \quad \forall i \tag{37}$$

Ending Configuration

$$x^{N_{\text{steps}}} = x_f. \tag{38}$$

The objective function (30) is a multistep extension of (28), where $L(x^j)$ is the vector of all of the edge lengths of the graph with node locations given by x^j. This objective penalizes sudden and large changes in the lengths of the actuators, and hence favors trajectories that require small changes in actuator lengths. The linear equality constraint in (31) constrains the location of the contact points for each configuration. Note that $C^1 = C^k \quad \forall k \leq j^*$, and $C^{N_{\text{steps}}} = C^k \quad \forall\, k > j^*$, as only two support polygons are used throughout the optimization. In (32), three linear inequality constraints keep x_{com} within the support polygon at each configuration to prevent premature rolling. The variable g_i^j and h_i^j describe the parameters of the line along edge i of the support triangle. For each configuration, we denote $f(x^j) \leq 0$ to represent all of the physical feasibility constraints for one configuration. We write the constraint $x^{N_{\text{steps}}} = x_f$ to ensure the proper final configuration.

We note that for the transition between the tipping configuration and the next configuration (x^{j^*} and x^{j^*+1}), large motions of the node positions are possible due to the rolling, even though change in the edge lengths may be small. For the rolling step alone, we constrain the change in edge lengths to be below a fixed threshold as shown in (37) to prevent the robot from jumping over a physical constraint, such as an actuator collision constraint, between configurations.

1) Tipping Constraints: In addition to the constraints that ensure that each configuration is feasible, we also impose additional constraints that ensure that the robot tips at the predefined tipping configuration. The linear equality constraint in (34) ensures that at the tipping configuration, the center of mass lie on the tipping edge of the support polygon. We denote the new node in the support polygon after the tip as node s. The two quadratic equality constraints in (35) and (36) ensure that the positions of node s, denoted $x_s^{j^*}$ is the proper distance away from each node on the rolling edge, denoted $x_{c1}^{j^*}$ and $x_{c2}^{j^*}$. Note that these constraints do not fix the height from which the robot tips onto the next support polygon. Were the height and position of the next point specified exactly, the two quadratic constraints would be replaced by three linear constraints, but the optimization would lose the ability to change the tipping height.

2) Optimization Results: The motion primitives produced by solving the optimization problem are shown in Fig. 9. For these results, we use $N_{\text{steps}} = 40$, with the tipping configuration $j^* = 20$. We initialize the optimization such that every configuration or after the tipping configuration is exactly the starting or ending configuration, respectively. We initialize the tipping configuration with all nodes of the initial and final support polygon in place, and the nodes that are not part of the support polygon positioned such that the center of mass is on the tipping edge. The optimization was solved using the fmincon solver available with MATLAB.

The top two rows of Fig. 9 correspond to the two different node correspondences for the K5 graph discussed previously. The first row is the resulting motion primitives when using the correspondence in Fig. 8(b). Here, the center node of the robot before rolling remains the center node after rolling. The second row is the resulting motion when the correspondence in Fig. 8(c) is used. In this case, the node initially in the center of the robot becomes the new node in the support polygon, and the top node of the robot remains the same both before and after rolling. We will refer to the gait with a constant internal node as the rolling gait, and the gait with the switching center node as an everting gait, as the robot seems to be everting its inside and outside as it moves. From a practical perspective, the fact that

