UNIVERSITY OF CALIFORNIA

SANTA CRUZ

KINEMATICS ALGORITHMS FOR TENSEGRITY STRUCTURES

A thesis submitted in partial satisfaction
of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

STEVEN BURT

March 2013

The Thesis of STEVEN BURT
is approved:

_____

Professor Gabriel Elkaim, Chair

_____

Professor Jacob Rosen

_____

Vytas SunSpiral

_____

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Kinematics Algorithms for Tensegrity Structures

by

Steven Burt

Tensegrity-based Structures hold promise for the field of lightweight, compliant robots. However, most prior efforts to model and plan the shape of these structures have focused on special cases or on static structures. This work attempts to generalize a dynamic-relaxation form-finding method into a form that provides solutions to problems analogous to Forward and Inverse Kinematics problems in serial-chain robots. Details of the implementation of these algorithms into a model tensegrity robot are also presented, as well as suggestions for experimental validation of results.

This work would not have been possible without the seemingly endless vision, enthusiasm, and optimism of Vytas SunSpiral.

1 Introduction

Tensegrities are a family of load-bearing, pin-jointed, space frame structures characterized by a large number of members that bear tensile loading and a limited number of (typically discontinuous) members that bear compressive loads. In contrast to classic space-frames, they are often over-defined and statically indeterminate, and gain rigidity through pre-stress [15]. Originally developed by the artist Kenneth Snelson, the term *tensegrity* was coined by Buckminster Fuller as a portmanteau of "tensile integrity".

The traditional tensegrity configuration uses discontinuous compressive members; at each point of connection there will be a single compressive member and multiple tensile members. In this case, connection points can be considered to be pin- or ball-jointed and no moments can be imparted to the members; this results in members which are loaded purely axially. This allows the members to be slender, and the resulting tensegrity structures can be visually striking. These unique visual properties led to their use in sculpture and architecture, and both Snelson and Fuller created several large installations that were either pure tensegrities or else used tensile members to a greater extent than had been found in traditional architecture [9], [20].


: Photo: Snelson

Figure 1: *Sleeping Dragon* by Kenneth Snelson (1982)

However, for several reasons they have found little use in large structures or architecture. First, it is difficult to achieve high structural stiffness with tensegrity structures. To push the ends of tensile members into place, it is necessary for the compressive members reach the full dimension of the structure. Thus, any structure will either have compressive members that have a large length, or be comprised of multiple smaller units. However, if the compressive members are made long, they may be subject to buckling or other deformation that reduces the rigidity of the structure. Similarly, if the structure is composed of many smaller units, it's rigidity will be reduced through the accrual of deflections between sections.

The rigidity of tensegrity structures is also a function of prestress and high levels of pre-stress may be needed to obtain a satisfactory rigidity. However, as pre-stress is increased, the capacity of the structure to respond to external load is decreased, and at the extreme case it may become impossible for a structure to achieve a particular goal for rigidity and load capacity.

Static structures also fail to make full use of the compelling feature of tensegrity: the pure axial loading of all members in response to any external load. In the case of a large static structure, the external loads will likely be primarily static or limited to a small range of load cases (for example in the case a building where self-weight provides most of the load and external variable forces such as wind would be comparatively small). Thus the load paths within the structure would be fixed and the structure can be optimized to carry these loads. Since many common building materials have a much greater compressive strength than tensile strength, static structures are often designed to carry the bulk of the load through a compressive path rather than tensile paths.

In conjunction with the challenges in design and construction, tensegrity structures have not proven attractive for the vast majority of architectural or structural purposes in comparison to more traditional designs; exceptions are typically cases where


: Photo: Arup

Figure 2: Kurilpa Bridge, Brisbane, Australia

the lightweight appearance of the tensegrity structure produces a design with unique or interesting aesthetic properties.

Despite their lack of widespread application, over the past 60 years tensegrities and related structures (such as suspended cable-networks) continue to have been studied, first within the structural engineering community and more recently within the robotics community. The robotics interest has proceeded in concert with more widespread interest in soft-bodied and other biologically-inspired robotics.

Tension networks are apparent in biological organisms at multiple scales. Ingber [10] first proposed the tensegrity model of animal cells, in which microtubules carry compressive loads and filaments carry tensile loads; cellular motion is produced by molecular processes that alter the lengths of these filaments. At the larger scale, researchers [24] have proposed tensegrity models for the vertebrate musculoskeletal system. Although this is complicated by features such as bursae, even if the structure is not a pure tensegrity it is clear that many of the loads experienced by the bodies of vertebrates are borne through tensile structures such as muscles, ligaments, and tendons.

In contrast to architectural structures, robotic systems often must respond to a variety of external loadings while maintaining light weight. In robotic design, stiffness is often a desired trait, but the goal is really controllability. Lately, roboticists have become interested in adding compliance to robots as both a means to protect them (and

their surroundings) from damage; these efforts are important as robots move into less controlled workspaces. In rigid arms, this may come in the form of joint compliance, where the structures connecting the joints remain stiff while compliance is built into the actuators; this typically requires torque-limiting controllers and efficient, back-drivable gear trains.

Alternatively, we can seek to build compliance into the structure itself. Tensegrity structures with infinitesimal modes make good candidates for this approach; as the structure deforms the load paths are naturally routed towards less-loaded elements in the structure. In prestressed tensegrities, it is even possible to build structures with non-linear elasticities as particular tensile elements become slack. This allows for structures that can do things such as collapse in the case of an impact, and then return to their prior state, undamaged, once the external force is removed.

In order to develop robotic systems that utilize tensegrity, techniques for modeling and motion planning are needed. As will be shown, solutions for these problems are non-obvious and non-trivial. However, much of the prior work in this field has been done by the structural engineering community, and the terminology and problems solved may not directly apply to the problems faced by roboticists. This work seeks to concretely define the problems faced by roboticists using terminology and concepts prevalent in classical robotics, thus presenting a framework to which techniques described in the structural engineering literature can be adapted and made accessible to the robotics community. In particular, this work examines tensegrity structures that are well-represented by the classical tensegrity model of a network of struts and tendons, and which achieve motion by changing tendon (and/or strut) lengths. In other words, these robots have a joint space consisting of tendon/strut lengths, and a configuration space represented by the locations of the nodes connecting the members. Thus we see the need for algorithms for forward kinematics which determine the position of the

robot based on its member lengths, and the need for inverse kinematics algorithms that determine member lengths based on a desired goal position.

2 Related Work

In the beginning, tensegrities were designed through geometric intuition and development of small physical models. More rigorous analysis of tensegrity structures began in the mid-1970s with work of Calladine [4] and Pellegrino [17]. These works dealt with tensegrities as static structures.

Several problems are associated with the static analysis of structures: determining whether a particular structure is in equilibrium, finding the equilibrium position of a given structure, identifying finite and infinitesimal modes of motion in the structure; determining a structures stiffness, and determining whether a particular structure is a minimal stable structure (i.e. whether members can be removed while retaining stability).

Tensegrity structures can be described as as weighted connected graphs where vertices (representing nodes) are connected by multiple edges (representing the members). These graphs are realized in 2- or 3-dimensions by attaching coordinates to the nodes such that the edge weights represent the member lengths. Members may be *struts*, capable of withstanding compressive loads, or *tendons*, capable of withstanding tensile loads.

Form-finding problems thus involve finding stable equilibrium states (i.e nodal positions, member lengths, internal forces, and connectivity) given some desired traits or knowledge about the structure. Typically, node positions or member lengths are given, and the form-finding algorithm seeks to find an internal stress state at equilibrium that provides self-stress and attendant stabilization of infinitesimal modes.

Much of the literature has focused on regular or symmetric tensegrity structures, structures where member lengths are of equal length and/or nodes occupy the vertices of regular polyhedra. This reduces the complexity of the form-finding problem. Many works also make assumptions such as there being no external forces. Motro [16] presents many canonical tensegrities as well as repeating structures built from multiple smaller segments.

Obviously, the form-finding problem is closely related to forward kinematics, and certain form-finding algorithms can be applied to the forward kinematics problem. Section 5 will further expand on the application of form-finding problems to our forward kinematics problem.

Tibert and Pellegrino [26] present a comparison of form-finding algorithms for tensegrities. They classify form-finding methods as either *static* or *kinematic*, depending on whether node locations and member lengths are altered during the running of the algorithm. Juan and Tur [11] extend this work, describing 3 kinematic algorithms, 6 static algorithms, and 4 other algorithms. Kinematic algorithms are suited to a condition where node locations are initially unknown, whereas static algorithms fix the node locations and find an internal force state that puts the structure in static equilibrium with self-stress.

The kinematic methods described in Juan and Tur consist of an analytic approach, presented in Connolly and Terrell [5], which makes use of symmetry to reduce the complexity of the problem, non-linear optimization approaches presented by Pellegrino [17], and a dynamic relaxation approach described by Zhang, Maurin and Motro [28]. Dynamic relaxation solves for equilibrium states by adjusting node positions in accordance with forces exerted on the nodes, and ends when the sum of forces at each node reaches zero.

A number of static methods are described; however in general these are not ap-

plicable to movable structures because they require that all node locations to be known *a priori*. One static method, generally referred to as the Force Density Method and described by Linkwitz [13], has been modified by Estrada, *et al* [7] to iterate while adjusting node positions. This allows solutions to be found when the node positions are initially unknown. However, it does not allow for specification of member lengths.

A final kinematic method, not described in Juan and Tur, has been presented in several works by So and Ye [21, 23, 22]. This formulates the problem as a semi-definite programming problem to find node locations only when member lengths are known. However, it does not consider internal or external forces and relies on rank conditions (as presented by Pellegrino) to assure stability.

