

Copyright © 1992, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

FINGER-LIKE BIOMECHANICAL ROBOTS

by

**D. Curtis Deno, Richard M. Murray, Kristofer S.J. Pister,
and S. Shankar Sastry**

Memorandum No. UCB/ERL M89/16

15 February 1992

ELECTRONICS RESEARCH LABORATORY

**College of Engineering
University of California, Berkeley
94720**

FINGER-LIKE BIOMECHANICAL ROBOTS

by

D. Curtis Deno, Richard M. Murray, Kristofer S.J. Pister,
and S. Shankar Sastry

Memorandum No. UCB/ERL M89/16

15 February 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

FINGER-LIKE BIOMECHANICAL ROBOTS

by

D. Curtis Deno, Richard M. Murray, Kristofer S.J. Pister,
and S. Shankar Sastry

Memorandum No. UCB/ERL M92/12

15 February 1992

Copyright © 1992

Abstract

We present a technique to analyze the forces and dynamics of a class of mechanical systems, called finger-like systems, which may be viewed as an extension of simple robots to include networks for force/displacement generation and transmission. Finger-like mechanical systems can be described using a graph-theoretic approach to force and displacement transmission. Branches consist of actuators, cables, springs, and other building blocks. Upon specification of the connectivity graph and branch behaviors, a symbolic mathematics program can generate the affine maps from actuator control variables to mechanical system torques and forces. This process systematizes and simplifies the determination of biological and robotic mechanical dynamics.

Contents

1	Introduction	1
2	Preliminaries	3
2.1	Lagrangian treatment example	3
2.2	Graph theory	5
3	Finger-like Robot Systems	9
3.1	System description and assumptions	9
3.2	Constructing the actuator graph	11
3.3	Main results	14
3.4	Remarks	15
4	Examples	18
4.1	Finger-like treatment of previous example	18
4.2	Biomechanical example of nonreciprocity	21
4.3	Human finger model, analysis	22
4.4	Human finger model, control	27
5	Conclusion	31
A	Undirectable Loops	32
B	Tableau analysis with Mathematica	35
B.1	Tableau.m	35
B.2	Fingerlike.m	36
B.3	Program listings	36
C	Mathematica Listings	43
C.1	example1.m	43
C.2	example2.m	44
C.3	example3.m	45
C.4	buchner.m	46

List of Figures

1	Simple 1 joint finger-like system example	4
2	Cojoined and contrajoined branches at an actuator node . . .	12
3	Branch and node assignments for Figure 1	18
4	Nonreciprocal biomechanical system.	22
5	Anatomical based diagram of Buchner finger model	23
6	Network diagram of Buchner finger model	25
7	Sequence of finger configurations	28
8	Corresponding joint angles and torques	29
9	Minimum norm actuator force trajectories	30
10	Simple undirectable loop example.	32
11	Transformation of an undirectable loop into a finger-like system	34

List of Tables

1	Notation and symbols for actuator networks	6
2	Mathematica listing for solution to 1-joint robot example. . .	21

1 Introduction

In this paper, a class of mechanical systems called *finger-like systems* is introduced for which a graph-based method for manipulator analysis and design is given. *Finger-like systems* may be viewed either as an extension of simple robots (to include force and displacement generation and transmission networks) or as a special case of non-rigid body robots (where the actuator related cables and springs permit useful simplifications). Motivation for the study of *finger-like systems* comes from biomechanical models of fingers where muscle actuators are separated from the fingers by low mass and low friction tendons [12, 2]. From the perspective of robotics, *finger-like systems* are useful models of manipulators with cable-driven joints. An important theme of this work is the generation of tools which aid in both the analysis and design of robot systems.

Graph theory has numerous engineering applications in systems which are modeled as a network of interacting elements. The most common applications are in lumped parameter models of mechanical systems and electrical circuit. Some of the best known graph-based system analysis tools include bond graphs [8], linear graphs [11], and signal flow graphs [10]. The approach of this paper resembles linear graphs, such as that used by Durfee *et al.* [6], but exploits the structure of *finger-like systems* to obtain the nonlinear dynamics of the underlying mechanical system. The kinematic analysis of tendon driven manipulators by Tsai and Lee [13] is also a general, graph-based method but is restricted to systems in which there is no tendon stretch or splitting and where circular pulleys are present at each rotary joint. In this paper a formalism called tableau analysis [5] is used to obtain and solve systems of simultaneous equations relating the actuator system's branch and node variables.

The related papers of Becker *et al.* [1] and Buchner *et al.* [2] analyze the kinematics of a human finger but do not offer a general algorithm suited to automation. A larger body of work can be found on specific tendon-actuated robot hands (e.g. Jacobson, McCarthy, and Salisbury). The work of Jacobson *et al.* on the MIT/Utah hand [7] contains a summary of past efforts as well as new contributions.

This paper provides a general, systematic formulation of dynamics and leads to insights on the design and analysis of actuator networks. We begin in Section 2 with an example and short review of robot dynamics followed by essentials of graph theory. Section 3 presents the main results of the paper: a definition of *finger-like systems*, an algorithm for the solution of

their dynamics, and a proof of this algorithm. Finally, in Section 4 we apply this technique to several examples, including a simplified model of the human finger. Final remarks are given in Section 5.

2 Preliminaries

Before proceeding directly with a formal presentation of finger-like systems, a selective review of mechanical system dynamics and graph theory is presented. In addition to fixing notation, this permits the main results to be presented succinctly in the subsequent section.

2.1 Lagrangian treatment example

The equations of an open-chain rigid-link robot manipulator can be derived using Lagrange's equations. Given the kinetic energy K and potential energy V as a function of the robot's configuration θ and its time derivative $\dot{\theta}$, we define the Lagrangian $L(\theta, \dot{\theta}) = K(\theta, \dot{\theta}) - V(\theta)$. The dynamics of the system obey Lagrange's equations,

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{\theta}_i} \right) - \frac{\partial L}{\partial \theta_i} = \tau_i,$$

where τ_i represents the actuator forces applied conjugate to the direction of motion $\dot{\theta}_i$. This is frequently written in matrix notation as

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + N(\theta, \dot{\theta}) = \tau, \quad (1)$$

where $M(\theta)\ddot{\theta}$ are the inertial forces, $C(\theta, \dot{\theta})\dot{\theta}$ represents the Coriolis and centrifugal force terms, and $N(\theta, \dot{\theta})$ are the nonlinear frictional forces as well as $(D_\theta V)^T(\theta)$ (the partial derivatives of potential energy V with respect to θ).

For biomechanical robots in particular, cables or tendons may be connected to the underlying mechanical system in a complex ways as they bend around joints, bearing surfaces, and travel through tendon sheaths. To introduce our methods which treat this general class of systems, consider the 1-joint example of Figure 1. The actuator network is composed of two tendons which are guided around joints and bones to their attachment points. The analysis in this section employs a classical Lagrangian mechanics approach and the Jacobian to derive the influence of cable tensions on the robot's dynamics. In Section 4 we shall reanalyze this robot using the graph-based approach described for finger-like systems. While this type of actuator network is admittedly simple, it illustrates some important ideas in formulating equations of motion using a Lagrangian technique. It is also a useful starting point of models of mechanical hands which use tendon drives.

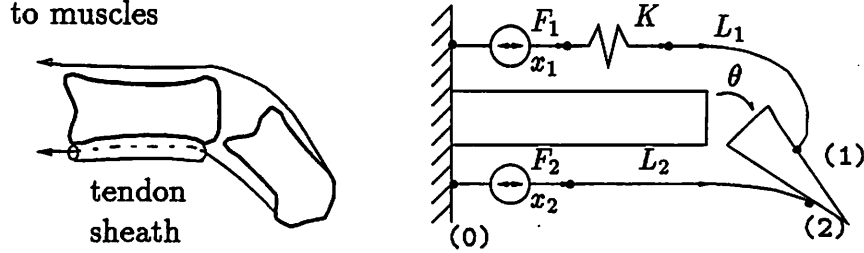


Figure 1: Simple 1 joint mechanical system with configuration parameterized by angle θ . There are two actuators modeled as force generators, F_1 and F_2 .

The actuator inputs are the two force generators with forces F_1 and F_2 and corresponding branch displacements x_1 and x_2 . Force generator 1 is anchored to reference node 0, denoted (0). Cable 1 consists of spring K of zero rest length and inextensible cable of length L_1 and acts on the robot's mechanical structure at node (1). The length of a hypothetical string which travels the actuator cable's path from (0) to (1) is written as $g_1(\theta) + L_1$. From Figure 1 it is apparent that $g_1(\theta)$ increases as angle θ increases.