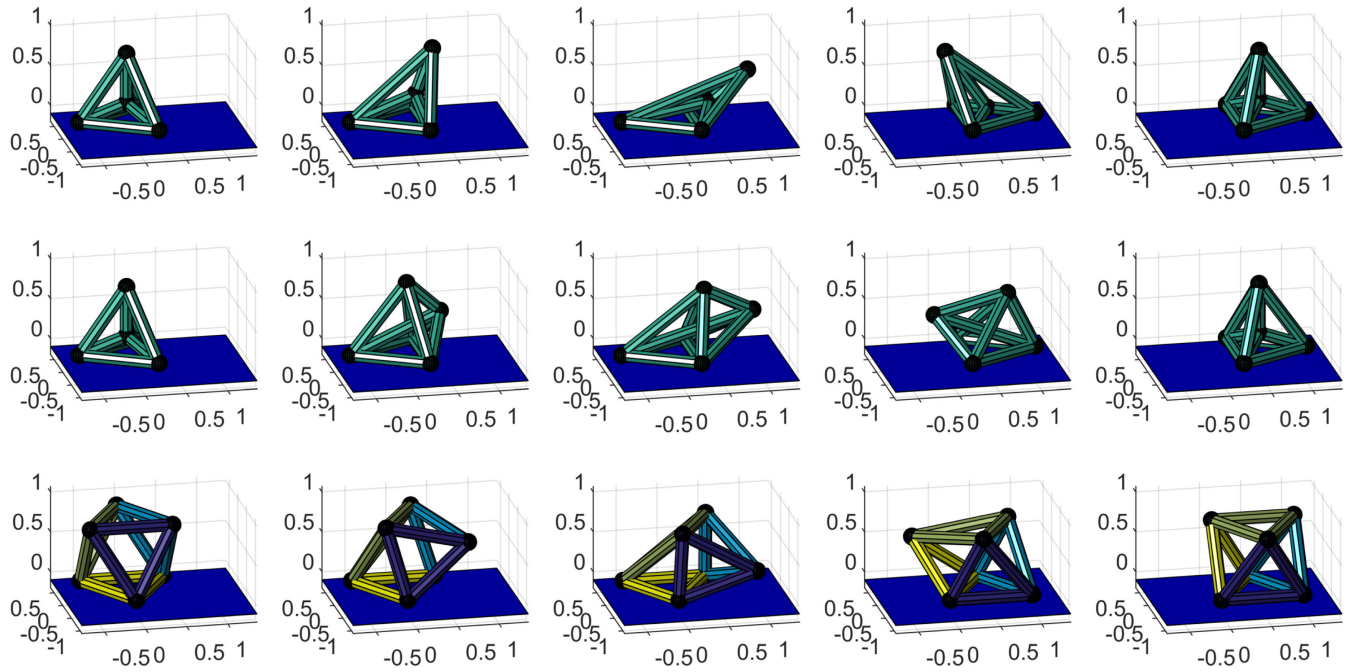

Fig. 9. Resulting motions obtained from the optimization given in (30). The top row shows a rolling gait of the K5 graph, the middle row shows an everting gait of the K5 graph, and the bottom row shows a rolling gait for an octahedral robot.

the top node of the everting gait always remains off the ground could allow it to house cameras or other components. We note that this everting gait requires an overconstrained network, as it requires that a simplex present in the initial graph switch its orientation as demonstrated in Fig. 4. The resulting motion primitive for the octahedron is shown in the third row of Fig. 9. The computation time for these offline primitives was 127, 134, and 166 s for the K5 everting gait, K5 rolling gait, and octahedron gait, respectively.

We can also compare the resulting motions in terms of cost. As computed by (30), the optimized motion primitive for the everting K5 has a cost of 0.198, the rolling K5 primitive has a cost of 0.062, and the octahedron has a cost of 0.025. Another interesting comparison between the motion primitives is the ratio of the maximum and minimum actuator length. The ratio of the overall longest actuator to the overall shortest actuator is 3.11 for the everting gait, 2.85 for the rolling gait, and 1.58 for the octahedron. These results demonstrate the need for high elongation actuators.

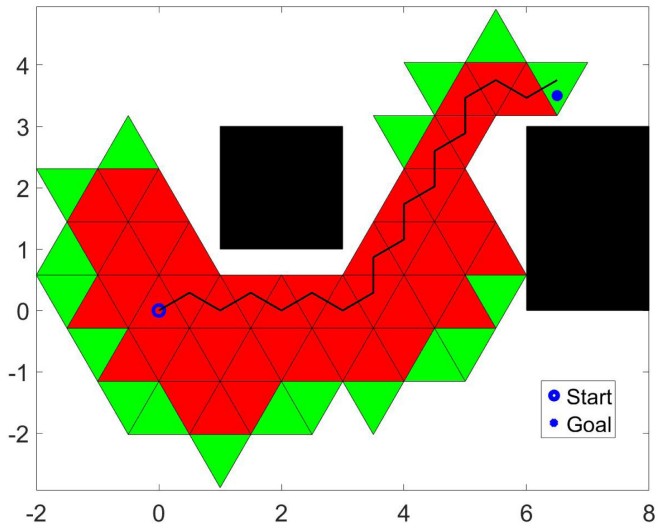

Fig. 10. Example of the A* planning algorithm. The black boxes are obstacles, the red triangles are the closed set (reachable configurations explored by A*), and the green triangles are the open set (configurations that the planner will consider adding to the closed set).

C. Optimizing a Path Over Motion Primitives

Given a motion primitive developed by the optimization, we need a planner to specify a series of support polygons from the initial configuration to the goal. This task corresponds to planning a path on a triangular grid of candidate support polygons. We use the A* algorithm for this task, but note that any discrete planning algorithm could be used for this task. An example of A* finding a path through obstacles is shown in Fig. 10. If a feasible trajectory is found that solves the optimization and a feasible path is found between the starting configuration and the goal region, a feasible path that satisfies all constraints is possible from the start to the goal region. Note that in the case of an environment with obstacles, the collision checking performed as part of the A* algorithm depends in part on the robot gait. The maximum extent of the computational gait must be used by the planner to ensure that it is possible to move from one support polygon to another. If a path of collision-free support polygons that leads from the start to the goal exists, the A* algorithm is

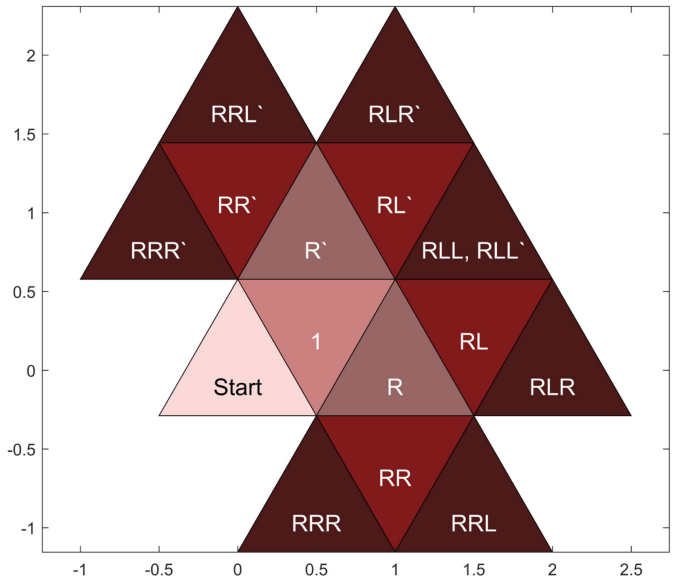

Fig. 11. Number of possible trajectories available when the robot rolls over one edge. Note that the trajectories denoted with a prime are the same as their counterparts, but with each letter switched. The marginal gain of adding more steps seems to be decreasing.

guaranteed to find it. However, it is possible that if A* fails to find a path, the robot could pass through the environment by using a different motion primitive.