When considering tensegrity structure dynamics and control, most literature focuses on specific structures and present algorithms suited to those particular structures. Sultan, Corless, and Skelton [25] present a tensegrity flight simulator, and derive control laws surrounding a fixed central point through analytical derivation. Mohr and Arsenault [14] present kinematic analysis of the octahedral tensegrity, again through analytical derivation.

Probably one of the first efforts to build a tensegrity structure and compare its actual physical characteristics with those predicted by a model is presented by Fest, *et al* [8]. A large tensegrity structure was designed and modeled using dynamic relaxation. Characteristics such as equilibrium node positions and the amount of deformation in response to loading were predicted from the model. A large structure (approximately 4.0 meters in diameter) was built and extensively instrumented. This effort showed good agreement between the model predictions and what was observed in the physical structure.

Section 5.2 further compares many of the forward kinematics algorithm, and ultimately focuses on dynamic relaxation as a suitable algorithm. For inverse kinematics,

none of the control algorithms reviewed are general-purpose, and new algorithm will be presented.

3 Work Presented

This work is composed of several parts: first, a set of parameters that can be used to describe a tensegrity structure is presented. These parameters are used to unambiguously define the configuration and state of a tensegrity structure; much in the way DH-Parameters [6] could be used to define the configuration and state of a jointed limb. These will be called "TR-Parameters" (Tensegrity Robot Parameters). These parameters are used as inputs for the form-finding algorithm presented in Section 5. This algorithm, entitled Tensegrity Forward Kinematics (TFK), is used as a forward kinematics algorithm for tensegrity structures. Several examples of structures are given to

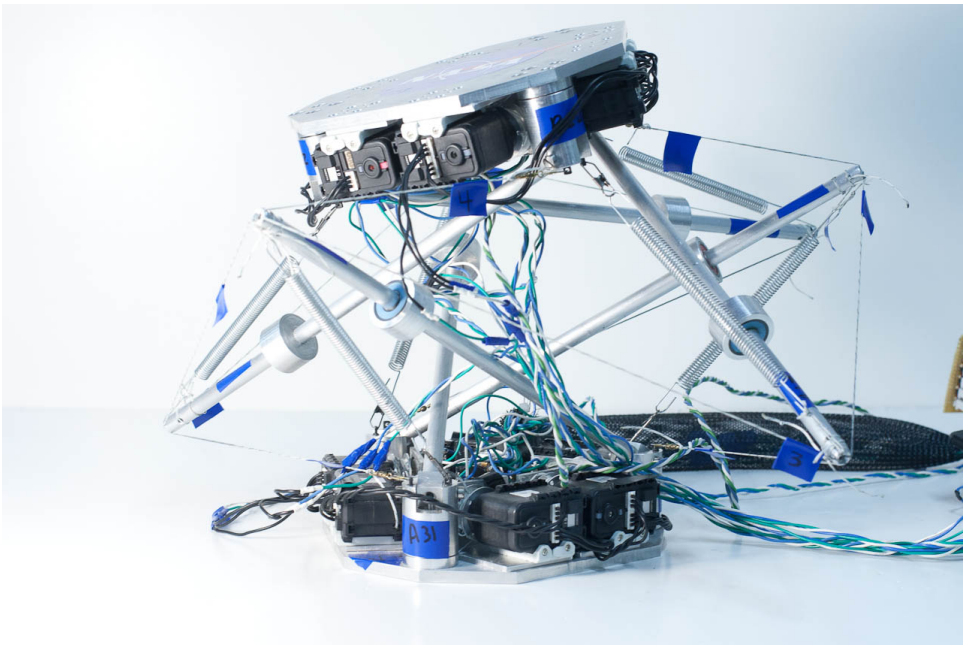


Photo:Author

Figure 3: TenseBot

show the algorithm's viability for use in a variety of systems.

Section 6 presents a scheme for inverse kinematics, entitled Tensegrity Inverse Kinematics (TIK), to place a given tensegrity structure in a desired position. This method uses inverse Jacobians.

These kinematic solvers were implemented on the "TenseBOT", a small tensegrity structure with adjustable tendon lengths, shown in 3. Section 7 describes this implementation.

4 Tensegrity Robot Parameters

Earlier works on tensegrity have tended to describe structures verbally or qualitatively, typically by referring to the solid prism with the same number of vertices, or through description of the overall structure [18]. These are sufficient to give a general indication of the shape of a tensegrity or for discussion of simple, regular tensegrities in the abstract sense. However, as we move to non-regular tensegrities, more complex shapes, or even seek to compare regular tensegrity shapes with different connectivities, a clearer and more exact descriptor is needed. This should provide a minimal set of parameters that would be needed to unambiguously define a structure. This goal is accomplished through the introduction of "TR-Parameters", which is a specific quantitative description of the structure. TR-Parameters encapsulate a set of basic parameters that describe the structural configuration of a tensegrity robot in a generic manner that would be suitable as the input to a forward-kinematics (form-finding) algorithm, much like DH-Parameters are sufficient to define a serial-chain robot's position.

4.1 TR Parameters

TR-Parameters include both constant parameters that are specific to the system and variable parameters that may be externally controlled. In a typical application, the connectivity of the system, member elasticities and masses, and the type of each member (i.e. whether it can support compressive load) are constant parameters, and the controllable or variable parameters are the lengths of the internal members.

The connectivity of the robot expresses which nodes are linked together and enumerates the members linking them. These members may be rods (capable of carrying both tensile and compressive loads) or tendons (capable of carrying tensile loads only). Each member has an elasticity value, representing its deflection in the loaded direction as a function of load. Members are connected at nodes, which are assumed to be infinitesimal and ball-jointed, such that no torques are transmitted through nodes. Each member is also given a mass which is used to apply weight due to gravity in the TFK algorithm; the TFK algorithm also allows for other external forces to be supplied. This is necessary because the structures have some elasticity and these external forces may result in deformation to the structure.

The final set of parameters is the lengths of the members. In a typical robot application some subset of these will be fixed, and some subset will be controlled. For the forward kinematics problem these will all be known; the inverse kinematics problem seeks to solve for the member lengths of the adjustable members. Notation for each of the TR-Parameters is shown in Table 1.

The connectivity or adjacency matrix, $\mathbf{C}$ appears quite often in the literature [19]. It is a rectangular matrix of dimension $m \times n$, where each row corresponds to a member and contains a -1 in a column corresponding to the node at one end of a member, and a 1 in the column corresponding to the node at the end of the other mem-

Table 1: TR Parameters

Symbol	Dim	Description
n	scalar	number of nodes (locations where members intersect)
m	scalar	number of members (could be a string, bar, or spring)
$\mathbf{C}$	$m \times n$	Configuration matrix. Each row corresponds to a single member, m_{ij} and the ith column and jth column are -1 and 1. The rest of the entries are zero.
$\mathbf{d}$	$m \times 1$	Vector containing the (undeformed) lengths of each member.
$\mathbf{t}$	$m \times 1$	Vector representing the type of each member, (i.e. 1 if it can support a compressive load, 0 otherwise)
$\mathbf{k}$	$m \times 1$	Vector containing the elasticities of each member.
$\mathbf{m}$	$m \times 1$	Vector of member masses.

ber. Although it is not a minimal representation, it has the advantage of being a useful form for matrix operations on the node and member matrices ($\mathbf{N}$ and $\mathbf{M}$). For example, $\mathbf{M} = \mathbf{N}\mathbf{C}^T$.

4.2 TR Parameters examples

To illustrate the definition of the TR-Parameters, three structures are demonstrated: A simple prism with three bars (the regular minimal tensegrity prism in $\mathbb{R}^3$), a four-bar prism, and the TenseBot as shown in Figure 3.

4.2.1 Three Bar Prism

The three bar prism structure is show in Figure 4. In this case, there are 3 fixed points arranged in an equilateral triangle comprising the base, three rod elements which lead from the base triangle to the upper triangle, and 9 tensile elements. Thus $\mathbf{C} \in \mathbb{R}^{9\times 6}$. This is the simplest stable tensegrity in $\mathbb{R}^3$. The TR-Parameters for this structure are shown in Table 2.

This structure has been studied extensively in the literature, and Tibert and Pel-

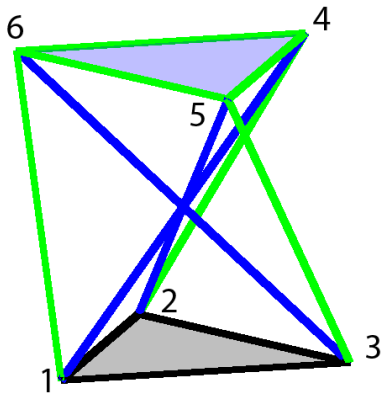


Figure 4: Minimal regular tensegrity prism

Table 2: 3-Prism TR-Parameters

$$\mathbf{C} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix} \quad \mathbf{d}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1.468 \\ 1.468 \\ 1.468 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{t} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{k} = \begin{bmatrix} 1000 \\ 1000 \\ 1000 \\ 1000 \\ 1000 \\ 1000 \\ 1000 \\ 1000 \\ 1000 \\ 1000 \\ 1000 \\ 1000 \end{bmatrix} \quad \mathbf{m} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

legrino's survey shows that according to multiple sources, an 1.468 strut-to-tendon length ratio produces an equilibrium structure with equal tendon lengths.