Similarly, the second actuator network consists of a force generator anchored at reference node (0) in series with cable 2 which has length L_2 and acts at mechanical structure node (2). As before, $g_2(\theta) + L_2$ denotes the length of a string along the actuator network's path from (0) to (2). In this case, $g_2(\theta)$ decreases as angle θ increases. We model the travel along the two actuator network pathways of Figure 1 with simple polynomial models as employed by [2]: $g_1(\theta) = R_1\theta$ and $g_2(\theta) = -(R_2 + R_2'\theta)\theta$.

A set of generalized coordinates for this system is (θ, x_1) which captures both the mechanical system's configuration θ and the actuator spring displacement via both θ and x_1 . The Lagrangian L for this system is

$$L = \frac{1}{2}\dot{\theta}^T M(\theta)\dot{\theta} - \left[V_g(\theta) + \frac{1}{2}(g_1(\theta) - x_1)^T K(g_1(\theta) - x_1) \right], \quad (2)$$

where V_g is the potential energy due to gravity. A more general problem would associate K with a symmetric, positive semidefinite stiffness matrix.

Following (1), Lagrange's equations are

$$\begin{aligned} M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + (D_\theta V_g)^T(\theta) + (D_\theta g_1)^T K(g_1(\theta) - x_1) &= \tau \\ -K(g_1(\theta) - x_1) &= -F_1, \end{aligned} \quad (3)$$

where τ is the generalized force conjugate to velocity $\dot{\theta}$ applied by F_2 (F_1 's action was captured via spring extension $g_1(\theta) - x_1$). The negative sign associated with F_1 is a result of our convention that tension correspond to positive force. Equation (3) is a mixed algebraic-differential equation.

Using the principle of virtual work, we relate the joint torque to tendon force F_2 as $\tau = (D_\theta g_2)^T(-F_2)$. Again, the negative sign results from our convention that tensile forces be positive. Combining this with (3) and the definitions of g_1 and g_2 yields

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + (D_\theta V_g)^T(\theta) = \begin{pmatrix} -R_1 & (R_2 + 2R'_2\theta) \end{pmatrix} \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} \quad (4)$$

which relates applied actuator forces to the robot's dynamics. Note that in this example we were able to calculate the joint torque due to pulling on the tendons without actually calculating the forces over the complex bearing surfaces. This is one of the primary advantages of using generalized coordinates.

2.2 Graph theory



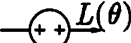
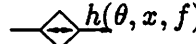
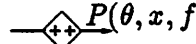
In this section we discuss the application of graph theory to mechanical systems. See [3] for a mathematically precise description of graph theory and [5] for a description of many of the tools used here, presented in the context of circuit theory.

Actuator networks as graphs

Implicit in network models of mechanical systems is a graph structure composed of nodes or vertices and branches or edges. *Nodes* represent points of confluence of *branches* (actuator elements). Associated with each branch is a displacement x (the distance across the nodes delimiting the branch such that extension corresponds to $x > 0$) and force f (such that a tensile force corresponds to $f > 0$). The graph structure is directed since we must assign reference directions for summing branch displacements and forces. The resulting directed graph, or *digraph*, is represented by the notation $\mathcal{G} = (\mathcal{N}, \mathcal{B})$, where \mathcal{N} is a set of nodes and \mathcal{B} is a set of branches between pairs of nodes. Table 1 summarizes our notation.

There is an important distinction between the models we have chosen for mechanical systems and those of electrical systems. In electrical circuit theory, the branch direction is chosen arbitrarily and used as a reference for

Table 1: Summary of notation and symbols used in this paper to represent actuator networks as a graph.

Description	Notation
attachment point, node	(n)
mechanical element, branch	b_i
branch displacement	x_i
branch force	f_i
node displacement w.r.t. ref. node	e_n
spring (zero rest length), $f = Kx$	
indep. force generator, $f_i = F(\theta)$	
indep. displ. generator, $x_i = L(\theta)$	
controlled force generator, $f_i = h(\theta, x, f)$	
controlled displacement generator, $x_i = g(\theta, x, f)$	
undirected actuator graph	$\mathcal{G}^o = (\mathcal{N}^o, \mathcal{B}^o)$
directed actuator graph	$\mathcal{G} = (\mathcal{N}, \mathcal{B})$
augmented graph (w mech. sys.)	$\mathcal{G}^* = (\mathcal{N}^*, \mathcal{B}^*)$

specification of the branch variables (voltage and current). In our actuation network, extension and tension are intrinsic quantities and independent of branch direction. The direction of a branch is used when summing displacements and forces to obtain the net displacement or force at a node. This difference can lead to false intuition if force is directly associated with current and displacement with voltage.

Given a system's digraph, we construct an *incidence matrix* which models the interconnectivity of \mathcal{G} . Given n nodes and b branches, we define the $n \times b$ matrix $A_a = (a_{ij})$ by

$$a_{ij} = \begin{cases} +1 & \text{if branch } j \text{ enters node } i \\ -1 & \text{if branch } j \text{ leaves node } i \\ 0 & \text{otherwise.} \end{cases}$$

We define the $(n-1) \times b$ *reduced incidence matrix* A as the matrix obtained by deleting from A_a the row corresponding to the reference node, which may be chosen arbitrarily.

An important concept in graph theory and our development of finger-like systems is the tree of a graph. A *tree* of a connected graph \mathcal{G} is a

subgraph which satisfies three fundamental properties: (1) it is connected, (2) it contains all of the nodes of \mathcal{G} , and (3) it has no loops. In general, a graph has many trees. There is a unique path along a tree between any pair of nodes.

Given a connected digraph \mathcal{G} and tree \mathcal{T} , the branches of \mathcal{G} may be partitioned into two disjoint sets: those which belong to \mathcal{T} (called tree branches or *twigs*) and those which do not belong to \mathcal{T} called *links* or chords. Since digraph \mathcal{G} has b branches and n nodes, there are $n - 1$ twigs (easily shown by induction) and $\ell = b - (n - 1)$ links. Every link of \mathcal{T} and the unique path on the tree between its two nodes constitute a unique loop called the *fundamental loop* associated with the link. There are precisely ℓ fundamental loops.

The *vertex law* condition that the net force acting at any node must be zero is given by $Af = 0$. This is the analog of Kirchhoff's Current Law (KCL). Furthermore, the net displacement around any loop must also be zero, analogous to Kirchhoff's Voltage Law (KVL). Although this is written most directly using a loop matrix (e.g. $Bx = 0$), for the purposes of the incidence matrix based tableau equations discussed below, the *path law* KVL is expressed as $x = A^T e$ [5].¹

Linear algebraic formulation

In general, a model of a branch's displacement or force can include branch displacements or forces elsewhere in the network. In this case we write a set of simultaneous equations incorporating all the branch rules as $h(x, f, u) = 0$. Any dependency on node displacements e can be eliminated since e is determined by a directed sum of branch displacements. For the case when the k^{th} branch's rule is an *affine function* of the branch variables, the branch rule has the form $\Gamma_k x + \Lambda_k f = \Phi_k u + \mu_k$. Combining the $k = 1, \dots, b$ branch equations we write $\Gamma x + \Lambda f = \Phi u + \mu$, where u is a vector of control inputs (e.g. independent forces and displacements) and μ is a vector of constants.

¹The reduced incidence matrix A is a special case of a cut set matrix Q in which the cut sets are chosen to be all branches incident on each node. Next consider the original digraph augmented by branches from the reference datum node to all other nodes. These branches form a tree which spans the graph and contains no loops (since they all fan out from the reference node). These branches of a tree are called *twigs* and thus we identify e with x_{twig} . Branches of the digraph which are not twigs are termed *links* and every link has a corresponding unique loop. On all such loops, KVL may be expressed as $x = Q^T x_{\text{twig}}$. The identification of e with x_{twig} and A as a special case of Q completes the argument.

A network is well-posed if given a fixed set of inputs u , the branch currents and forces are uniquely defined. Under this assumption, the system of equations which must be satisfied is

$$\underbrace{\begin{pmatrix} 0 & 0 & A \\ -A^T & I & 0 \\ 0 & \Gamma & \Lambda \end{pmatrix}}_T \underbrace{\begin{pmatrix} e \\ x \\ f \end{pmatrix}}_w = \underbrace{\begin{pmatrix} 0 \\ 0 \\ \Phi u + \mu \end{pmatrix}}_u.$$

That is, given the input forces and displacements, u , we wish to find the corresponding branch displacements and forces $w = T^{-1}u$. This is clearly possible only when the square *tableau matrix* T is invertible. This is true if and only if the system is well-posed.

Tableau analysis incorporates the loop and node constraints together with the branch rules as implicit equations in a unified framework. It is distinguished from nodal and loop analysis by not relying on each element's force being a function of its displacement and vice versa. This generality facilitates its use in automated solution algorithms such as that employed by SPICE, a widely available electrical circuit simulation package.