D. Smoothing Between Primitives

In this section, we leverage symmetry in the robot and the triangular grid of candidate support polygons to consider motion primitives for moving between several support polygons, as opposed to just moving from one support polygon to its neighbor. We present two approaches: one where we relax the requirement to return to the symmetric configuration between every step to a requirement to return to the configuration at larger numbers of intermediate steps, and a second approach where we optimize a trajectory that enables a robot to continue in a straight line indefinitely without returning to the symmetric configuration.

1) Smoothing Over Multiple Support Polygons: In the extreme, we could optimize directly over the entire trajectory from beginning to end, but such a procedure may be expensive to compute online. Instead, we quantify the marginal gain of optimizing over trajectories of increasing length, but while maintaining the same support polygons. We note that for a robot traveling through a triangular grid, if we eliminate the option to move backward at every step, the robot can choose to roll over the left or right edge. Shown in Fig. 11 is a partial triangular grid that gives the sequence of turns to arrive at each cell, assuming the initial motion is from the "start" to the "1" cell. Each path can be represented by a $p-2$ digit binary word, where p is the number of transitions between support polygons or rolling events. By symmetry of the robot and the grid of support polygons, switching all entries in the binary word results in a mirrored trajectory. This means that for p (where $p \geq 2$) steps there are 2^{p-2} possible paths to compute. This means that

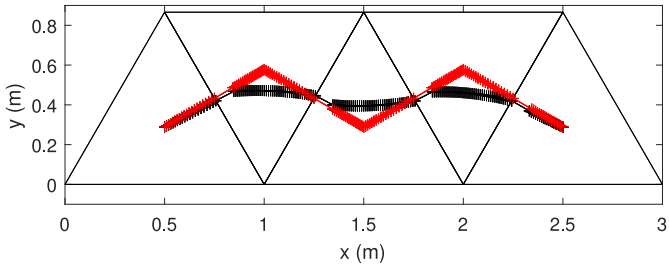

Fig. 12. Series of one step trajectories stitched together (red) compared with the smoothed optimization over four steps (black) for the rolling gait of the K5 network. Note that the path of the center of mass is more direct for the smoothed primitive than the compilation of single-step primitives.

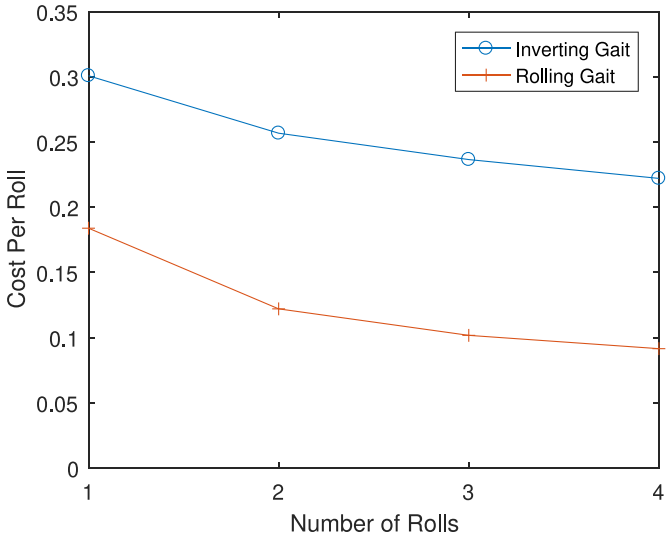

Fig. 13. Comparison of the cost per roll when optimized over multiple rolls. The cost per roll is monotonically decreasing.

for paths with two rolling events there is only a single motion primitive possible, meaning there is no loss of generality for optimizing the trajectory over two steps as opposed to a single step.

To understand the cost savings of optimizing over multiple support polygons, we compute the cost of moving 1 to 4 steps along the pattern of support polygons shown in Fig. 12, along with the direct comparison of the center of mass path when both 1 and 4 steps were used. The cost to complete a single roll is shown in Fig. 13. We note initial improvement in the cost when moving from one step to two steps, but observe diminishing returns by using longer and longer primitives. Qualitatively, the motion of the center of mass in the smoothed and unsmoothed trajectories is shown in Fig. 12. For the K5 graph, a reasonable compromise appears to be to always use the two-step motion primitives unless the robot is within one step of the goal. We illustrate combined behavior of the A* planner and the smoothed primitives to navigate between the waypoints of the "S" trajectory shown in Fig. 14.

2) Gaits With No Return to the Nominal Configuration: The smoothing methods presented previously in this section relaxed the requirement of returning to the symmetric configuration

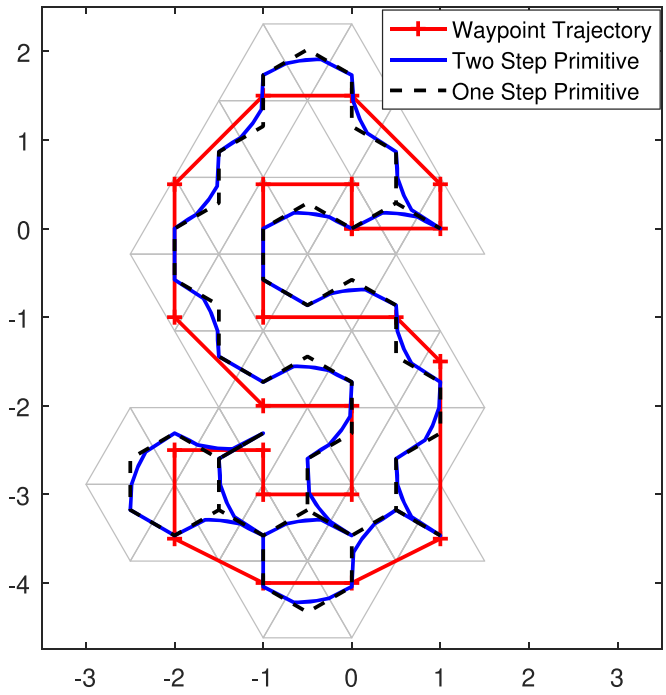

Fig. 14. Trajectories of the center of mass following the "S" trajectory when both the one-step and the two-step motion primitives are used. The support polygons are shown in gray.