4.2.2 Four Bar Prism

The four-sided prism structure is show in Figure 5. In this case, there are 4 fixed nodes, and 4 nodes at the top. There are a total of 16 members. The TR Parameters for this structure are shown in table 3. Unlike triangular prisms, prisms with square faces may have inelastic compliance (that is deformation that occurs without a deformation to the internal members). As will be seen in section 5.4.2, the TFK algorithm will be able to

find a solution with these TR-Parameters as input.

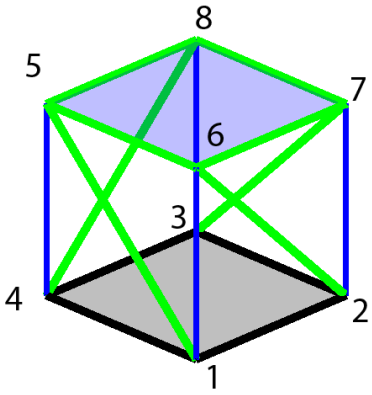


Figure 5: 4-prism

Table 3: 4-Prism TR-Parameters

$$\mathbf{C} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{d}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1.5 \\ 1.5 \\ 1.5 \\ 1.5 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{t} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{k} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{m} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

4.2.3 TenseBot

TenseBot, shown in Figure 6, is a more complicated structure with 12 nodes and a total of 30 members. TenseBot is based on the concept of a tensegrity flight simulator presented by Sultan [25] and for which control theory was explored in [12]. The Sultan model is not kinematically determinate and has finite modes; this means that it is not

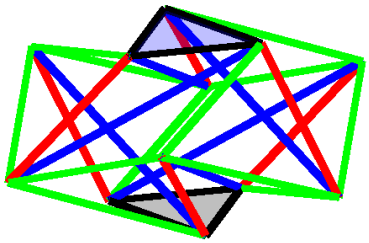


Figure 6: TenseBot

fully rigid. To rectify this, some additional members are added, making the structure analogous to the tensegrity icosohedron, with 4 tendons and one strut connected at each node. The base and top triangles, which are tendons in the tensegrity icosohedron, are struts in this model, however they generally would carry tensile as opposed to compressive loads. Thus the base and top are each rigid triangles, and there are 6 intermediate saddle nodes. There are 6 bar members which can freely pivot and rotate, 12 string tendon members (which have a high elastic modulus), and 6 spring tendon members. The TR-Parameters for TenseBot are shown in Table 4.

Table 4: TenseBot TR-Parameters

$$\mathbf{C} = \begin{bmatrix}
1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\
1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1
\end{bmatrix}
\quad
\mathbf{d}_0 = \begin{bmatrix}
5.65 \\ 5.65 \\ 5.65 \\ 5.65 \\ 5.65 \\ 5.65 \\ 12.00 \\ 12.00 \\ 12.00 \\ 12.00 \\ 12.00 \\ 12.00 \\ 8.50 \\ 8.50 \\ 8.50 \\ 8.50 \\ 8.50 \\ 8.50 \\ 8.50 \\ 8.50 \\ 8.50 \\ 8.50 \\ 8.50 \\ 8.50 \\ 8.50 \\ 4.00 \\ 4.00 \\ 4.00 \\ 4.00 \\ 4.00 \\ 4.00
\end{bmatrix}
\quad
\mathbf{t} = \begin{bmatrix}
1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0
\end{bmatrix}
\quad
\mathbf{k} = \begin{bmatrix}
1000 \\ 1.50 \\ 1.50 \\ 1.50 \\ 1.50 \\ 1.50 \\ 1.50
\end{bmatrix}
\quad
\mathbf{m} = \begin{bmatrix}
0 \\ 0 \\ 0 \\ 3.5 \\ 3.5 \\ 3.5 \\ 0.1
\end{bmatrix}$$

5 Forward Kinematics via Dynamic Relaxation

This section presents an algorithm for determining the robot equilibrium state (i.e. positions of all of the nodes and internal forces) using dynamic relaxation, given its TR-parameters and an approximate (non-equilibrium) initial position.

5.1 Problem formulation

The goal of the algorithm is to take the current configuration of the structure (TR-Parameters and current member lengths) and to calculate the equilibrium location of the nodes. While the algorithm will find the location of all the nodes, we specifically identify a subset as the "output nodes" $\mathbf{N}_O$ since in most applications we are primarily concerned with positioning specific parts of the structure, such as an "end-effector". In describing TFK we will use the notion from the TR-Parameters (Table 1) and the TFK Notation in Table 5.

Table 5: TFK Notation

Symbol	Dim	Description
$\mathbf{N}$	$3 \times n$	Matrix representing the locations of the nodes. $\mathbf{N} = [\mathbf{N}_B \vert \mathbf{N}_I \vert \mathbf{N}_O]$
$\mathbf{N}_B$		(Fixed) base node(s),
$\mathbf{N}_O$		Output node(s)
$\mathbf{N}_I$		Internal nodes.
$\mathbf{F}_E$	$3 \times n$	External forces on each node (not including gravity)
$\mathbf{M}$	$3 \times m$	Matrix representing the orientation and length of each member. If member k leads from node i to node j, $M_k = N_i - N_j$ where M_k is the kth column of $\mathbf{M}$, and N_i, N_j are the ith and jth columns of $\mathbf{N}$.
μ	$3 \times m$	The orientation of each member expressed as a unit vector. Each column of μ can be obtained by normalizing the corresponding column of $\mathbf{M}$: $\mu_k = \Vert M_k \Vert$.
$\mathbf{f}$	$m \times 1$	Vector containing the axial force of each member.
$\mathbf{R}$	$3 \times n$	Matrix representing the net force on each node.

The general formulation is: Given the TR-Parameters and given the locations of $\mathbf{N}_B$ (the base/fixed nodes), and assuming that $\mathbf{N}_B$ has enough dimensionality to constrain the "base" of the structure, find $\mathbf{N}_I, \mathbf{N}_O$ and $\mathbf{f}$ such that $\mathbf{R} = f(\mathbf{N}, \mathbf{f}) = \mathbf{0}$ (i.e. the structure is at equilibrium with no residual forces at the nodes). In practice, it is sufficient to choose a threshold close to zero which $\mathbf{R}$ may not exceed.

5.2 Solutions Explored

Because the node positions and internal forces are initially unknown and the relationships between them are nonlinear, it is difficult to write a closed form solution. Since we desire an algorithm that can be applied without particular assumptions or placing requirements on the structure (such as symmetry or regularity), the approaches tried focused on numerical as opposed to analytical or algebraic solutions. In addition, because the node positions are unknown and the member lengths are constrained, the algorithm

Table 6: Comparison of Solution Methods

	Kinematic	Generic to all Structures	Considers Forces	Constrains Member Lengths	Allows Elastic Members	Notes
Graph Realization	Yes	Yes	No	Yes	No	
Nonlinear Optimization	Yes	Yes	Yes	Yes	Yes	Nonconvex and large number of free variables.
Analytical methods	Yes	No	Yes	Yes	Yes	Number of variables must be reduced via analysis specific to structure
Force-Density Method	No	Yes	Yes	Yes	Yes	
Iterative Force-Density Method	Yes	Yes	Yes	No	Yes	Member lengths may not be constrained, and large deviations are possible
Dynamic Relaxation (Zhang *et al*)	Yes	Yes	Yes	Optional	Yes*	
Simplified Dynamic Relaxation	Yes	Yes	Yes	Yes	Yes*	As proposed below (section 5.3)

* – *All members elastic*

must generally be of a kinematic type as opposed to a static type as described by Tibert and Pellegrino and allow specification of member lengths. This means that the the numeric (Force-Density) approach of Estrada *et al* is not suited to this problem.

Table 6 provides an overview of several solution methods that were considered. A successful algorithm will be of kinematic type, consider forces, constrain member lengths, and should allow elastic members. This allows suggests that we consider Graph Realization, Nonlinear Optimization, and Dynamic Relaxation methods, which will be discussed in the following sections. Analytical methods (i.e. finding a closed-form solution through algebraic simplification) were not considered because the simplification process relies on specific knowledge of the structure, and the iterative Force-Density methods were not considered because they do not allow member lengths to be specified.

5.2.1 Unyielding Tensegrities and Graph Realization

Setting aside consideration of forces and elasticity, the form-finding problem can also be restated as a problem known as the Graph Realization Problem, Distance Realization Problem, or Sensor Localization Problem: given a weighted graph $G = (V, E, w)$, can this graph be realized in three dimensions by finding node positions such that the distances between nodes equal the (known) edge weights? This graph thus represents an *unyeilding* tensegrity. With an incomplete, connected graph, this problem is known to be NP-hard, and viable heuristics exist only for specific subcases.

Biswas and Ye [1] [2], working with sensor networks, propose a relaxation of the problem into a semi-definite programming formulation, and So [23] [22] suggests its use for structures such as tensegrity frameworks. However, this approach fails as the edge-to-vertex ratio decreases, and it is not suitable for many practical tensegrity structures. So suggests that this relaxation produces a result only when there is a single realization, and that if there are multiple realizations the semi-definite problem becomes non-convex. This means that the solver is much more likely to fail with less-connected graphs, because these graphs are more likely to have multiple valid solutions since there are fewer constraints. For example, there may be internal mirrorings that would not exist if an additional member is added.

5.2.2 Nonlinear Optimization

When generalized as much as possible, the problem is a nonlinear optimization where the parameters are the node locations and the objective is to minimize $\mathbf{R}$, constrained by the known lengths of the stiff members and the force-to-length relationship of the elastic members (springs). However, this most general problem is not well conditioned to be solved with standard nonlinear solvers. First, the solution space is of very high

dimension: for the TenseBot, there are 9 free nodes (with 3 spatial dimensions each) and 30 members that carry force, leading to a 57-dimensional solution space; the objective $\mathbf{R}$ has 36 dimensions. Even satisfying the constraints on member length requires satisfying the Graph Realization Problem.