Tellegen's theorem, given below, is a graph theoretic result which has an interpretation as the conservation of power (and hence energy). Since Tellegen's theorem holds for any such x , it holds for $x(t)$ and $x(t + \delta t)$ and thus $(\dot{x}^T f)(t) \equiv 0$. This result is used later in this paper to derive the action of the actuator network on the mechanical system.

Theorem 1 (Tellegen) *Fix a reduced incidence matrix A . Let x and e be any set of branch and node displacements such that $x = A^T e$. Furthermore, let f be any set of branch forces such that $Af = 0$. Then*

$$x^T f = 0.$$

Proof. Since $x^T = e^T A$ and $e^T A f = 0$, then $x^T f = 0$. □

3 Finger-like Robot Systems

The review of mechanical system dynamics and graph theory in the previous section serves as a foundation for the results of this section. We now combine these tools to study the structure of finger-like biomechanical systems.

3.1 System description and assumptions

A finger-like system consists of 2 parts: an actuator-related system and an underlying mechanical system. The *mechanical system* \mathcal{M} is a collection of rigid bodies constrained to move in its configuration space Θ . In general, configuration θ includes angular as well as rectilinear quantities which, for this paper, take the form $\theta \in \Theta \subset \mathbb{R}^n$. For the purposes of this development of finger-like systems, we assume \mathcal{M} to already possess a formulation of its intrinsic dynamics (perhaps developed via Lagrangian mechanics) in the form of a second order differential equation expressing a balance of forces conjugate to configuration velocities:

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + N(\theta, \dot{\theta}) = 0.$$

The function N may contain friction or other dissipative forces in addition to forces derived from potential fields including gravity and spring forces.

The *actuator system* \mathcal{A} consists of a collection of force and displacement transmission elements which obey certain restrictions. These elements interact with each other and the mechanical system at junctions or nodes.

Assumption 1 Branches join at nodes and exist in a quasi-static force equilibrium. That is, any dynamics of \mathcal{A} are much faster than those of \mathcal{M} and so to track the behavior of \mathcal{M} we need only look at \mathcal{A} 's equilibrium forces and displacements.

Assumption 2 The geometry of the actuator system elements is a function only of configuration θ . Having frozen θ , finite displacements of the nodes and branches of \mathcal{A} occur in a fixed geometry. This assumption enforces a 1-D like nature to the actuator network which we shall exploit.

Denote the set of all nodes in the actuator system as \mathcal{N}° and the set of all actuator branches as \mathcal{B}° . The nodes and branches together form the connected graph $\mathcal{G}^\circ = (\mathcal{N}^\circ, \mathcal{B}^\circ)$. If this graph is not initially connected, it may be made connected by a process called *hinging* where disjoint subgraphs are joined at a common reference node.

Assumption 3 Connected graph \mathcal{G}^o is fixed over the entire configuration space, Θ .

Each branch has 2 scalar variables associated with it: displacement x and force f . We define positive displacement to be an increase in length of a branch. We take positive force in a branch to be that associated with the lengthening of that branch if it were a spring (i.e. tensile force in a branch is positive, compression is negative). Additionally, each branch has a direction that may be represented as an ordered pair of its nodes. Such definitions may be extended to rotary springs, gears, and other elements in a similar fashion, although they will not be developed here.

Each node n has a single scalar variable, its displacement e_n from the reference node (0). Node displacements have the interpretation of *directed distance* which agrees with the conventional notion of distance if all the branches along a path from (0) to (n) are similarly directed.

Assumption 4 The laws governing the behavior of the branch elements are affine in the set of branch variables and external inputs u .

As an example of a branch law, a spring with rest length a and spring constant k observes $f = k(x - a)$. More generally, for the vector of all branch forces f and displacements x there exist matrices $\Lambda(\theta)$, $\Gamma(\theta)$, $\Phi(\theta)$ and vector $\mu(\theta)$ and exogenous input vector u such that

$$\Lambda f + \Gamma x = \Phi u + \mu.$$

Such a representation permits independent force and displacement sources, controlled sources, and multiport composite elements such as gears and pulleys.

The set of nodes \mathcal{N}^o may be partitioned into those nodes which are in contact with the mechanical system, \mathcal{N}_M^o , and those which are purely actuator related, \mathcal{N}_A^o . Assumption 2 restricts actuator branches incident on nodes \mathcal{N}_A^o to be essentially colinear (see node (3) in Figure 2). This restriction is not necessary at nodes \mathcal{N}_M^o since they are fixed as functions of θ and one could equivalently alter the angle of incidence at the very site of attachment arbitrarily.

Definition 1 A finger-like robot system is a mechanical system \mathcal{M} with undriven dynamics

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + N(\theta, \dot{\theta}) = 0$$

which has been augmented by an actuator-related force/displacement generation and transmission system \mathcal{A} such that assumptions 1–4 are satisfied.

3.2 Constructing the actuator graph

The actuator graph corresponding to \mathcal{A} must be augmented to properly reflect the interaction with the mechanical system. To construct the complete actuator graph, we follow these steps:

1. Specify an undirected graph \mathcal{G}° representing the actuator network.
2. Assign directions to the branches and form a digraph \mathcal{G} .
3. Augment \mathcal{G} to model interaction with the physical system and form digraph \mathcal{G}^* .

The first step entails describing the interconnections between elements of the actuator network. In step 2, the geometry of this attachment is made explicit by defining which direction the actuator elements act relative to each other. Finally, in the last step, the interaction between the mechanical system and the actuator network is specified.

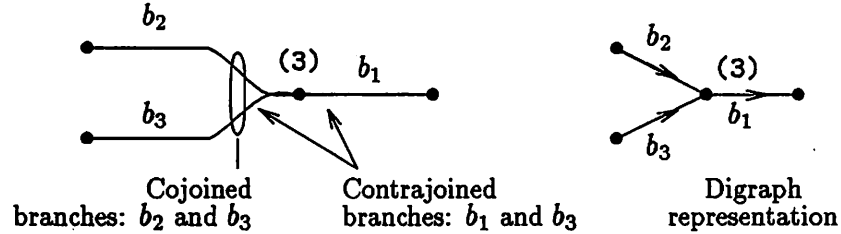
Original actuator graph, \mathcal{G}°

The algorithm for constructing the actuator digraph \mathcal{G}^* begins with a representation of the network as an undirected graph, \mathcal{G}° . Each branch in this graph corresponds to a physical device in the actuator system, such as a spring, force generator (muscle), or length of tendon. Nodes correspond to connections between these devices and to ends of devices which terminate on the mechanical system. We use (0) to represent the reference node and require that it be fixed on the mechanical system and independent of configuration θ . In a biological finger, a convenient choice for the reference node is a proximal skeletal site such as a muscle's point of origin in the forearm.

Formation of Digraph, \mathcal{G}

We next convert the original undirected graph \mathcal{G}° into a directed one, \mathcal{G} . This step basically consists of choosing arbitrarily certain branch directions and propagating consistent direction assignments over the remaining branches of the graph.

The assumption that the actuator geometry is fixed for θ fixed implies that the actuator network has a 1-dimensional nature. That is, wherever two actuator branches join, they either act in the same direction (*cojoined*) or in opposite directions (*contrajoined*). An example of this is shown in Figure 2.



Nodal incidence list for node (3): $\{b_2, b_3; b_1\}$

Figure 2: Examples of muscles which act in the same and opposite directions on a node. Note the 1-D nature of branch interaction at node (3) on the left. A nodal incidence list is formed by partitioning the incident branches into two disjoint sets of cojoined branches.

The physical arrangement at each node in \mathcal{N}_A^o is summarized by a *nodal incidence list* (see Figure 2) which may in turn be derived from a graphical diagram of the actuator network. The nodal incidence list bookkeeps the relative directions of branches incident on a node.

The directions assigned to the branches must be consistent with the force summation (and hence physical placement) of the actuator branches. If two actuator branches intersect at a node in \mathcal{N}_M^o , which corresponds to attachment with the mechanical system, it is not necessary for branch pairs to act in a 1-dimensional fashion, as discussed above. Thus, to propagate branch direction assignments we only need consider branches incident on nodes in \mathcal{N}_A^o and incorporate the information embodied in the nodal incidence lists.

Branch direction assignment proceeds according to the following steps. First, construct a modification of \mathcal{G}^o , denoted \mathcal{G}^m , in which all branches which terminate on a node in \mathcal{N}_M^o receive distinct nodes in \mathcal{N}_M^o . Next, form a tree for \mathcal{G}^m and for each twig incident on root node (0), choose a direction arbitrarily and ascend its subtree using the nodal incidence lists at each node to assign branch directions. The remaining undirected branches are links of \mathcal{G}^m and their direction is determined by the two nodal incidence lists at their nodes. For simplicity, we assume that a consistent direction assignment around all fundamental loops is always possible. Fundamental loops for which this is possible are termed *directable loops* (see Appendix A for examples of *undirectable loops* and methods for overcoming this limitation). At the conclusion of this process, all branches in \mathcal{G}^m , and hence \mathcal{G}^o ,

are now directed. The result is a digraph representation of the actuator system, $\mathcal{G} = (\mathcal{N}, \mathcal{B})$.