from every step to every N steps, where N is some integer number of steps. An equivalent optimization approach could also be used to develop primitives that enable moving between different intermediate configurations without passing through the symmetric configuration. A key question with this approach is how to define the best intermediate configurations. One option is to include the shape of the intermediate configuration as part of the optimization itself. As a demonstration, we develop primitives that allow the octahedron and K5 robots to locomote along an arbitrarily long straight path of support polygons, similar to the motion shown in Fig. 12. We previously optimized a trajectory that starts at a given symmetric configuration and ends at equivalent symmetric configuration, whereas we now optimize a trajectory that starts at a tipping configuration and ends at an equivalent tipping configuration for the next support polygon, where the shape of the tipping configuration itself is part of the optimization. We encode the symmetry between the first and last configurations as follows:

$$Ax_0 + b = x_{N_{\text{config}}} \tag{39}$$

where A and b define a linear transform and necessary assignment of node correspondences to ensure that the initial and final configurations are equivalent. We repeat the optimization presented in (30)–(38), but including the initial configuration as one of the optimization variables, and replacing (38) with (39). The resulting gaits are demonstrated in the supplementary video. The cost of this smoothed gait for the octahedron, K5 inverting gait, and K5 rolling gait is 76%, 51%, and 46%, respectively, the cost of the repeatedly using the one-step trajectory that starts and ends in the symmetric configuration, representing a substantial savings. Utilizing these gaits online in the robot requires storing the repeating gait as well as the trajectory to move to and from the symmetric configuration to be used at the beginning and end of the straight line trajectory. This means that the memory required to store this gait is equivalent to the memory required to store a two-roll primitive. Due to the cost to move from the initial configuration to the rolling configuration, the multistep primitives, such as those shown in Fig. 12, are superior for short sequences of support polygons. However, as the length of the trajectory increases, the cost of using the repeated gaits approaches the cost obtained by optimizing over the entire trajectory, but requires a smaller amount of memory to store. Future work could seek to define intermediate gaits and other shapes that enable different behaviors, such as turning.

VI. Comparison of the Greedy and Two-Tiered Approach

We now compare the behaviors of the greedy, roll-unaware planning method presented in Section IV with the two-tiered planning method presented in V. We find that, on average, the two-tiered planning method finds more efficient trajectories than the greedy approach. In addition, the two-tiered planning approach always finds a successful trajectory if a sequence of support polygons exists that leads to the goal, whereas the greedy approach is often unable to find a successful trajectory. However, we note that the one-step planning method applies to every infinitesimally rigid robot, whereas the two-tiered planning approach applies only to robots of a restrictive symmetry class.

Conceptually, we can compare the behavior of the two planners by comparing the resulting center of mass trajectories in Figs. 6 and 14. With the two-tiered planning approach, the trajectory of the center of mass takes a less direct path between waypoints, as the constraint to move the support polygon along the triangular grid ensures that the center of mass does not move in a straight line. Despite the apparent inefficiency of the trajectory from the two-tiered planner, we find that it results in lower cost trajectories. We hypothesize that this occurs because the robot remains in a better conditioned state. The two-tiered planner generates trajectories with consistent motion of all of the free nodes, whereas in the trajectories of the greedy planner, free nodes seem to be flailing about a relatively steady center of mass trajectory.

For a quantitative comparison of the performance of the planners, we generated 100 sequences of 5 random waypoints in a 5 unit by 5 unit region and use the proposed planning methods to find a trajectory to visit the waypoints sequentially. As the output of the optimization is a kinematic trajectory, we scale the trajectories such that completing the entire trajectory takes 1 unit of time, and convert the trajectory to continuous time by linearly interpolating the node positions between the discrete configurations returned by the optimization. This rescaling ensure an equivalent average velocity between the different experiments, and allows a direct comparison in terms of the cost. To evaluate the cost of the trajectories, we use the following cost function:

$$J = \frac{\int_0^1 \|\dot{L}(x(t))\|^2 dt}{d} \tag{40}$$

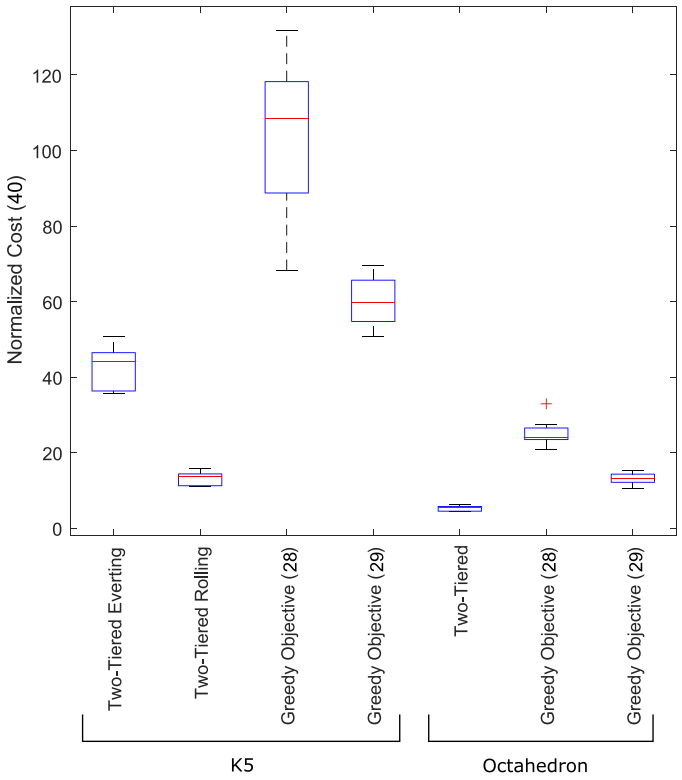

Fig. 15. Comparison of the results obtained by applying both the greedy planning methods and the two-tiered planning method. The two-tiered planning method resulted in trajectories with the lowest cost. With more nodes, the octahedron is also more efficient than the K5 graph.

where d is the sum of the straight line distances between the waypoints. This cost is a continuous time version of (28), divided by the path length to give an efficiency metric as the average cost to move a unit distance. Using both the K5 and the octahedron robot, Fig. 15 compares the efficiency of the paths resulting from the two-tiered planning method (using both the rolling and everting primitive for the K5 graph), and the greedy method using both (28) and (29) as the objective. For both the octahedron and the K5 graph, the two-tiered planning approach attains lower cost and lower variance than the greedy approach for the same robot. Interestingly, for both the octahedron and the K5 graph, a lower overall cost is obtained by using (29), the cost function that penalizes actuator for deviating from a nominal length, as the cost as opposed to (28), which penalizes changes in actuator length at each time step and is the single-step version of (40). This seems to indicate that long-term efficiency is achieved by keeping the robot in a relatively well-conditioned state.

In addition to cost, the other key criteria by which to evaluate the planners is their ability to find a complete path without violating constraints. For the 100 randomly generated trajectories and using the K5 robot, the one-step optimization method successfully found a path between all waypoints for 12% of the trials when using (28) as the objective, and for 70% of the trials when using the formation-control based objective given in (29). We repeated the experiments with the octahedron, and found convergence occurred for 85% of the trajectories when using the objective in (28), and 75% when using the objective in (29). Interestingly, for the K5 graph, the formation objective (29) leads to more frequent convergence than the minimum norm objective (28), but for the octahedron, the results are reversed. The most common failure mode for the K5 graph is rolling onto a support polygon with extremely long actuator lengths between the support nodes, and then being unable to move the remaining nodes far enough to cause a tip without violating constraints. The use of the formation objective that seeks to keep the edge lengths at a nominal operating point tends to avoid this scenario. For the octahedron, failure most commonly occurred through inability to satisfy the rigidity maintenance constraint. The approximately equal edge lengths favored by the formation objective seem to be slightly more likely to put the robot into a configuration with low rigidity.

The two-tiered planning approach has the valuable guarantee of persistent feasibility and performs better than the one-step method in terms of cost, probably because the robot remains in a relatively well conditioned state throughout the motion. However, this method only applies to robots that meet strict symmetry requirements. The one-step method is general to any infinitesimally rigid robot, but contains no guarantee of persistent feasibility.

VII. Translating a Quasi-Static Plan to a Dynamic Robot

The planning methods presented in this article provide quasi-static trajectories, but implementation on a real robotic system means that dynamic effects will be present. To address this discrepancy, we utilize the quasi-static trajectories that are a result of the optimization as an input to a controller that forms a part of a dynamic simulation. We use the inverse kinematic to transform the trajectories in terms of node positions ($x(t)$) into trajectories of desired actuator lengths ($L_d(t)$) and actuator velocities ($\dot{L}_d(t)$), and then utilize a simple PID controller to compute the force ($\tau(t)$) to be applied by each actuator as follows:

$$\tau(t) = K_p e(t) + K_d \dot{e}(t) + K_I \int_0^t e(t)dt \tag{41}$$

where $e(t) = L(t) - L_d(t)$.

For this controller, the robot only has knowledge of the lengths of its actuators, and requires no knowledge of the position of its nodes in space. For this proof-of-concept implementation, we assume that each actuator can be approximated as having half of its mass at each node. Knowing the forces applied by the actuator, we determine the forces on each node as follows:

$$M\ddot{x} = F_{\text{tot}} = \begin{bmatrix} R(x)^T C^T \end{bmatrix} \begin{bmatrix} \tau(t) \\ E(t) \end{bmatrix} + F_{\text{ext}} \tag{42}$$

where $E(t)$ are the ground reaction forces and F_{ext} are the external forces applied on the robot, which in this case is the gravity acting on each node. We solve for $E(t)$ such that the resultant force (F_{tot}) on the ground nodes are equal to 0. Rolling occurs when the reaction force at any ground node is negative in the vertical direction. When his occurs, we remove the node