One approach to deal with the constraints and reduce the number of dimensions in the problem is to represent each of the members as an elastic member, either with very high stiffness or a low stiffness representative of the actual stiffness of that member. Then the force in that member is entirely a function of the distance between the two nodes. In the case of the highly stiff members, because of the high reaction forces generated by deformation it will be difficult for the system to come into equilibrium unless the distance betweens the ends of these members is close to their undeformed length. Thus, it is no longer necessary to constrain the solver externally; the constraints will be satisfied by the solution.

To use nonlinear solvers such as Newton's method, BGFS, Gradient Descent, or Levenberg–Marquardt, $\mathbf{R}$ must be reduced to a single dimension. With the goal of finding $\mathbf{R} = \mathbf{0}$, the obvious way to do this is by taking the norm of the norm of the column vectors of $\mathbf{R}$; the norm of a column vector is the magnitude of the force on that node, the norm of norms is good representation of the overall residual forces and can only equal zero if all the components are zero.

However, this approach is not typically successful. When compressed to a single dimension, the objective function is generally not convex, and may have many local minima. Further, there will likely be more than one point where $\mathbf{R} = \mathbf{0}$. For example, in the case of a structure sitting on a table (represented in the model as a case where the bottom-most nodes are fixed), one could imagine a stable configuration where the structure has fallen through the table and reaches an equilibrium.

In addition, these methods typically use a gradient and look for a solution on

the direction of the gradient (either by moving a small amount as in gradient descent, or by using the gradient to linearize and solve as in NLLS, or a weighted combination of both (LM). However in this case, the local gradient often does not point towards an overall solution, leading the solver to iterate chaotically or enter repeated oscillations without converging to a solution.

5.3 TFK Algorithm (Simplified Dynamic Relaxation)

Algorithm 1 TFK Algorithm

function TFK($TRParams, \mathbf{N}_{init}, \mathbf{F}_E$)
  $\mathbf{N} \leftarrow \mathbf{N}_{init}$
  repeat
    $\mathbf{F}_G \leftarrow FindGravity(\mathbf{N}, \mathbf{m})$
    $\mathbf{F}_{ext} \leftarrow \mathbf{F}_E + \mathbf{F}_G$
    $\mathbf{M} \leftarrow \mathbf{N} * \mathbf{C}^T$
    $\mathbf{d} \leftarrow \sqrt{\mathbf{diag}(\mathbf{M} * \mathbf{M}^T)}$
    for $i = 1 \rightarrow m$ **do**
      $f_i \leftarrow k_i * (d_{0,i} - d_i)$
      if $t_i \neq 1$ & $f_i < 0$ **then**
        $f_i \leftarrow 0$
      end if
    end for
    $\mu \leftarrow \mathbf{M} * \mathbf{diag}(1/\mathbf{d})$
    $\mathbf{F} \leftarrow \mu * \mathbf{diag}(\mathbf{f})$
    $\mathbf{R} \leftarrow \mathbf{F} * \mathbf{C} + \mathbf{F}_E$
    $\mathbf{v} \leftarrow (1 - \varepsilon) * \mathbf{v} + \varepsilon * \mathbf{R}$
    $\mathbf{N} \leftarrow \delta * \mathbf{v}$
  until $\max(\mathbf{R}) < Threshold$
  return $\mathbf{N}$
end function

Rather than searching for a solution directly, an alternative approach based on physical simulation can be used. Generally, this strategy is referred to as *dynamic relaxation*; forces are modeled and nodes are moved until these forces reach an equilibrium. In contrast to other approaches, this method can typically yield solutions given

adequate initial conditions. Because it achieves the goal of determining robot position given it's parameters, we call this algorithm the *TFK Algorithm* (Tensegrity Forward Kinematics).

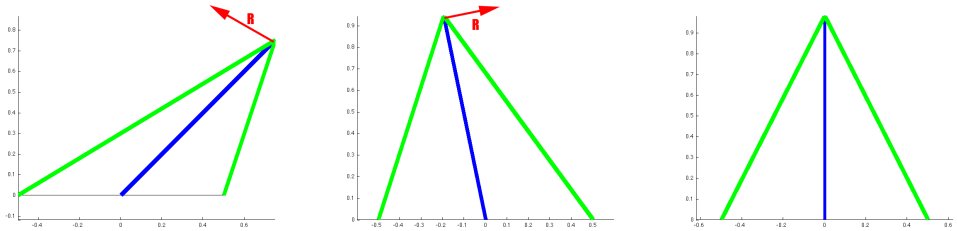
Similar to the nonlinear optimizations, this algorithm assumes that all members contain elasticity and have a (known) undeformed length. Forces are computed in each member and summed at each node and these forces produce accelerations on the nodes. The algorithm (Algorithm 1, below) iterates by adjusting the node locations until the sum of forces on each node reaches zero. This is a state of static equilibrium.

It takes as input the TR-Parameters, an initial guess at the node location ($\mathbf{N}_{init}$), and the external forces ($\mathbf{F}_E$). The algorithm then iterates: gravitational and external forces at each node are calculated, the member matrix $\mathbf{M}$ is populated using the current node positions, and the lengths of each member ($\mathbf{d}$) are computed from $\mathbf{M}$. Next, forces in each member are computed by multiplying their elastic constant by the amount of deflection. Forces in tensile members that are shorter than their rest length are set to zero, representing them entering a slack state. Compressive members however are allowed to carry both tensile and compressive loads.

Once the orientation of each member and the forces in them are known, these forces can be applied to the nodes along with external forces, producing the matrix of residual forces $\mathbf{R}$. This represents the net force on each of the nodes in the structure. These forces produce accelerations at the nodes, changing the (virtual) nodal velocities and thus the nodal positions. The algorithm iterates until nodes move to an equilibrium configuration (*i.e.* where $\mathbf{R} = 0$ or $\mathbf{R} < Threshold$).

A simple single-bar example is presented in Figure 7. The structure is a bar of length 1, symmetrically connected to two tendons. Figure 7a shows an initial condition and the resultant force vector. In 7b, the bar has swung past the equilibrium point and the resultant now pulls it the other way. In 7c it has reached equilibrium. Several more

examples will be shown in Section 5.4.



(a) Initial guess (b) After several iterations (c) Equilibrium state

Figure 7: Single-bar, 2-D TFK example

In lines 16 and 17 of the algorithm, we introduce two tuning parameters, δ and ε, that can be adjusted to produce good convergence behavior. These parameters could be set to represent the member masses and a damping factor, which would result in a physically realistic convergence process. However, this is not necessary if we are only interested in finding an equilibrium solution, since at equilibrium $\mathbf{v} = 0$ and $\mathbf{R} = 0$, and thus δ and ε will not affect $\mathbf{N}$. Since δ and ε are free, to reduce the algorithm run-time we can tune them for a specific structure by using an optimization technique, such as a simplex search (see Table 7).

An advantage of the TFK algorithm is that is can also be used to calculate deformations caused by external forces. Since $\mathbf{N}$ and thus $\mathbf{R}$ include both nodes internal to the structure and nodes at the base and output, external loads can be easily included in line 5.

Dynamic relaxation has been explored by Zhang *et al* [28]. Their method tracks the kinetic energy of the system (initially zero) and looks for situations where kinetic energy decreases from one step to another, and then backtracking to the peak kinetic energy and resetting all velocities to zero. This is because states with maximal kinetic energy would have minimal stored energy and thus be near points of equilibrium.

However, we found the backtracking to be computationally intensive and complex; it was not needed to reach stability provided that sufficient damping was provided to the system elsewhere.

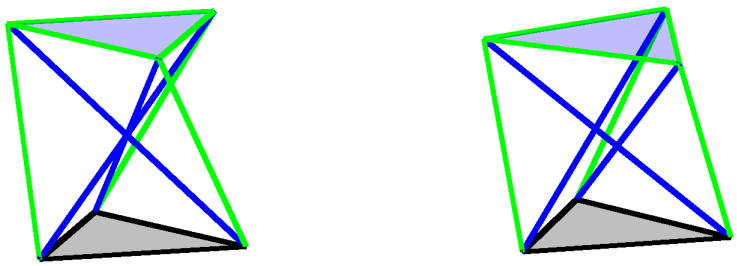
Table 7: Optimization of δ,ε parameters

	Unoptimized			Optimized		
	δ	ε	Iter	δ	ε	Iter
3-Prism	0.01	0.05	644	0.0234	0.3193	91
TenseBot	0.01	0.05	3241	0 .0114	0.0428	925

5.4 TFK Examples

We applied the TFK algorithm to the structures described earlier in Section 4.2. Below we show both the starting configurations and the resulting equilibrium configuration that is calculated.

5.4.1 Three Bar Prism



(a) Initial guess (b) Solved (equilibrium) state

Figure 8: 3-Prism TFK

An initial $\mathbf{N}$ is generated:

$$\mathbf{N} = \begin{bmatrix} -0.5774 & 0.2887 & 0.2887 & 0.5774 & -0.2887 & -0.2887 \\ 0 & 0.5000 & -0.5000 & -0.0000 & -0.5000 & 0.5000 \\ 0 & 0 & 0 & 1.0000 & 1.0000 & 1.0000 \end{bmatrix}$$

This can be seen by inspection to consist of two equilateral triangles, one on the ground plane, and one at a plane in $z = 1$ and rotated 180 degrees. Note that this initial guess does not satisfy $\mathbf{d}_0$, nor does it need to. Thus, it is easy to select an initial $\mathbf{N}$.