At the time the branches are labeled, the laws that govern each branch may also be stored. We cannot yet solve for the relations among the actuator elements, however. The mechanical system \mathcal{M} , considered frozen at configuration θ , constrains node positions and branch displacements and has not yet been tied to the actuator system. In order to obtain force balance at the nodes in \mathcal{N}_M , the digraph must be augmented with displacement sources to bookkeep forces which act on \mathcal{M} . Additionally, actuator branches act on \mathcal{M} by crossing bearing surfaces such as pulleys. For this case, the lengths of actuator loops are affected by configuration θ and configuration dependent displacement sources need to be introduced into loops to maintain meaningful loop constraints.

Finger-like system augmented digraph, \mathcal{G}^*

To construct the augmented graph, \mathcal{G}^* , we require a tree for the actuator digraph \mathcal{G} . For each node $(n) \in \mathcal{N}_M \setminus (0)$, attach a displacement generator between (0) and (n) such that the net length (sum of the directed branch lengths) around the newly created loop is zero. The magnitude of the displacement generator relies on the physical geometry of the actuator system and requires additional information beyond the individual branch relations. This process is continued for each node in \mathcal{N}_M .

To complete the construction of \mathcal{G}^* , the loop constraints for each fundamental loop are incorporated. For each such loop, insert a branch in-series with the link such that the net length (sum of the directed branch lengths) around the loop is zero. Again, this information is derived from the physical geometry of the actuator system and requires additional information beyond the individual branch relations.

As a result of augmenting the digraph with these displacement sources, a consistent and complete digraph representation of \mathcal{A} acting upon \mathcal{M} is obtained. We denote this augmented digraph of the finger-like system as $\mathcal{G}^* = (\mathcal{N}^*, \mathcal{B}^*)$. The branches of \mathcal{G}^* may be partitioned into two disjoint sets: \mathcal{B}_A^* from the original actuator system, and \mathcal{B}_M^* from the augmentation to incorporate the mechanical system. With the inclusion of the mechanical system's action, the whole system becomes thermodynamically closed—power is exchanged between \mathcal{A} and \mathcal{M} , but is conserved. By Tellegen's theorem, power is conserved ($\dot{x}^T f = 0$) in \mathcal{G}^* . Since $\mathcal{B}^* = \mathcal{B}_A^* \cup \mathcal{B}_M^*$, the power delivered to/from the mechanical system from/to the actuator system

is given by the branch forces and displacement velocities of branches B_M^* .

Tableau equations

The tableau equations for \mathcal{G}^* may be formed as

$$\begin{pmatrix} 0 & 0 & A \\ -A^T & I & 0 \\ 0 & \Gamma(\theta) & \Lambda(\theta) \end{pmatrix} \begin{pmatrix} e \\ x \\ f \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \Phi(\theta)u + \mu(\theta) \end{pmatrix}. \quad (5)$$

This system of linear equations is comprised of three parts: node force balance ($Af = 0$), loop constraints ($x = A^T e$), and device law characteristics ($\Gamma(\theta)x + \Lambda(\theta)f = \Phi(\theta)u + \mu(\theta)$). The device law characteristics are allowed to vary as the robot configuration varies. This is often necessary since as the robot configuration θ changes, the actuator network geometry changes. The actuator system's interconnections might also change with configuration but in this treatment are considered fixed or to have been reflected in the device laws.

The inverse of the tableau matrix exists by the well-posedness assumption and premultiplication of both sides of (5) yields the branch and node variables as functions of θ and u .

3.3 Main results

Lemma 1 *The actuator network variables e, x, f are affine functions of the control input forces and displacements u under the restriction that the underlying robot's state $(\theta, \dot{\theta})$ is fixed and the actuator mechanism is well-posed. To fix notation we write*

$$\begin{aligned} e &= P_e(\theta) + Q_e(\theta)u \\ x &= A^T P_e(\theta) + A^T Q_e(\theta)u \\ f &= P_f(\theta) + Q_f(\theta)u. \end{aligned}$$

Proof. The inverse of the tableau matrix exists by the well-posedness assumption. The result follows upon inverting (5). \square

Theorem 2 *The joint torques applied by the actuator network are affine in control input u*

$$\tau = G(\theta)u + S(\theta).$$

Proof. Let $\mathcal{B}_M^* \subset \mathcal{B}^*$ be the set of branches in \mathcal{G}^* which model the interaction between the actuator and mechanical systems (i.e. the augmented branches of the previous section). Each of these branches consists of an independent displacement generator which models the branch displacement as a function of θ . Hence

$$x_a =: P_a(\theta),$$

where x_a is the vector of branch displacements of the augmenting, independent displacement generators. Since these branches fully describe the power transfer between the actuator and mechanical systems, we can apply the principle of virtual work. Let f_a be the branch forces associated with x_a so that

$$\begin{aligned} \tau &= (D_\theta P_a)^T f_a \\ &= (D_\theta P_a)^T (P_f(\theta) + Q_f(\theta)u)|_a, \end{aligned}$$

where the last equation is obtained by restricting the full set of branch force equations to the augmenting branches, \mathcal{B}_M^* . Defining

$$\begin{aligned} S(\theta) &:= (D_\theta P_a)^T P_f(\theta)|_a \\ G(\theta) &:= (D_\theta P_a)^T Q_f(\theta)|_a \end{aligned}$$

completes the proof. □

Corollary 1 *The dynamics for a finger-like system have the form*

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + (D_\theta V)^T(\theta) = G(\theta)u + S(\theta). \quad (6)$$

Equation (6) reveals the actuator control inputs u enter linearly through the $G(\theta)$. For overactuated systems, $G(\theta)$ is non-square with a null space corresponding to inputs in which internal actuator tensions may vary without developing net force on the mechanical system. The $S(\theta)$ term models spring-like energy storage in the actuator network. If input u is zero, then $S(\theta)$ is the restoring force generated as a result of bending the finger away from its equilibrium position. Robots driven by torque motors at their joints are a special case with $S = 0$, $G = I$, and $u = \tau$.

3.4 Remarks

In this formulation, models of actuator systems may include elements that respond to forces or displacements elsewhere in the system. Unless these relationships are balanced in a particular fashion, however, the resulting

system will not be *reciprocal*. In a Lagrangian based formulation of dynamics, at a particular configuration θ the potential energy stored in the actuator branches is $V(x)$. Branch force, then, is the gradient of the potential function $f = \nabla_x V = (D_x V)^T$. The linear map between δx and δf is the hessian, $D_x^2 V$, which is symmetric by the equality of mixed partials. It is possible in finger-like systems, however, to obtain nonsymmetric linear maps and thus go beyond the realm of a direct Lagrangian formulation.

This symmetry is the hallmark of a reciprocal system. Roughly speaking, in a reciprocal system if an additional force δf across branch i causes a additional displacement δx across branch j , then applying δf across branch j should cause displacement δx across branch i . Many linear and nonlinear system models are not reciprocal, including elementary models of transistors and fluidic amplifiers. The inclusion of controlled sources often results in a lack of reciprocity. For example, the method used here allows us to embed an affine control law as part of the actuator network rather than as a separate stage outside of the mechanical and actuator systems. An example of a nonreciprocal system will be given in Section 4.2.

Tableau analysis allows a general class of actuator networks. Both displacement-controlled force sources and force-controlled displacement sources may be included within the same framework. This greatly facilitates automating the solution over such techniques as nodal and loop analysis. Our exposition applies to actuator networks which are affine in the control inputs, having fixed θ . Nonlinear actuator networks can be approached in a similar fashion provided the actuator kinematics has a unique solution. This corresponds to the condition that the T matrix in (5) be invertible. If the nonlinear system is invertible in this sense, then we can write $f = P_f(\theta, u)$ and hence

$$\tau = (D_\theta P_a)^T P_f(\theta, u)|_a.$$

Solving for actuator branch forces f in closed form will not generally be possible for the nonlinear case and numerical methods must be employed.

Although the actuator network may contain numerous implicit relations, the form of the equations for affine networks allows a simple computer implementation. We have implemented the calculations required for the finger-like analysis in Mathematica [14], a symbolic manipulation program. By specifying a node and branch list together with selected relations for the lengths along certain actuator system paths, it is possible to solve for $G(\theta)$ and $S(\theta)$ in symbolic form. All of the examples presented in this paper were analyzed using this software. The software is available via anonymous ftp

from `robotics.berkeley.edu` in the directory `pub/Fingerlike`.

We also note that in the case that the branch extensions are linear in the configuration variable θ , G and S become constant matrices. Simplifications are possible, then, when mechanical pulleys of uniform radius are present where tendons move across joints (e.g. many tendon-driven mechanical hands [13]). In the following example, however, G is dependent on θ .