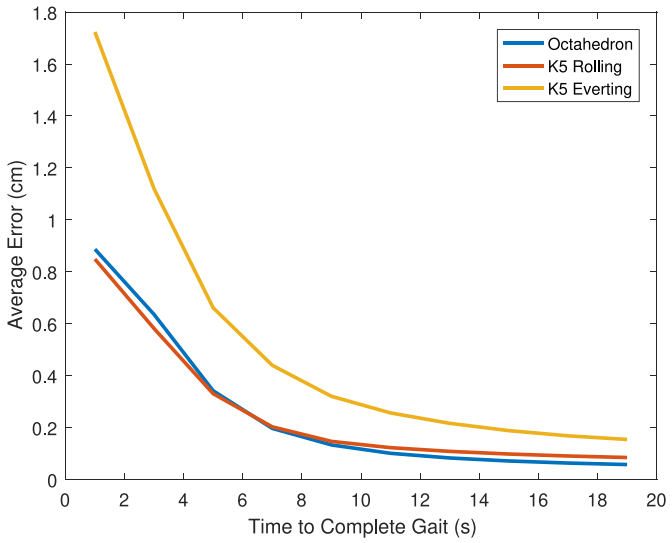

Fig. 16. Comparison of the average error between the planned quasi-static trajectory and the position of the nodes in a dynamic simulation where the quasi-static input serves as an input to a PID controller. The error is compared against the overall time that it takes to complete the trajectory. The average error decreases as the movement proceeds more slowly.

from the support polygon (and the corresponding constraints from the C matrix) and continue propagating the dynamics until a new node makes contact with the ground. We assume that all collisions with the ground are perfectly inelastic, meaning that the velocity of the node that contacts the ground immediately becomes 0. We use MATLAB ode45 to perform the dynamic simulation.

To evaluate our system, we utilize the gaits shown in Fig. 9 as inputs to the PID controller. The videos of the simulated dynamics are shown in the supplementary video. For these simulations, we assume that the initial length of the longest actuator in each robot has a length of 1 m, and that each node has a mass of 1 kg. To evaluate the gaits, we compute the average error between the nodes in the dynamic simulation and their expected location from the quasi-static plan. As the quasi-static gaits do not account for the rolling behavior, we do not compute the error for the portion of the trajectory where only two nodes of the dynamic robot are on the ground. Fig. 16 shows how the average error changes based on how many seconds it takes to execute the gait. If the gaits are performed in 6 s or longer, the average error is below 1 cm for each of the different gaits. The exact magnitude of the error will depend on the tuning of the controller, the size, and mass properties of the robot, and characteristics of the gait, but these experiments demonstrate the overall trend of better agreement between the quasi-static plan and the physical trajectory as the overall speed of the robot decreases. These demonstrations also illustrate that the quasi-static trajectories lead to useful behaviors in a fully dynamic system.

VIII. Conclusion

In this article, we derived the differential kinematics for networks of linear actuators connected at universal joints and shown that if the embedded graph describing the robot is infinitesimally rigid, any desired motion of the nodes can be achieved through some motion of the edges. We then framed the locomotion problem as a nonlinear optimization over the node positions while enforcing constraints that guarantee the feasibility of the robot. We also discussed that constraints to maintain the infinitesimal rigidity of the robot tend to divide the state space of the robot into separated regions, even though the singular configurations themselves make up a set of zero measure. We discussed the control of both minimally rigid graphs and overconstrained graphs, and demonstrated that overconstrained graphs can achieve some behaviors that minimally rigid graphs cannot, such as the everting locomotion gait of the K5 graph. We presented two planning schemes: one where we solved a single-step nonlinear optimization online to achieve a desired instantaneous motion of the center of mass, and another where we optimized over many configurations that compose a motion primitive, including in the optimization direct consideration of the rolling behavior. The single-step approach was applicable to robots of arbitrary configuration, but there was the possibility that the robot will reach a state from which it cannot continue in the desired direction without violating physical constraints. While there was no guarantee of persistent feasibility with this approach, we found that long trajectories can be achieved based on the choice of the cost function. The two-tiered approach ensured persistent feasibility, but required the robot to satisfy certain symmetry properties.

In future work, we will attempt to blend the properties of these two control approaches, namely by finding ways to guarantee persistent feasibility for robots composed of linear actuators in arbitrary infinitesimally rigid configurations. We will also extend our planning approach to directly consider dynamic effects, explicitly considering forces in the members, and incorporating the inertia properties of the system. One possibility is to use these kinematic trajectories as starting points for a dynamic model, similar to the methods in [31]. We will also explore taking the centralized controllers presented in this article and finding a distributed version of a similar controller, where computation is performed locally at the actuators in the system.

In future work, we will demonstrate this system with novel robotic hardware of the type presented in [17]. We have noted that the current optimization procedure does not have any guarantees on global optimality. We will work to find convex relaxations of the nonconvex constraint such that we are able to find suboptimality guarantees for certain frameworks.

References

[1] G. J. Hamlin and A. C. Sanderson, "TETROBOT: A modular approach to parallel robotics," *IEEE Robot. Autom. Mag.*, vol. 4, no. 1, pp. 42–50, Mar. 1997.

[2] W. H. Lee and A. C. Sanderson, "Dynamics and distributed control of Tetrobot modular robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1999, vol. 4, pp. 2704–2710.

[3] W. H. Lee and A. Sanderson, "Dynamic rolling locomotion and control of modular robots," *IEEE Trans. Robot. Autom.*, vol. 18, no. 1, pp. 32–41, Feb. 2002.

[4] B. K. Wada, J. L. Fanson, and E. F. Crawley, "Adaptive structures," *J. Intell. Mater. Syst. Struct.*, vol. 1, no. 2, pp. 157–174, 1990.

[5] P. C. Hughes, W. G. Sincarsin, and K. A. Carroll, "Trussarm—A variable-geometry-truss manipulator," *J. Intell. Mater. Syst. Struct.*, vol. 2, no. 2, pp. 148–160, 1991.