This converges (with $\max(\mathbf{R}) < 10^{-5}$) to the following node positions:

$$\mathbf{N} = \begin{bmatrix} -0.5774 & 0.2887 & 0.2887 & 0.4981 & -0.0038 & -0.5019 \\ 0 & 0.5000 & -0.5000 & 0.2920 & -0.5774 & 0.2854 \\ 0 & 0 & 0 & 0.9555 & 0.9555 & 0.9555 \end{bmatrix}$$

In this state the member lengths each match the desired member lengths (i.e tendons of length 1, bars of length 1.468.)

5.4.2 Four Bar Prism

The four bar prism solution is shown in figure 9.

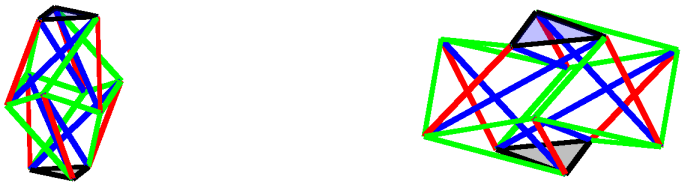
5.4.3 TenseBot Platform

The TenseBot platform is considerably more complex than the regular tensile structure as it has 12 nodes and 30 members; it incorporates both rigid bars, inelastic tendons, and elastic springs. In practice it will also have members with unequal lengths as the tendons are adjustable. Given an adequate initial condition and tuning parameters, this algorithm presented is able to solve the structure in under 1,000 iterations, which takes

(a) Initial guess (b) Solved (equilibrium) state

Figure 9: 4-Prism TFK

under 2 seconds on a modern (2 GHz dual core i5) laptop. See Figure 10.



(a) Initial guess (b) Solved (equilibrium) state

Figure 10: TenseBot TFK

5.5 Convergence and Stability

Because this is an iterative algorithm, it needs a suitable initial guess. The initial conditions ($\mathbf{N}_{init}$) do not need to represent an equilibrium state, but does need to be close enough to a desirable equilibrium state such that the algorithm converges to this solution. Generally, suitable initial states can be determined via inspection and knowledge of the structure. For example, Figure 9 shows the initial conditions of a four bar prism where the bars are oriented vertically, which is easy to generate. The member lengths

of the initial guess do not need to be the same length as the member lengths supplied in the TR-Parameters. The only other requirements of the initial guess is that no two nodes have the same location, and that the nodes are not all coplanar.

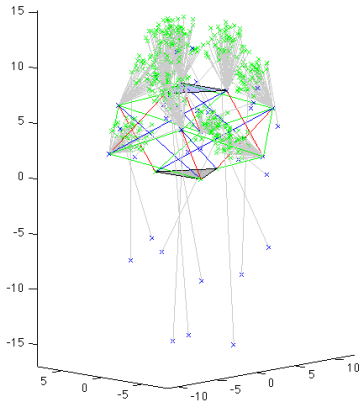


Figure 11: TenseBot: Sensitivity to choice of initial conditions

Figure 11 shows the effect of attempting to solve the TenseBot structure with initial conditions randomly altered. To evaluate the stability of the algorithm with respect to variation in the initial parameters, 50 different initial conditions (shown as green ticks) were chosen by adding random values to a known-good starting location. In 48 of the cases the algorithm converged to the same final equilibrium position. Two cases showed convergence to other locations of either extreme asymmetry or else a degeneration to a location under the base plane. It is should be generally possible to detect these cases and modify the initial conditions to achieve a better solution. In general, once a usable initial condition is found, it can be reliably used for all TFK queries for that structure and thus only needs to be done once during initial set up of the TR-Parameters for the structure.

6 Inverse Kinematics via Jacobians

In rigid chain robotics, the concept of inverse kinematics typically applies to the selection of joint-space parameters based on a desired end-effector position. Thus, the analogous problem for the tensegrity robot is choosing a subset of configuration parameters (such as tendon lengths) to position the robot based on a particular desired set of control parameters. As the output of the TFK algorithm is the node positions and it may not be convenient to specify node positions directly (as they may be unintuitive and geometric constraints may exist), these control parameters will be the output of a function that takes the node positions as an input. This function is structure/task specific and serves to define the state of the end effector based on the node positions of the tensegrity in convenient parameters; it may also contain additional parameters which we would like to control.

For example, in the TenseBot the upper platform consists of three rigidly connected nodes. It is most convenient to consider this plate as single body with 6 degrees of freedom (position and orientation in 3 dimensions); and writing the relationship between the position of the nodes in the plate and these parameters is straightforward. In the TIK algorithm presented below (see Algorithm 2), this function is named $OutputPos$ and the resulting parameters $\mathbf{P}$.

The TIK algorithm thus takes a given goal position (expressed in $\mathbf{P}$) and finds a solution for $\mathbf{d}$ that results in the desired $\mathbf{P}$. The algorithm presented uses the method of inverse Jacobians to move through the configuration space until the desired $\mathbf{P}$ is reached. The Jacobian in this case is the Jacobian reflecting the change in $\mathbf{P}$ due to changes in $\mathbf{d}$. That is, $\mathbf{J} = \mathscr{J}(OutputPos(TFK(\mathbf{d})))$. The inverse of this Jacobian times the position error vector creates a vector that represents the change in $\mathbf{d}$ needed to reduce this error were the system linear. The algorithm takes a small step in this

Algorithm 2 TIK Algorithm

1: **function** TIK($\mathbf{P}_{des}, TRParams, \mathbf{N}_{init}$)
2: $\quad\mathbf{N} \leftarrow TFK(TRParams, \mathbf{d}, \mathbf{N}_{init}$
3: $\quad\mathbf{d} \leftarrow \mathbf{d}_{init}$
4: $\quad\mathbf{P} \leftarrow OutputPos(\mathbf{N})$
5: $\quad$**repeat**
6: $\quad\quad\mathbf{e} \leftarrow \mathbf{P}_{des} - \mathbf{P}$
7: $\quad\quad\mathbf{J} \leftarrow Jacobian(\mathbf{d})$
8: $\quad\quad\Delta\mathbf{d} \leftarrow \mathbf{J}^{-1} * \mathbf{e}$
9: $\quad\quad\Delta\mathbf{d} \leftarrow \varepsilon * (\Delta\mathbf{d})$
10: $\quad\quad\mathbf{d} \leftarrow \mathbf{d} + \Delta\mathbf{d}$
11: $\quad\quad\mathbf{N} \leftarrow TFK(TRParams, \mathbf{d}, \mathbf{N})$
12: $\quad\quad\mathbf{P} \leftarrow OutputPos(\mathbf{N})$
13: $\quad$**until** $\|\mathbf{P}_{des} - \mathbf{P}\| < Threshold$
14: **end function**

direction, and then recomputes the position and Jacobian in order to choose the next step.

In the TIK Algorithm, steps 1 to 3 initialize the variables. Step 5 calculates the current error in position. Step 6 computes a Jacobian at the current position, and step 7 computes a step based on the Jacobian inverse. Step 8 normalizes $\Delta\mathbf{d}$ to a reasonable step size, and the remaining steps update $\mathbf{d}$ and recompute the position and Jacobian. The process is repeated until the error in $\mathbf{P}$ drops under an error threshold.

An effective optimizations is to use Broyden's method [3] to approximate the Jacobian. This is valuable because numerically computing each column of the Jacobian requires a run of TFK, and normally the Jacobian must be calculated on each iteration. These Jacobian computations account for most of the runtime of the unmodified algorithm. For this optimization, we compute $\mathbf{J}$ directly (in step 7) on the first iteration but estimate $\mathbf{J}$ using the approximation $\mathbf{J}_+ = \mathbf{J} + \frac{\Delta\mathbf{P} - \mathbf{J}\Delta\mathbf{d}}{\Delta\mathbf{d}^T\Delta\mathbf{d}}\Delta\mathbf{d}$ on subsequent iterations.

An example of TIK being used to control the bar angle on the one-bar model is shown in Figure 12. In this case, the upper node is movable, and the bar easily computed from the node coordinates. Thus $\mathbf{P} = \theta$ and $OutputPos(\mathbf{N}) = \tan^{-1}(y/x)$ where y and

x are elements of **N**. In the figure, the bar starts are $\theta = 90^\circ$ and is commanded to $\theta = 45^\circ$. This is achieved by adjusting the lengths of the two tendon elements.

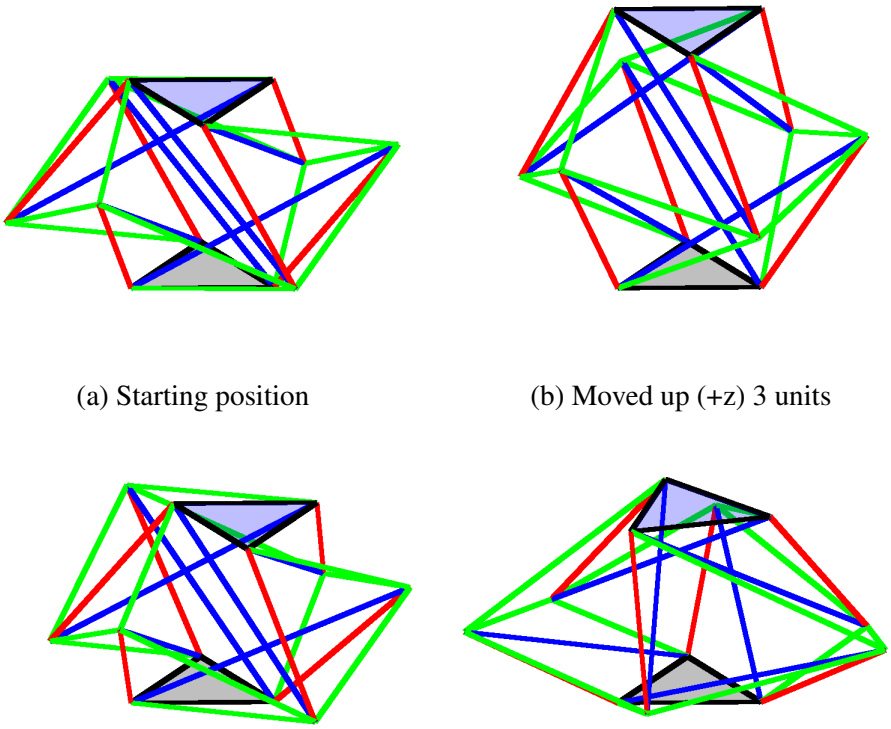
(a) Starting position (b) $\theta = 45^\circ$

Figure 12: TIK used on single-bar system

Because each step is generated through the TFK algorithm, it always represents a stable equilibrium state for the structure. Therefore, if the TIK algorithm is used to plan a trajectory for the robot, it assures that the intermediate points in that trajectory will also be stable.