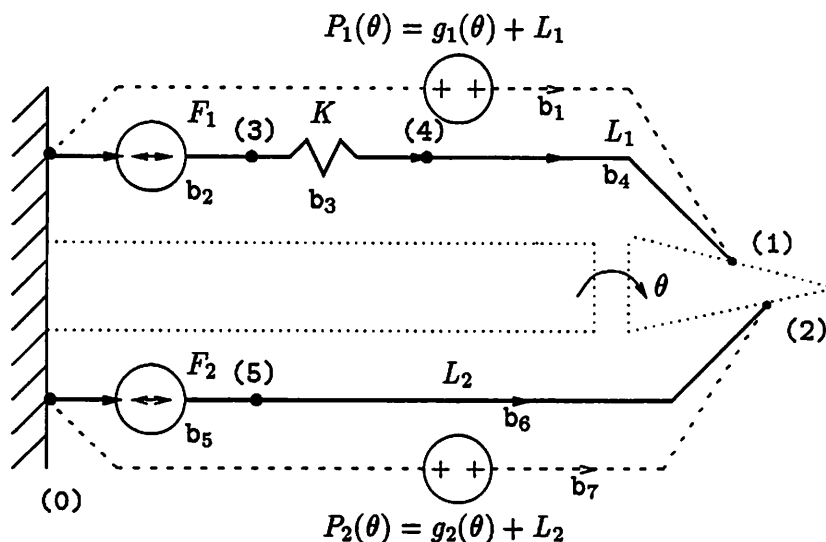


Figure 3: Branch and node assignments for Figure 1. The solid lines are branches of the actuator system graph. The thick lines form a tree which spans the graph and the thin lines are the links (of which there are none in this example). Dashed lines correspond to augmenting branches.

4 Examples

4.1 Finger-like treatment of previous example

In this section we illustrate the graph based technique for finger-like systems with the 1-joint robot shown in Figure 1 and previously analyzed by a Lagrangian formulation in Section 2.1. We begin by labeling the nodes and branches of the system (see Figure 3). To assign directions to the branches, we note that the reference node (0) is part of the mechanical system and hence we can assign directions arbitrarily to b_2 and b_5 . Once these directions are chosen, all other directions in the graph are directed similarly, since the branches in each tendon act in the same direction (cojoined).

To augment the digraph, we attach displacement generators from nodes (1), (2) $\in \mathcal{N}_M$ to node (0). These displacement generators bookkeep the extension of the tendons by requiring that the net displacements around resulting loops is zero. The displacements are determined by using the

models described in Section 2.1:

$$\begin{aligned} x_1(\theta) &= x_2 + x_3 + x_4 = L_1 + R_1\theta \\ x_2(\theta) &= x_5 + x_6 = L_2 - (R_2 + R'_2\theta)\theta. \end{aligned}$$

We now proceed with the tableau analysis. The incidence matrix for the system, with its branches and nodes labeled, is

$$A_a = \begin{array}{c|ccccccc} & b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 \\ \hline (1) & +1 & 0 & 0 & +1 & 0 & 0 & 0 \\ (2) & 0 & 0 & 0 & 0 & 0 & +1 & +1 \\ (3) & 0 & +1 & -1 & 0 & 0 & 0 & 0 \\ (4) & 0 & 0 & +1 & -1 & 0 & 0 & 0 \\ (5) & 0 & 0 & 0 & 0 & +1 & -1 & 0 \\ \hline (0) & -1 & -1 & 0 & 0 & -1 & 0 & -1 \end{array}$$

with the bottom row corresponding to the reference node (0). Note that all columns have exactly one +1 and one -1 entry (each branch enters and leaves once) and the sum across the rows is the net number of branches entering or leaving the given node.

Next we write the device laws for each branch. Letting x represent the displacement across the branch and f the tension, we use the primitive device laws described in Table 1. We have

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -K & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\Gamma} x + \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\Lambda} f = \underbrace{\begin{pmatrix} 0 \\ F_1 \\ 0 \\ 0 \\ F_2 \\ 0 \\ 0 \end{pmatrix}}_{\Phi u} + \underbrace{\begin{pmatrix} P_1(\theta) \\ 0 \\ 0 \\ L_1 \\ 0 \\ L_2 \\ P_2(\theta) \end{pmatrix}}_{\mu}.$$

We construct the tableau matrix

$$T = \begin{pmatrix} 0 & 0 & A \\ -A^T & I & 0 \\ 0 & \Gamma & \Lambda \end{pmatrix}$$

and solve $T\mathbf{w} = \mathbf{u}$ to obtain

$$e = \begin{pmatrix} P_1 \\ P_2 \\ P_1 - F_1/K - L_1 \\ P_1 - L_1 \\ P_2 - L_2 \end{pmatrix}, \quad x = \begin{pmatrix} P_1 - F_1/K - L_1 \\ F_1/K \\ L_1 \\ P_2 - L_2 \\ L_2 \\ P_2 \end{pmatrix}, \quad \text{and} \quad f = \begin{pmatrix} -F_1 \\ F_1 \\ F_1 \\ F_1 \\ F_2 \\ F_2 \\ -F_2 \end{pmatrix}.$$

The augmenting branches are b_1 and b_7 and hence we have

$$x_a = \begin{pmatrix} x_1 \\ x_7 \end{pmatrix} = \begin{pmatrix} P_1(\theta) \\ P_2(\theta) \end{pmatrix} =: P_a(\theta)$$

$$f_a = \begin{pmatrix} f_1 \\ f_7 \end{pmatrix} = \begin{pmatrix} F_1 \\ F_2 \end{pmatrix}$$

and

$$\begin{aligned} \tau &= D_\theta P_a(\theta)^T f_a = \begin{pmatrix} D_\theta P_1 & D_\theta P_2 \end{pmatrix} \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} \\ &= \begin{pmatrix} -R_1 & (R_2 + 2R_2'\theta) \end{pmatrix} \begin{pmatrix} F_1 \\ F_2 \end{pmatrix}. \end{aligned} \quad (7)$$

The analysis above is consistent with the results of the Lagrangian technique given in Section 2.1 above. The righthand side of (4) is identical to the expression for τ in equation (7). In a completely Lagrangian derivation, $S(\theta)$ would be lumped into $(D_\theta V)^T$ since it is a potential energy term. In this simple example, however, $S = 0$. Also, since the force generator inputs act directly on the underlying mechanical system, the actuator matrix $G(\theta)$ is simply the Jacobian transpose $(D_\theta P)^T$. The Mathematica code used to analyze this example is given in Table 2.

It is interesting to consider the case where nodes (1) and (2) in the example above are coincident. In this case, the construction of digraph \mathcal{G} involves an intermediate digraph \mathcal{G}^m where these nodes are again separated. The analysis could proceed using \mathcal{G}^m and would be identical to that just described. Instead, the finger-like procedure to generate augmented digraph \mathcal{G}^* would first add a displacement generator from (0) to the common node in \mathcal{N}_M and then insert another displacement generator to enforce the loop constraint. Solving the tableau equations for this new graph yields the same solutions for actuator branch variables and the same differential equation for the dynamics.

```

(* example1.m - Mathematica solution to example *)
<<Fingerlike.m

(* Tendon displacement functions *)
P1[theta_] := R1 theta + L1
P2[theta_] := -(R2 + R2p theta) theta + L2

BuildTableau[ex1,
  Nodes[{n1,n2,n3,n4,n5,base}];
  (* Branch laws for upper tendon *)
  b1 = Displacement[base, n1, P1[theta]];
  b2 = Force[base, n3, F1];
  b3 = Spring[n3, n4, K];
  b4 = Displacement[n4, n1, L1];
  (* Branch laws for lower (non-compliant) tendon *)
  b5 = Force[base, n5, F2];
  b6 = Displacement[n5, n2, L2];
  b7 = Displacement[base, n2, P2[theta]];
];

A = IncidenceMatrix[ex1]; (* get incidence matrix *)
v = SolveTableau[ex1]; (* solve tableau eqs *)

tau = FingerlikeTorques[ex1, {b1,b7}, {theta}];

```

Table 2: Mathematica listing for solution to 1-joint robot example. This example uses the package `Fingerlike.m` described in Appendix B.sec.

4.2 Biomechanical example of nonreciprocity

Consider the system shown in Figure 4. In this model the actuators are arranged in an agonist-antagonist pair. The net force developed on the mass M depends indirectly on the inputs $u_1 = F_1$ and $u_2 = F_2$: actuator force is modified by the displacements or stretch of the opposing cables or tendons. Such a system may be viewed as an extension of Figure 1 and is a simple model of the knee-jerk stretch reflex [4].