[6] S. Curtis *et al.*, "Tetrahedral robotics for space exploration," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 22, no. 6, pp. 22–30, Jun. 2007.

[7] C.-H. Yu, K. Haller, D. Ingber, and R. Nagpal, "Morpho: A self-deformable modular robot inspired by cellular structure," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2008, pp. 3571–3578.

[8] J. C. Zagal, C. Armstrong, and S. Li, "Deformable octahedron burrowing robot," in *Proc. Int. Conf. Synthesis Simul. Living Syst.*, 2012, pp. 431–438.

[9] A. Mazzone and A. Kunz, "Sketching the future of the smartmesh wide area haptic feedback device by introducing the controlling concept for such a deformable multi-loop mechanism," in *Proc. IEEE Joint Eurohapt. Conf. Symp. Haptic Interfaces Virtual Environ. Teleoper. Syst.*, 2005.

[10] R. Kovacs *et al.*, "TrussFormer: 3D printing large kinetic structures," *Proc. ACM Symp. User Interface Softw. Technol.*, 2018, pp. 113–125.

[11] M. Pieber, R. Neurauter, and J. Gerstmayr, "An adaptive robot for building in-plane programmable structures," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2018, pp. 5320–5327.

[12] A. Spinos and M. Yim, "Towards a variable topology truss for shoring," in *Proc. IEEE Ubiquitous Robots Ambient Intell.*, 2017, pp. 244–249.

[13] A. Spinos, D. Carroll, T. Kientz, and M. Yim, "Variable topology truss: Design and analysis," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 2717–2722.

[14] A. Sofla, D. Elzey, and H. Wadley, "Shape morphing hinged truss structures," *Smart Mater. Struct.*, vol. 18, no. 6, pp. 065012–065020, 2009.

[15] A. Lyder, R. F. M. Garcia, and K. Stoy, "Mechanical design of odin, an extendable heterogeneous deformable modular robot," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2008, pp. 883–888.

[16] F. Collins and M. Yim, "Design of a spherical robot arm with the spiral zipper prismatic joint," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2016, pp. 2137–2143.

[17] Z. Hammond, N. Usevitch, E. Hawkes, and S. Follmer, "Pneumatic reel actuator: Design, modeling, and implementation," in *Proc. Int. Conf. Robot. Autom.*, 2017, pp. 883–888.

[18] J. Bruce *et al.*, "SUPERball: Exploring tensegrities for planetary probes," in *Proc. 2th Int. Symp. Artif. Intell., Robot., Autom. Space*, 2014.

[19] A. P. Sabelhaus *et al.*, "System design and locomotion of SUPERball, an untethered tensegrity robot," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 2867–2873.

[20] M. Abrahantes, A. Silver, and L. Wendt, "Gait design and modeling of a 12-tetrahedron walker robot," in *Proc. 39th Southeastern Symp. Syst. Theory*, 2007, pp. 21–25.

[21] X. Wang, X. Wang, Z. Zhang, and Y. Zhao, "Motion planning of kinematically redundant 12-tetrahedral rolling robot," *Int. J. Adv. Robot. Syst.*, vol. 13, no. 23, 2016.

[22] X. Wang, X. Wang, and Z. Zhang, "Dynamical modelling and a decentralized adaptive controller for a 12-tetrahedral rolling robot," *Trans. FAMENA*, vol. 42, no. 2, pp. 51–66, 2018.

[23] S. Jeong *et al.*, "Variable topology truss: Hardware overview, reconfiguration planning and locomotion," in *Proc. IEEE Int. Conf. Ubiquitous Robots*, 2018, pp. 616–621.

[24] K. Kim, A. K. Agogino, A. Toghyan, D. Moon, L. Taneja, and A. M. Agogino, "Robust learning of tensegrity robot control for locomotion through form-finding," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 5824–5831.

[25] C. Paul, F. J. Valero-Cuevas, and H. Lipson, "Design and control of tensegrity robots for locomotion," *IEEE Trans. Robot.*, vol. 22, no. 5, pp. 944–957, Oct. 2006.

[26] J. Friesen, A. Pogue, T. Bewley, M. de Oliveira, R. Skelton, and V. Sunspiral, "DuCTT: A tensegrity robot for exploring duct systems," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 4222–4228.

[27] A. P. Sabelhaus *et al.*, "Inverse statics optimization for compound tensegrity robots," *IEEE Robot. Autom. Lett.*, vol. 5, no. 3 pp. 3982–3989, 2020.

[28] X. Xu, F. Sun, Y. Luo, and Y. Xu, "Collision-free path planning of tensegrity structures," *J. Struct. Eng.*, vol. 140, no. 4, 2013, Art. no. 04013084.

[29] S. M. LaValle and J. J. Kuffner, Jr., "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, 2001.

[30] Z. Littlefield, K. Caluwaerts, J. Bruce, V. SunSpiral, and K. E. Bekris, "Integrating simulated tensegrity models with efficient motion planning for planetary navigation," in *Proc. Int. Symp. AI, Robot, Autom. Space*, 2016.

[31] Z. Littlefield, D. Surovik, W. Wang, and K. E. Bekris, "From quasi-static to kinodynamic planning for spherical tensegrity locomotion," in *Proc. Int. Symp. Robot. Res.*, 2017, pp. 947–966.

[32] C. Paul, J. W. Roberts, H. Lipson, and F. V. Cuevas, "Gait production in a tensegrity based robot," in *Proc. Int. Conf. Adv. Robot.*, 2005, pp. 216–222.