6.1 TIK on TenseBot

We have also applied the TIK algorithm to the TenseBot model described in section 5.4.3. The "output" parameters are $\mathbf{P} = (x, y, z, \phi, \theta, \psi)$, the position and 3-2-1 Euler angle representation of the output flange. Starting from a case where all tendons are given a length of 8.5 inches, the TFK algorithm puts the flange at $\mathbf{P} = (0, 0, 7.5324, 0, 0, -0.0029)$ (units in inches and radians). Some examples of movements from this initial position are shown in Figure 13.



(a) Starting position

(b) Moved up (+z) 3 units

(c) Moved forward(+x) 3 units

(d) Rotated 45° around z-axis ($+\psi$)

Figure 13: Inverse Kinematics for TenseBot

7 Implementation

The motion control algorithm has been implemented on a small tensegrity structure, named "TenseBot" at NASA-Ames Research Center. TenseBOT implements the analogue of a 6-DOF Stewart platform. It was originally designed and built by a senior project team at the University of Idaho. The structure is a 6-strut tensegrity, with a triangular base and upper platform. Three struts are connected to the base, and three struts are connected to the upper platform. There are 12 adjustable tendons: six saddle tendons which connect upper and lower struts, and six draw tendons which connect struts to the opposite platform. There are are also six passive members, and 6 virtual members comprising the base and upper platforms. In total there are 30 members, and

12 nodes where members intersect.

7.1 Hardware Design

The physical and hardware design was developed by the UIdaho team. The struts are aluminum tubes, and the tendons are hi-modulus Spectra line. The tendon lines run through the struts to spools attached to servo motors mounted to the base and top platform. In this arrangement, tendons' lengths are controlled directly by the servo motors.

The servo motors are Dynamixel AX-12A serially controlled servo motors. They accept commands via a serial bus, and are connected in a chain. The AX-12s can accept both position commands (over a limited range of travel), or operate in a free-run mode where they are given speed commands. To facilitate a larger range of tendon motion, the free-run mode is used, and external 10-turn potentiometers are used to measure position.

A Parallax propeller board is used to control the servo motors. This board reads potentiometer position using 12-bit ADCs, and generates servo controller speed commands using a PI control scheme to move to desired potentiometer values. Power supply for the servo motors as well as supply voltage for the potentiometers is located on the board.

The goal potentiometer values are obtained from control software running on a host computer. The Propeller is connected to the host via USB. The control software implements the kinematics solvers described in the prior sections. The computer also reads back actual potentiometer values and motor velocities from the Propeller.

The control software is implemented as a MATLAB GUIDE GUI program, which handles communication, user input, and calls the kinematics solvers. The hardware and physical design of the robot were unchanged from what was delivered by

the UIdaho team. Both the control software and the embedded Propeller code were rewritten.

7.2 Embedded system

The Propeller is an 8-core micro-controller that can run spin code, a device specific language, as well as native assembly code. Unlike typical micro-controllers with a single processor core that use interrupts to ensure timely response to external events, the Propeller does not have interrupts but instead uses multiple cores that run in parallel. Thus particular cores are given their own tasks, and can communicate with each other via shared memory. Third party drivers to handle communication with the AX-12 as well as the ADCs and USB port are available, simplifying implementation. These drivers typically each use their own core while the main control loop runs on the first core.

The controller uses a separate PI control for each tendon. The current position is read from the potentiometer and the control input is computed based on the current error and the sum of errors. Because the AX-12 does not have a true torque command mode, the control input is given as a speed command; when the AX12 is under load the speed commanded is roughly proportional to the torque actually produced. Since the speed will be limited under low loads, it is not necessary to add the D (derivative) term to the control scheme.

The control inputs are sent to the AX-12s via a call to the AX-12 driver. Potentiometer readings are read from the ADC by a call to the ADC driver. The current position, desired position and speed are each stored in arrays.

Communications between the host computer and the propeller are via a simple, fixed-length message protocol. There are two message types, and communications are

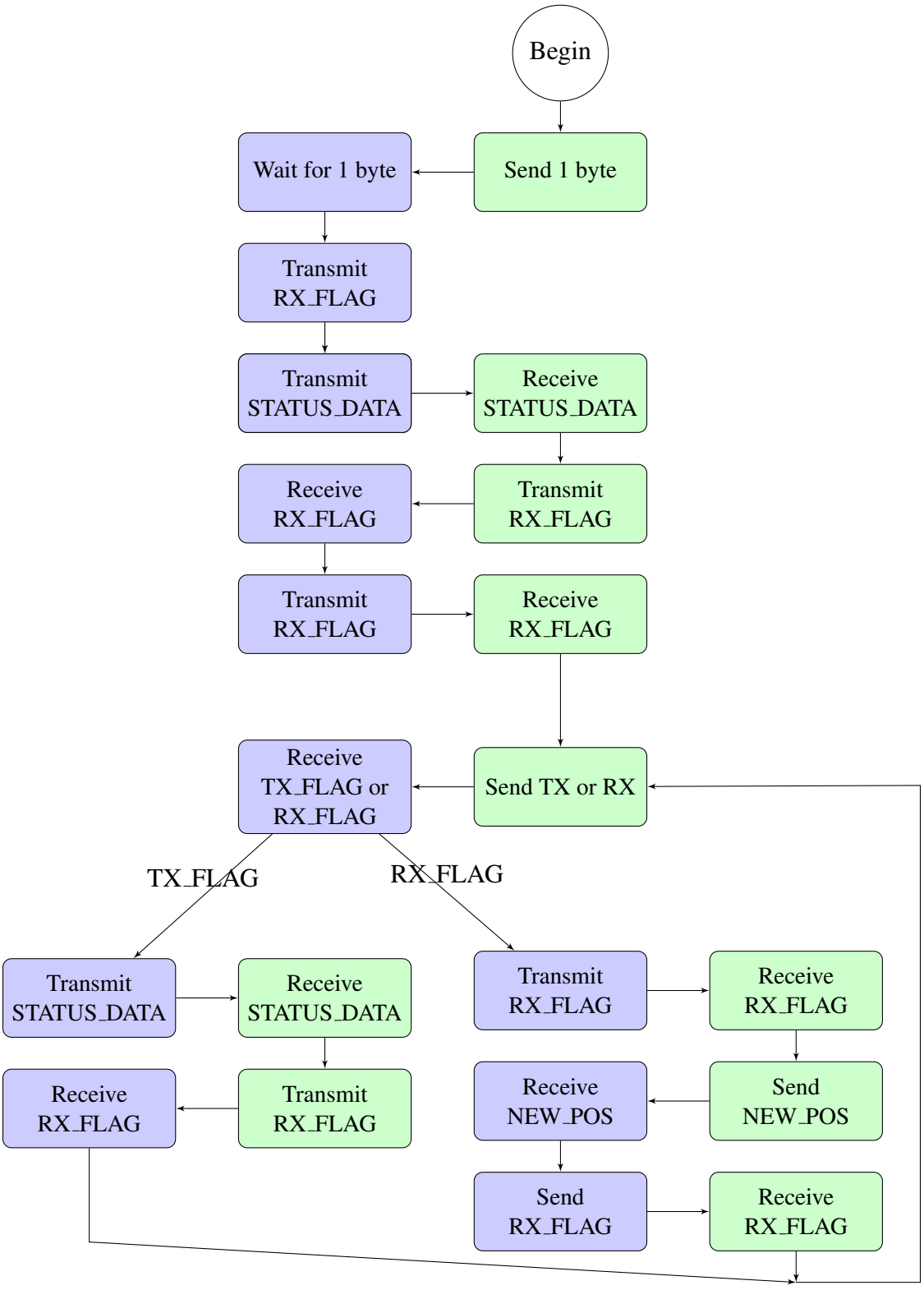


Figure 14: Communications between Host and Propeller

always initiated by the computer. In one message type, the computer asks the propeller for the current state of the robot; in this case the propeller's communications code reads the state arrays and sends them to the host. In the other message type, the computer sends a new position to the controller. In either case each of the arrays are serialized and sent as unsigned 16-bit integers.

This communications scheme preserves backwards compatibility with earlier versions of the controller software. The communications driver runs on its own core, so communications are carried out asynchronously with the control loop and the communications code cannot block the control loop code (except during the brief period of time needed to write to memory).

7.3 High-Level Control

The host computer code is a MATLAB GUI application (using the GUIDE interface). It communicates with the Propeller using the communications described earlier. The control computer uses the forward and inverse kinematics schemes described in sections 5 and 6.