Unless $g_{12} = g_{21}$, the system is not reciprocal. As a result, the application of Lagrangian mechanics is blocked at the stage of writing down an expression for the potential energy. However, it is possible to represent Figure 4 by a reciprocal (spring-mass-spring) mechanical system with a more complex actuator system attached to it. Such a treatment would not exploit the algebraic structure of finger-like systems as directly.

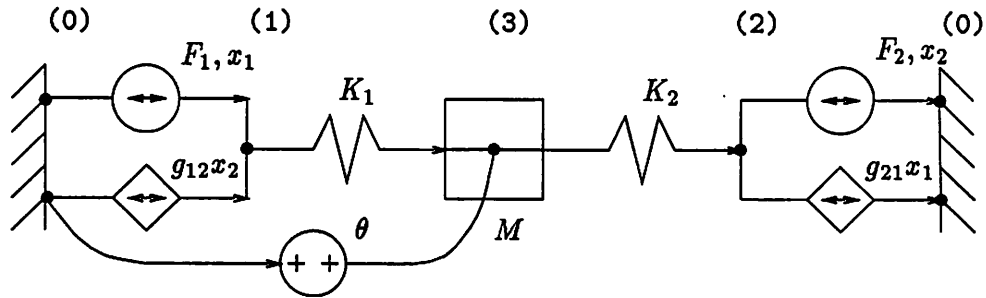


Figure 4: A nonreciprocal biomechanical system. This diagram represents a simple version of the stretch reflex in biological motor control. The action of an agonist-antagonist pair is modeled by two pairs of force generators. The centrally derived contraction signals are modeled by independent force generators F_1 and F_2 . The action of muscle spindle stretch receptors on spinal alpha motorneurons is modeled by the controlled force generators. The (Ia fiber) stretch information is shown inhibiting the antagonist muscle only (i.e., $g_{ij} < 0, i \neq j$), ignoring the excitation of the homonymous agonist (for which $g_{ii} > 0$) [4]. This system is nonreciprocal if $g_{12} \neq g_{21}$.

Using the method described in this paper, it is possible to analyze this nonreciprocal example. Generating the actuator graph with the appropriate device laws and solving the corresponding tableau equation yields

$$\tau = S(\theta) + \begin{bmatrix} G_1 & G_2 \end{bmatrix} \begin{pmatrix} F_1 \\ F_2 \end{pmatrix}$$

$$S(\theta) = \frac{K_1 K_2 (g_{12} + g_{21}) + g_{12} g_{21} (K_1 + K_2)}{K_1 K_2 - g_{12} g_{21}} \theta$$

$$G_1 = -\frac{K_2 (K_1 + g_{21})}{K_1 K_2 - g_{12} g_{21}}$$

$$G_2 = \frac{K_1 (K_2 + g_{12})}{K_1 K_2 - g_{12} g_{21}}$$

4.3 Human finger model, analysis

In this section we apply the finger-like method to analyze the kinematics and dynamics of a simplified human finger model. The paper of Buchner, Hines, and Hemami [2] treats a prototypical finger which consists of 3 rotary joints, 5 muscle actuators, and both inextensible and spring-like tendons. This

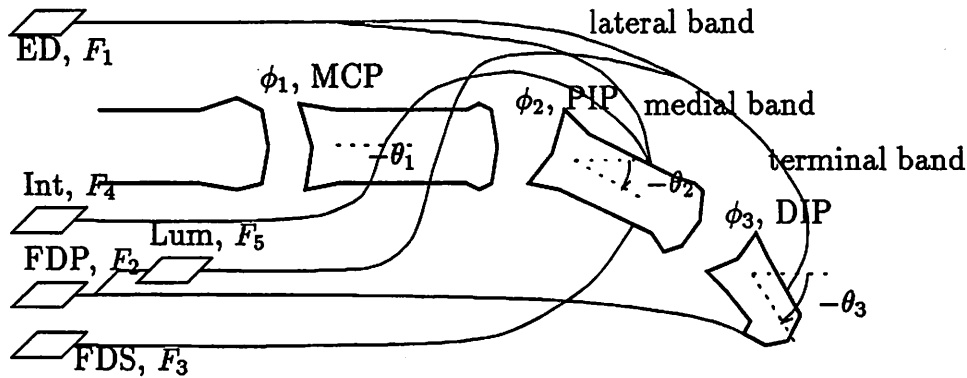


Figure 5: Simplified representation of muscles and tendons involved in controlling a human finger [2]. The angles made by the proximal, middle, and distal phalange (finger) segments are denoted θ_1 through θ_3 . The joint or hinge angles are denoted by $\phi_1 = \theta_1$, $\phi_2 = \theta_2 - \theta_1$, and $\phi_3 = \theta_3 - \theta_2$. Actuation has been simplified to a set of 5 muscles: extensor digitorum communis (ED), flexor digitorum profundus (FDP), flexor digitorum superficialis (FDS), the palmar and dorsal interossei (Int), and the lumbricals (Lum). The ED tendon splits into medial and lateral bands between joints 1 and 2 and the Lum muscle branches off the FDP tendon. The metacarpal-phalangeal (MCP) joint is nearest the wrist with the proximal interphalangeal (PIP) and distal interphalangeal (DIP) joints further out.

model, shown in Figure 5, employs a number of tendon cables and includes complex pulley and tendon sheath pathways as part of the actuator network. The configuration of the underlying robot is described by the angles of the 3 skeletal segments θ_1 , θ_2 , and θ_3 . Actuator muscles are modeled as force generators: extensor digitorum (ED, F_1), palmar and dorsal interossei (Int, F_4), flexor digitorum profundus (FDP, F_2), flexor digitorum superficialis (FDS, F_3), and the lumbricals (Lum, F_5).

The actuator network diagram of Figure 6 is a reformulation of the anatomy of Figure 5 and serves to guide the analysis of the complete system's dynamics. The four nodes where the actuator network contacts the robot mechanical object are labeled (1)–(4). The Buchner *et al.* model conforms to the finger-like requirements since tendon mass and friction is assumed negligible. The dynamics are readily formed from combining the usual Lagrangian mechanics formulation for the robot rigid bodies and the finger-like system actuator matrix $G(\theta)$ and actuator network spring forces

$S(\theta)$:

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + (D_\theta V)^T(\theta) = S(\theta) + G(\theta)u$$

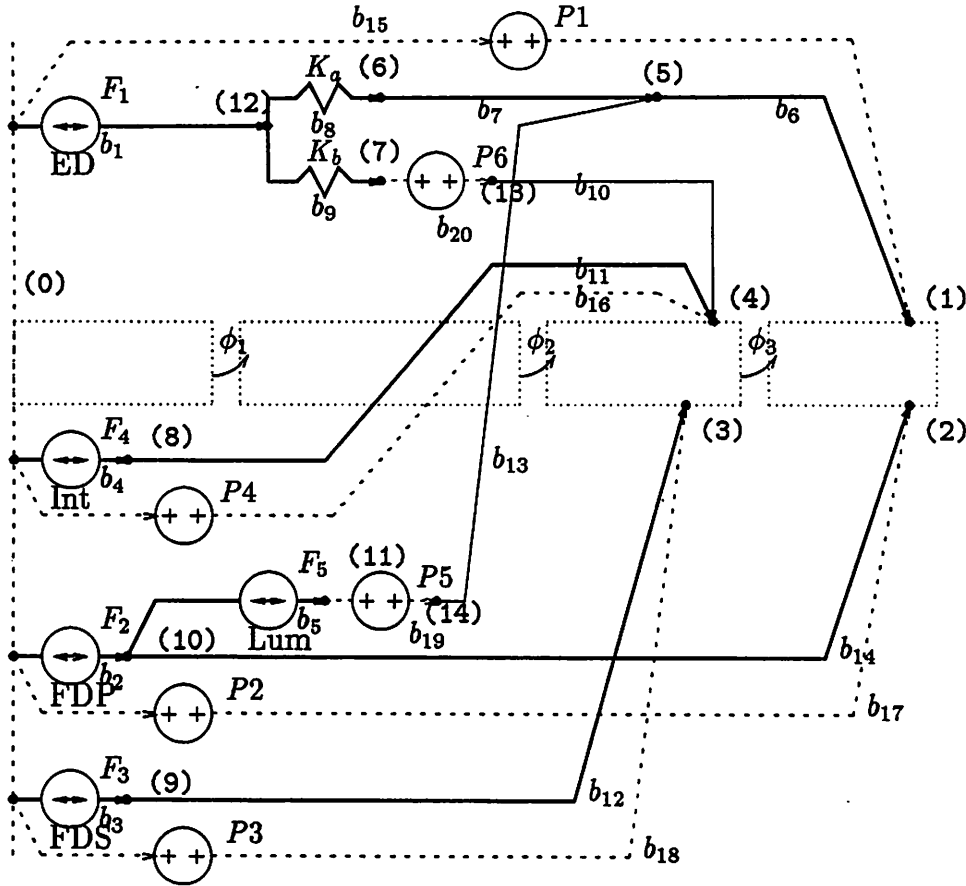
where $u = (F_1, \dots, F_5)$ is the vector of muscle force inputs.