[33] A. Iscen, A. Agogino, V. SunSpiral, and K. Tumer, "Controlling tensegrity robots through evolution," in *Proc. 15th Annu. Conf. Genetic Evol. Comput.*, 2013, pp. 1293–1300.

[34] M. Zhang *et al.*, "Deep reinforcement learning for tensegrity robot locomotion," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 634–641.

[35] J. Rieffel and J.-B. Mouret, "Adaptive and resilient soft tensegrity robots," *Soft Robot.*, vol. 5, no. 3, pp. 318–329, 2018.

[36] C. Rennie and K. E. Bekris, "Discovering a library of rhythmic gaits for spherical tensegrity locomotion," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 2290–2295.

[37] K. Caluwaerts, M. D'Haene, D. Verstraeten, and B. Schrauwen, "Locomotion without a brain: Physical reservoir computing in tensegrity structures," *Artif. Life*, vol. 19, no. 1, pp. 35–66, 2013.

[38] B. Cera and A. M. Agogino, "Multi-cable rolling locomotion with spherical tensegrities using model predictive control and deep learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 1–9.

[39] D. Surovik, K. Wang, M. Vespignani, J. Bruce, and K. E. Bekris, "Adaptive tensegrity locomotion: Controlling a compliant icosahedron with symmetry-reduced reinforcement learning," *Int. J. Robot. Res*, pp. 1–22, 2019. [Online]. Available: https://journals.sagepub.com/doi/10.1177/0278364919859443

[40] M. Vespignani, C. Ercolani, J. Friesen, and J. Bruce, "Steerable locomotion controller for six-strut icosahedral tensegrity robots," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2018, pp. 2886–2892.

[41] K. Caluwaerts *et al.*, "Design and control of compliant tensegrity robots through simulation and hardware validation," *J. Roy. Soc. Interface*, vol. 11, no. 98, 2014, Art. no. 20140520.

[42] N. Usevitch, Z. Hammond, S. Follmer, and M. Schwager, "Linear actuator robots: Differential kinematics, controllability, and algorithms for locomotion and shape morphing," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 5361–5367.

[43] S. Park, E. Park, M. Yim, J. Kim, and T. Seo, "Optimization-based nonimpact rolling locomotion of a variable geometry truss," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 747–752, Apr. 2019.

[44] L. Krick, M. E. Broucke, and B. A. Francis, "Stabilisation of infinitesimally rigid formations of multi-robot networks," *Int. J. Control*, vol. 82, no. 3, pp. 423–439, 2009.

[45] L. Asimow and B. Roth, "The rigidity of graphs," *Trans. Amer. Math. Soc.*, vol. 245, pp. 279–289, 1978.

[46] S. Pellegrino and C. R. Calladine, "Matrix analysis of statically and kinematically indeterminate frameworks," *Int. J. Solids Struct.*, vol. 22, no. 4, pp. 409–428, 1986.

[47] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE J. Robot. Autom.*, vol. 4, no. 2, pp. 193–203, Apr. 1988.

[48] D. Zelazo, A. Franchi, H. H. Bülthoff, and P. R. Giordano, "Decentralized rigidity maintenance control with range measurements for multi-robot systems," *Int. J. Robot. Res.*, vol. 34, no. 1, pp. 105–128, 2015.

[49] D. Zelazo, A. Franchi, F. Allgöwer, H. H. Bülthoff, and P. R. Giordano, "Rigidity maintenance control for multi-robot systems," in *Proc. Robot.: Sci. Syst.*, 2012, pp. 473–480.

[50] M. H. Trinh, M.-C. Park, Z. Sun, B. D. Anderson, V. H. Pham, and H.-S. Ahn, "Further analysis on graph rigidity," in *Proc. IEEE Conf. Decis. Control*, 2016, pp. 922–927.

[51] H. Gluck, "Almost all simply connected closed surfaces are rigid," in *Geometric Topology*. Berlin, Germany: Springer, 1975, pp. 225–239.

[52] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

Nathan S. Usevitch received the B.S. degree in mechanical engineering from Brigham Young University, Provo, UT, USA, in 2015, and the M.S. degree in mechanical engineering in 2017 from Stanford University, Stanford, CA, USA, where he is currently working toward the Ph.D. degree in mechanical engineering.

His research interests include soft robotics, haptics, truss robots, and multirobot systems.

Zachary M. Hammond received the B.S. degree in mechanical engineering from the University of California, Berkeley, Berkeley, CA, USA, in 2014, and the M.S. degree in mechanical engineering in 2017 from Stanford University, Stanford, CA, USA, where he is currently working toward the Ph.D. degree in mechanical engineering.

His research interests include soft robotics, mechanism design, and grasping.

Mac Schwager received the B.S. degree from Stanford University, Stanford, CA, USA, in 2000, and the M.S. and Ph.D. degrees from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 2005 and 2009, respectively, all in mechanical engineering.

He is currently an Assistant Professor with the Aeronautics and Astronautics Department, Stanford University. He was a Postdoctoral Researcher working jointly with The GRASP Lab, University of Pennsylvania, Philadelphia, PA, USA and Computer Science and Artificial Intelligence Laboratory, MIT, from 2010 to 2012, and was an Assistant Professor with Boston University, Boston, MA, USA, from 2012 to 2015. His research interests include distributed algorithms for control, perception, and learning in groups of robots, and models of cooperation and competition in groups of engineered and natural agents.

Prof. Schwager was the recipient of the NSF CAREER award in 2014, the DARPA YFA in 2018, a Google Faculty Research Award in 2018, and the IROS Toshio Fukuda Young Professional Award in 2019.