The main program structure is presented in Figure 15. At initialization, the drivers are started and the goal positions are set to the current position as read from the potentiometer. After initialization, the main control loop is started.

The forward kinematics solver is implemented in the function **fkin**, which takes the robot TR-Parameters and an initial guess at node positions at input, and returns an equilibrium position with the given member lengths. This solver is generalized and takes all system-specific parameters from the configuration parameter structure.

The inverse kinematics solver is implemented in the function **ikin**; this solver uses a generalized Jacobian inverse to attempt to select member lengths which produce

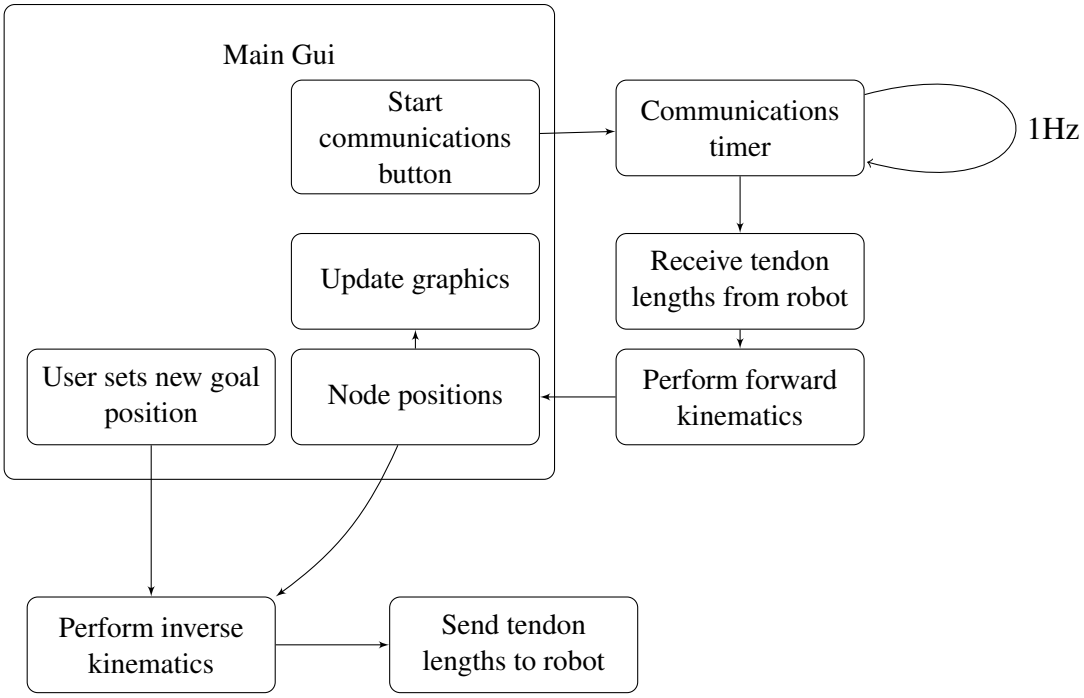


Figure 15: Program Structure

a desired set of output parameters. Again, this is a generalized solver; the configuration parameter structure specifies the system specific parameters, including a function that produces and specifies which of the members should be actuated. This calls the forward kinematics solver **fkin**. This implementation calculates the Jacobian numerically by finite differences for the initial position, and uses Broyden's method to approximate the Jacobian.

The GUI program initializes the configuration structures with robot-specific settings, handles communication with the Propeller via USB, and calls the solvers as needed. It uses the forward kinematics solver to determine the current robot position based on telemetry from the robot (i.e. tendon lengths), and the inverse kinematics solver to determine tendon lengths based on a goal position specified by user input.

The communications code receives data from the Propeller (position, desired position and commanded velocity), and sends data to the Propeller (new desired posi-

tion). The communications protocol is a flag-passing protocol of minimal complexity. Messages are of fixed length and format. The protocol is presented in Figure 14. All communications are initiated by the host. A timer calls the "receive data" call at a predefined rate; this gets the current status of the robot. Data sends occur in response to user activation of the "send tendon lengths" command.

7.4 Calibration

For the kinematics model to accurately represent the physical system, it is important that the physical values used by the model be accurately represented. These representations are the basis for the calibration of the system.

First, lengths of fixed elements (such as the struts and length of the base) must be accurately measured. The model also includes these elements elasticity, but as long as these elements have a high stiffness it does not have to be accurately measured. For spring elements, their zero-force length and spring constant must be measured. Because the spring may pull its own coils into contact with each other, there may be a nonlinearity at the minimum spring length. Thus, the spring constant should be measured by taking multiple readings in the region of linear, elastic deformation of the spring.

For the tendons, the goal is to develop a relationship between the string length and potentiometer reading. Because the strings are wound onto a spool and the string cross section is small in relation to the spool diameter, this relationship will be close to linear. Thus for calibration for the tendons it suffices to measure a slope and an intercept. Although two points would be sufficient, measuring multiple points and applying a least squares fit to a linear function will reduce error.

7.5 Open issues

Generally, the robot is not yet a fully reliable system. This is due partly to design problems with the system as a whole and partly to unresolved debugging issues. Many of the issues relating to the robot design are described in Section 8. There are also some bugs that interfere with reliable operation of the system. First, there is an intermittent behavior where a motor will begin running and not stop at the desired position. It is still unclear whether this is due to a communications issue between the motor and the Propeller, or a failure within the control loop.

Because of the motor runaways (which can lead to the spools becoming unwound) we have not yet been able to perform a suitable calibration to be able to proceed to further physical testing of the system and the control algorithms. Further steps in the development and testing of the system will be addressed in Section 8.3.

8 Future Work

There are many opportunities for future work and research, both in the area of the kinematics algorithms and the TenseBot. The kinematic algorithms are rather rudimentary, and there are several improvements that could improve their reliability and performance. In addition, there are many aspects of the TenseBot design that could be changed to improve its performance.

8.1 TenseBot Improvements

During the implementation of the control algorithms on the TenseBot, several areas of potential improvement became obvious. These relate to both the mechanical design and motor and sensor selection and could result in improved performance and reliability, as

well as making it possible to investigate alternative control schemes (such as internal force-based control).

A major issue with the current implementation is the selection of motors. The servo motors currently used have internal controllers and take commands via a serial bus. This makes interfacing with them from the micro-controller easy. However the servos have several limitations that make them inappropriate for this project. Ideally, these servos would be used in a position control mode; they have an internal potentiometer and controller well-suited to position control. This is the designed use for these servos. However, the internal potentiometer has a limited range of 300 degrees, and the servo may only be positioned within that range using the position control mode. This range of travel would only allow for approximately 1" of tendon adjustment. To achieve a longer range of travel in this case, the only option is to increase the spool diameter which would cause packaging difficulties (and reduce the amount of force that could be applied to the tendon by the motor).

The servos do however have a free-run mode. In this mode, rather than taking position commands, the servos take speed and direction commands. This mode is generally intended for use in driving wheels on a mobile robot or other situations where the servo would be lightly loaded and exact speed control might not be necessary. This free run mode is what is presently used in the robot, and external 10-turn potentiometers are used to measure the tendon spool positions. The position control loop is thus implemented in software on the Propeller and is subject to the latencies in the ADC, the Propeller itself, and the communication over the serial bus. This limits the performance of the controller, and also makes the position control subject to bugs in the Propeller code. For example, if the Propeller hangs up while an axis is moving, there is nothing to ultimately stop its motion without user intervention.

The servos also do not offer very good performance, especially in free-run

mode. They are quite slow, especially when pulling against load. Even while unloaded, the gear train has a significant amount of back-torque and backlash that introduces non-linearity. There is also a considerable amount of holding torque while the motor is unenergized (which is not necessarily a bad thing). In addition, all power for the servos currently runs through the Propeller board and its power supply; this is a 9V, 6A power supply that feeds a 5V regulator. Each AX-12A can pull up to 900mA, meaning that it is possible for the power supply to become overloaded if many axes are being driven simultaneously.

Some potential solutions to the problem would be to implement a bi-level control scheme where the position mode is used where possible, and the free-run mode is used to move the motor through the internal potentiometer deadband (or position it within the deadband) when necessary. It may also be possible to open up the servos and replace the internal potentiometers with a connection to the external potentiometers (although this may lead to problems of insufficient resolution).

A more extensive solution, and one that would be better suited to some of the future goals of the project, would be to replace the current servo motors with a small DC motor and gear head. These would need to have suitable motor control boards, ideally ones that use a current controller to give linear torque control for the motors. This could be used for simple position control in concert with the current inverse kinematics algorithm; other affiliated researchers are investigating other control schemes that rely on tendon force control and a DC motor under torque/current control would be ideal for this, as the motor torque would be well correlated to string tension.

A second issue is that each of the axes need to have potentiometers to measure the spool position, and there is a large amount of wiring needed to connect to the potentiometers to the ADC and power supply back on the controller board. In addition, the potentiometer mounting has been an ongoing issue; the original team used tape to

mount the potentiometer bodies which often failed or moved. These potentiometers are designed to be panel mounted but there is not enough room on the base to fit the potentiometers in a panel. Smaller 10-turn potentiometers are not available. Glue has been used to fix them in place, however, the glued connection also tends to come loose. The solution here would probably be to build a new, larger baseplate and use a more robust panel-mounting for the potentiometers.

An additional improvement to the mechanical design would be relocating some or all of the motors to the lower base. This should be possible because all of the tendons either connect directly to the lower base, or have at least one end that connects to a strut that connects to the lower base. The advantage to this would be that the upper, movable platform would have reduced mass.