The solid lines in Figure 6 are all branches of the actuator system digraph \mathcal{G} . The thick lines form a tree which spans the graph and the thin lines are the links. The finger-like system digraph is not complete without two additional nodes, (13) and (14), and six additional branches, b_{15} – b_{20} . This augmentation of the actuator system completes the full system digraph, \mathcal{G}^* , by adding the constraints resulting from the mechanical system. It is through these six augmenting displacement sources that the actuator and mechanical systems exchange power. These displacement sources come from two distinct categories:

- Force balance at nodes (1)–(4) is accomplished by augmenting the graph with displacement sources, b_{15} – b_{18} . The addition of these branches in parallel with the path from the reference node (0) to the mechanical system attachment nodes is done in a fashion which implements the actuator geometry as a loop constraint and thus satisfies the loop or path law (KVL).
- The zero sum of branch displacements around the two fundamental loops indexed by links b_{10} and b_{13} is accomplished by augmenting the graph with displacement sources, b_{19} and b_{20} . The insertion in series with the corresponding link automatically satisfies the force balance vertex law (KCL).

The functions for these displacement sources comes from length and joint angle measurements on humans during surgery or postmortem. In Buchner *et al.* these relations are given quadratic approximations. A good example is the displacement source of Figure 6's branch 16 where $x_{16} = -[R_{41}\phi_1 - R_{42}\phi_2]$. The term $R_{41}\phi_1 - R_{42}\phi_2$ is the length from reference node (0) along the Interosseus muscle and tendon to the site of attachment, node (4). The positive values for R_{41} and R_{42} provided by Buchner *et al.* are consistent with the path taken by the Interosseus tendon b_{11} . It travels on the palmar side of joint 1 so that extension of the joint (positive θ_1 direction) leads to a lengthening of the interosseus muscle/tendon combination. Similarly, since the tendon travels on the dorsal side of joint 2, extension of this joint leads to a decrease in the length of the interosseus muscle/tendon combination.

In Buchner's development, ϵ is the relative displacement of the lateral band and terminal tendon versus the medial band and K is the associated



$$\begin{aligned}
 P1 &= -R_{11}\phi_1 - (R_{12} - R'_{12}\phi_2)\phi_2 - R_{13}\phi_3 \\
 P2 &= (R_{21} + R'_{21}\phi_1)\phi_1 + (R_{22} + R'_{22}\phi_2)\phi_2 + (R_{23} + R'_{23}\phi_3)\phi_3 \\
 P3 &= (R_{31} + R'_{31}\phi_1)\phi_1 + R_{32}\phi_2 \\
 P4 &= R_{41}\phi_1 - R_{42}\phi_2 \\
 P5 &= -[R_{51}\phi_1 - (R_{52} + R'_{52}\phi_2)\phi_2 + R_{11}\phi_1 + (R_{12} - R'_{12}\phi_2)\phi_2] \\
 P6 &= R_{11}\phi_1 + R_{12}\phi_2 + R_{41}\phi_1 - R_{42}\phi_2
 \end{aligned}$$

Figure 6: Network diagram of finger-like version of Buchner *et al.* finger model corresponding to the more anatomically based Figure 5. Hinge angles ϕ_1 , ϕ_2 , and ϕ_3 are positive in the direction of extension. The finger is shown fully extended in its reference configuration $\phi_i = 0$. Common reference node (0) is shown at left. Nodes (1)–(4) are the sites of attachment of the actuator network to the mechanical system. Bold lines indicate tree branches, dashed lines correspond to augmenting branches.

spring constant. The springs K_a and K_b in Figure 5 represent an equivalent description ($K_a = K/(1 - \alpha)$ and $K_b = K/\alpha$) so that the series spring $K = K_a K_b / (K_a + K_b)$.

The Buchner *et al.* paper contains a basic error, in addition to some minor misprints. They implicitly assumed nodes (1) and (2) were connected together and thus, in our finger-like system parlance, wrote the equivalent of the loop constraint for the fundamental loop indexed by branch b_{13} as going back along the FDP's tendon, b_{14} . The impact of this error on their analysis of muscle forces during finger motion trajectories is unclear, but is an example of the complexity of tendon kinematics. After having structured the analysis by the finger-like methods described in this paper it would be more difficult to make such a mistake.

The solution for Buchner *et al.* finger model, derived using the software package described in Appendix B, is given by

$$\begin{aligned}
 G_{11} &= R_{11} \\
 G_{12} &= -R_{21} - 2R'_{21}\phi_1 \\
 G_{13} &= -R_{31} - 2R'_{31}\phi_1 \\
 G_{14} &= -R_{41} \\
 G_{15} &= R_{21} - R_{51} + 2R'_{21}\phi_1 \\
 \\
 G_{21} &= \frac{K_a R_{12} + K_b R_{12} - 2K_a R'_{12}\phi_2}{K_a + K_b} \\
 G_{22} &= -R_{22} - 2R'_{22}\phi_2 \\
 G_{23} &= -R_{32} \\
 G_{24} &= R_{42} \\
 G_{25} &= R_{22} + R_{52} + 2R'_{22}\phi_2 + 2R'_{52}\phi_2 \\
 \\
 G_{31} &= \frac{K_a R_{13}}{K_a + K_b} \\
 G_{32} &= -R_{23} - 2R'_{23}\phi_3 \\
 G_{33} &= 0 \\
 G_{34} &= 0 \\
 G_{35} &= R_{13} + R_{23} + 2R'_{23}\phi_3
 \end{aligned}$$

and a stiffness term

$$S_1 = 0$$

$$S_2 = \frac{-2K_a K_b R'_{12} \phi_2 (R'_{12} \phi_2^2 - R_{13} \phi_3)}{K_a + K_b}$$

$$S_3 = -\frac{K_a K_b R_{13} (-R'_{12} \phi_2^2 + R_{13} \phi_3)}{K_a + K_b}.$$

There are clearly many possible geometric quantities and forces and displacements which could be identified. A virtue of the finger-like system results described above is a focusing on a reduced set of variables sufficient to describe the dynamics.

4.4 Human finger model, control

From the standpoint of trajectory generation, the actuator redundancy requires some resolution scheme. At each point in time along a desired trajectory, $\theta(t)$, there appear to be 2 degrees of freedom remaining after choosing 5 muscle forces to provide 3 joint torques. The minimum-norm actuator force approach has been championed in biological systems by Pellionisz [9], wherein:

$$u^+ = G^+ (M\ddot{\theta} + C + N - S). \quad (8)$$

However, it encounters difficulties as a result of nonnegative cable/muscle force restrictions.

In general, there may be a subset of m branches which are *cable-like* in the sense that their branch forces, denoted f_c , must be > 0 . Modifying the notation of Lemma 1 slightly, we restate this in terms of the existence of inputs $u \in U_c = \{u : f_c = P_c(\theta) + Q_c(\theta)u > 0\}$. A necessary and sufficient condition for the existence of admissible inputs $u \in U_c$ which can produce arbitrary joint torques τ is that $\ker(G) \cap U_c$ be nonempty. In order to reject arbitrarily large disturbances, it is sufficient that u be both in $\ker(G)$ and that $Q_c(\theta)u > 0$. Under this condition, one could always add enough of $u \in \ker(G)$ to input u^+ in equation (8) to keep $f_c \geq 0$. The existence of a kernel or null space of G requires overactuation, and so redundancy serves an important role in tendon driven systems.

A sample trajectory where the Buchner finger model is flexed and then extended over an interval of one second was created to illustrate all these ideas. A sequence of finger configurations is shown in Figure 7 with corresponding finger segment angles and torques shown in Figure 8.

Actuator forces necessary to achieve this trajectory are shown in Figure 9. The set of muscle forces computed from equation (8) are all in the ± 10 N range which correspond to weights of about ± 1 kg. Also shown in

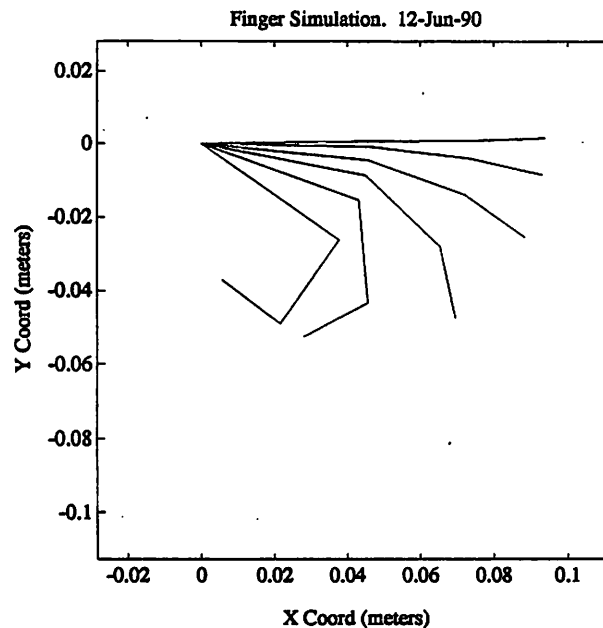


Figure 7: Sequence of finger configurations at 0.1 second intervals. The sequence begins with the finger fully extended and at zero velocity. One half second later, the finger is almost fully flexed. The remainder of the trajectory is nearly the reverse of this portion.