8.2 Algorithm Improvements

The current forward kinematics algorithm uses dynamic relaxation, iteratively simulating a physical model of the robot as it moves towards an equilibrium position. There are several issues with this approach. First, it can take a large number of iterations to find an equilibrium point; this is because the size of the steps needs to be kept small to reduce integration error, and the damping factor needs to be kept high to reduce overshoot. This leads to slow convergance. Zhang, Maurin, and Motro [28] propose tracking the kinetic energy of the system and looking for increases in kinetic energy, and then backtracking until a minimum is found and resetting the velocities to zero at this point. This approach may serve to reduce the number of iterations needed to reach a solution but it also significantly increases the computational complexity, especially in the backtracking phase (which requires a line-search for a minimum). However, it would allow for a reduction in the damping factor and an increase in the step-size factor.

Currently, no collision checking is performed during the solution process, allowing members to pass through one another. Implementing some form of collision checking would prevent situations such as where the model falls down through the floor. However, it would add considerable complexity to the process. For example when two bars collide, the motion of the nodes involved must be constrained somehow. This is complicated because there will likely be multiple non-fixed nodes. A realistic model would need to simulate contact forces and apply sufficient contact force to make sure that the nodes do not intersect. Alternatively, hard constraints could be imposed, but it would be important to ensure that this does not add energy to the system or prevent the relaxation from proceeding towards an equilibrium entirely.

There are also other algorithmic approaches to the form-finding problem some of which may ultimately be better suited to the needs of our problem. In particular, the Force-Density Method appears to have a great deal of potential because it linearizes some aspects of the force/node position relationship, allowing linear solvers to be used. The Force-Density algorithms proposed so far do not guarantee member lengths [26] [11] [27], but it is possible that this approach may be structured a different way that would provide better control over the member lengths in the final solution. This approach may also prove viable for inverse kinematics, wherein the solver is used to iterate toward a goal position and the tendon lengths needed are thus found (rather than find the position associated with given tendon lengths). Tran and Lee [27] report very low numbers of iterations (often ten to twelve) needed to achieve a solution, which indicates that other algorithms using this approach might be developed with similar performance.

The TIK algorithm is also a very simple, generic approach. As long as a forward kinematics algorithm is available, it is possible to consider many approaches to inverse kinematics taken from classical robotics and path planning. Methods such as genetic

algorithms or machine learning algorithms might be possible; the high dimensionality of the problem gives a large null-space of alternative solutions to consider. Caching of known solutions and selecting a nearby known solution as a start point is another method that may reduce the amount of iterations needed.

8.3 Physical Testing

A primary goal of implementing the control algorithms on the TenseBot is to perform physical tests that could validate the algorithms. So far, the reliability and calibration issues have prevented this testing from occurring. For the forward kinematics algorithm, validation testing would consist of measuring the position of the robot nodes while the robot is in a variety of configurations, and comparing those positions to what is predicted by the TFK algorithm. Inverse kinematics algorithms can be tested by comparing the desired versus achieved positions.

The key issue preventing further testing of the system is the reliability issue. This stems largely from the architecture, where the position-control loop is closed on the microcontroller. Therefor, additional troubleshooting is needed to make this system reliable. Once the system is stable and the servos can move to commanded positions with high reliability, the next step would be to do an accurate calibration.

Calibration consists of measuring static member lengths, measuring the spring constant and undeformed length of the springs, and then finding the correlation between potentiometer position and tendon length for each of the tendons. The spring constant measurement can be done by attaching a variety of weights and measuring the spring deflection. To get an accurate measurement this should probably be done in a sliding vertical fixture.

To perform the potentiometer calibration, the pot values and tendon lengths

should be measured with a caliper at multiple positions. The SixBarTest firmware can be used for this. The accuracy needed can be inferred from the Jacobian.

To test the accuracy of the algorithms, we must measure the position of the upper platform relative to the lower platform. This is difficult due to the large number of degrees of freedom present in the system. One method to measure the position would be to build a rigid framework that mounts to the base of the robot and extends upward and around the robot and rigidly locates three base points. Length measurements can then be made from points on the upper framework to triangulate points on the robot surface. With three points on the robot surface, the platforms position and orientation can be measured. Making measurements at a number of locations could serve to characterize the accuracy of the algorithms.

9 Conclusions

The use of tensegrity is an exciting concept for the field of robotics, especially in the context of mobile and biomimetic robots. However, adoption and use is limited by the tools and techniques needed to analyze and control them, as well as methods to describe their design and state. This research has sought to simplify and clarify a set of parameters suitable for describing tensegrity states, and some some algorithms suited for computation of static robot states. In addition to basic positional control of tensegrity robots, these algorithms have several potential uses in the modeling and design of robots that use more advanced control methods. For example these algorithms could be used to calculate particular positions and internal forces in the extents of a robot that uses rhythmic motion excited by Central Pattern Generators. In addition, the concepts presented in this work provide a framework to develop new algorithms. Because the TR-Parameters provide a means to precisely communicate the conceptual design of a

tensegrity structure, algorithms using these parameters can be easily exchanged and compared. The TR-Parameters also allow for easy, unambiguous communication of tensegrity structure parameters.

References

[1] P. Biswas and Y. Ye. Semidefinite programming for ad hoc wireless sensor network localization. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, pages 46–54. ACM, 2004.

[2] Pratik Biswas. *Semidefinite Programming Approaches to Distance Geometry Problems*. PhD thesis, Stanford University, 2007.

[3] C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 19(92):pp. 577–593, 1965.

[4] CR Calladine. Buckminster fuller's "tensegrity" structures and clerk maxwell's rules for the construction of stiff frames. *International Journal of Solids and Structures*, 14(2):161–172, 1978.

[5] R. Connelly and M. Terrell. Globally rigid symmetric tensegrities. *Structural Topology 1995 núm 21*, 1995.

[6] J. Denavit. A kinematic notation for lower-pair mechanisms based on matrices. *Trans. of the ASME. Journal of Applied Mechanics*, 22:215–221, 1955.

[7] G Gomez Estrada, H J Bungartz, and C Mohrdieck. Numerical form-finding of tensegrity structures. *International Journal of Solids and Structures*, 43(22-23):6855–6868, November 2006.

[8] E. Fest, K. Shea, B. Domer, and I.F.C. Smith. Adjustable tensegrity structures. *Journal of Structural Engineering*, 129(4):515–526, 2003.

[9] R. B. Fuller. Tensile-integrity structures, November 13 1962. US Patent 3,063,521.

[10] D.E. Ingber et al. Cellular tensegrity: defining new rules of biological design that govern the cytoskeleton. *Journal of Cell Science*, 104:613–613, 1993.

[11] S.H. Juan and J.M. Mirats Tur. Tensegrity frameworks: Static analysis review. *Mechanism and Machine Theory*, 43(7):859–881, 2008.

[12] N Kanchanasaratool and D Williamson. Motion control of a tensegrity platform. *Communications in Information and Systems*, 2(3):299–324, 2002.

[13] K. Linkwitz. Formfinding by the "direct approach" and pertinent strategies for the conceptual design of prestressed and hanging structures. *International Journal of Space Structures*, 14(2):73–87, 1999.

[14] Marc Mohr, Chris A; Aresenault. Kinematic analysis of a translational 3-dof tensegrity mechanism. In *Transactions of the Canadian Society for Mechanial Engineering*, volume 35, pages 573–584, 2011.

[15] R. Motro. Tensegrity systems: the state of the art. *International Journal of Space Structures*, 7(2):75–83, 1992.

[16] Rene Motro. *Tensegrity: Structural Systems for the Future*. Kogan Page Science, 2003.

[17] S Pellegrino. Matrix Analyis of Statically and Kinematically Indeterminate Frameworks. *International Journal of Solids and Structures*, 22(4):409–428, 1986.

[18] R.E. Skelton, R. Adhikari, J.P. Pinaud, W. Chan, and JW Helton. An introduction to the mechanics of tensegrity structures. In *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, volume 5, pages 4254–4259. IEEE, 2001.

[19] R.E. Skelton and M.C. De Oliveira. *Tensegrity systems*. Springer, 2009.

[20] K.D. Snelson. Continuous tension, discontinuous compression structures, February16 1965. US Patent 3,169,611.

[21] A.M.C. So and Y. Ye. Theory of semidefinite programming for sensor network localization. *Mathematical Programming*, 109(2):367–384, 2007.

[22] Anthony Man-Cho So. *A Semidefinite Programming Approach to the Graph Realization Problem: Theory, Applications, and Extensions*. PhD thesis, Stanford University, June 2007.

[23] Anthony Man-Cho So and Yinyu Ye. A Semidefinite Programming Approach to Tensegrity Theory and Realizability of Graphs. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '06, pages 766–775, New York, NY, USA, October 2006. ACM.

[24] M. L. Stephen. The tensegrity-truss as a model for spine mechanics: Biotensegrity. *Journal of Mechanics in Medicine and Biology*, 2(03n04):375–388, 2002.

[25] Cornel Sultan, Martin Corless, and Robert E Skelton. Tensegrity Flight Simulator. *Journal of Guidance, Control, and Dynamics*, 23(6):1055–1064, June 2010.

[26] AG Tibert and S. Pellegrino. Review of form-finding methods for tensegrity structures. *International Journal of Space Structures*, 18(4):209–223, 2003.

[27] Hoang Chi Tran and Jaehong Lee. Advanced form-finding of tensegrity structures. *Computers and Structures*, 88(3-4):237–246, February 2010.

[28] Li Zhang, Bernard Maurin, and Rene Motro. Form-Finding of Nonregular Tensegrity Systems. pages 1–7, August 2006.