Figure 9 are the set of smallest (2-norm) muscle forces which also command the same motion, but obey the positivity constraints. These forces are in the 0–150 N range, an increase by a factor of roughly 10 in dynamic range.

The muscle input forces to achieve this trajectory depend, rather sensitively, on the precise trajectory. Near the fully flexed configuration, a 5% variation in joint angles requires a twenty-fold increase in input force. Thus it is apparent that good robot design and control algorithms are complementary—the controller can avoid trajectories near highly sensitive configurations and good design can make these less likely.

Design heuristics for a finger-like system might include the following over the robot's workspace:

- Maintain a small actuator related spring force, $\|S(\theta)\|$, so that actuator inputs are not transferred into stored energy.
- Seek a large induced norm $\|G(\theta)\|$ for good mechanical advantage,

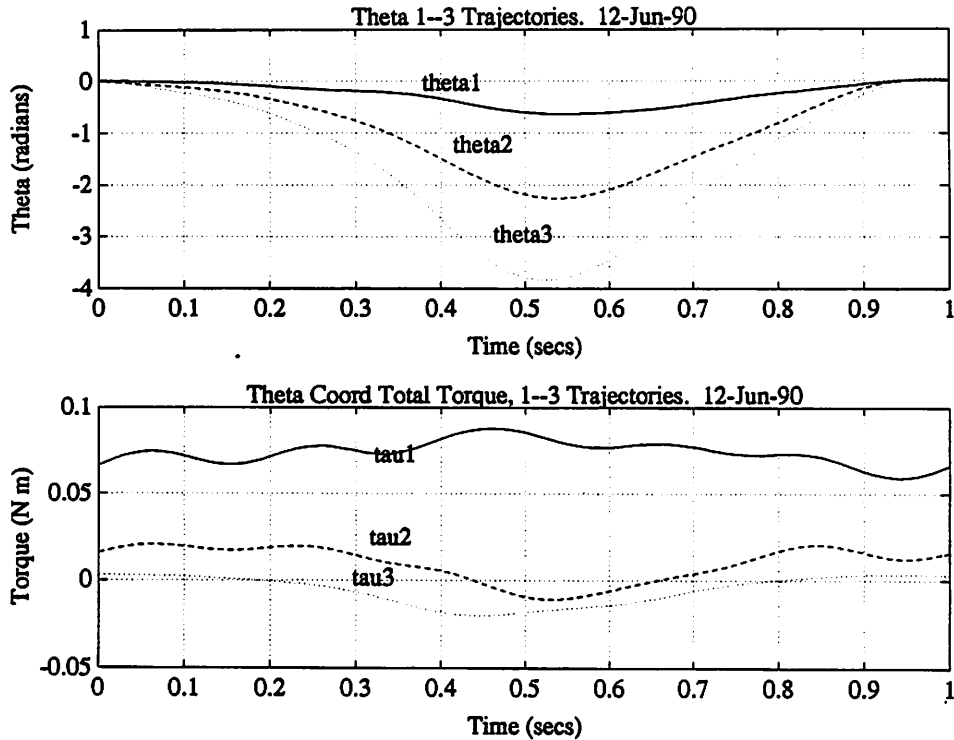


Figure 8: Finger segment angles θ and corresponding conjugate torques τ over the 1 second example trajectory shown in Figure 7.

efficiently transforming inputs into mechanical system torques.

- Arrange the actuation network such that the condition number of $G(\theta)$ is small, implying balanced actuators.
- Design to provide sufficient actuators and attachment sites and/or loops to control the mechanical system. If there are fewer augmenting displacement sources than configuration variables, then it is not possible to instantaneous achieve trajectory matching. However, average matching may still be possible (e.g. nonholonomic system control schemes which generate bracket direction motion).

The finger-like system analysis tools described in this work are offered to assist in the evaluation of designs.

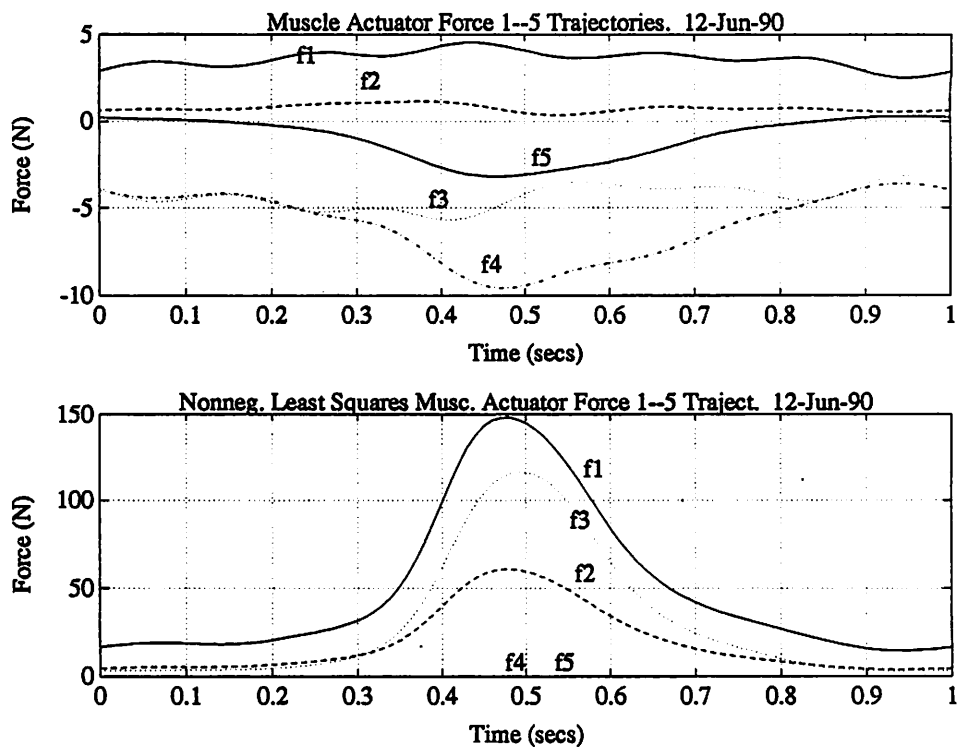


Figure 9: Muscle actuator force trajectories to provide the torque and motion of Figures 7 and 8. The top panel shows the forces which are computed at each point in time by (8), a straightforward least squares solution. The bottom panel shows input force trajectories when subject to the additional restriction of nonnegative actuator and tendon forces. The dynamic range of these inputs is about 10 times greater to obey nonnegativity constraints.

5 Conclusion

We have presented a graph-theoretic method to derive the quasi-static relationship between a control input u and the resulting joint torques $\tau = G(\theta)u + S(\theta)$ for a class of tendon-driven systems referred to as finger-like systems. As a result of this simple dependence on input u , control laws for this class of systems may be based on those already in use. Information relating inputs u to applied torques τ and branch forces f is accessible through a tableau analysis of the actuator branch variables. This information is useful for both the selection of inputs and the design of the actuator system itself.

The finger-like system algorithm consists of the following steps. The actuator system is represented as a graph, \mathcal{G}^o whose branches correspond to elements such as cables, springs, and actuators which are joined at nodes. Device law characteristics are entered as well as a description of the geometry of branches incident on the nodes. From this information a directed graph, \mathcal{G} , of the actuator system is formed. The digraph is then augmented with displacement sources to model the interaction between the actuator and mechanical systems. This final digraph, \mathcal{G}^* , is used for tableau analysis and the solution of all branch and node variables at a particular configuration θ . The expressions for branch displacement and force in the augmenting displacement sources are used to determine the actuator-applied torques on the underlying mechanical system.

The process has been automated with the help of symbolic manipulation software. This technique allows complicated actuator networks with interdependencies of force and displacement to be analyzed in a straightforward manner. The primary advantages of this method are its systematic and computational nature.

Acknowledgements

The authors would like to thank Karin Hollerbach for her efforts with human finger modeling at the outset of this project.

This work was supported in part by the Smith-Kettlewell Eye Research Foundation and Rachael C. Atkinson Fellowship Award (DCD), an IBM Manufacturing Fellowship and NSF IRI-90-14490 (RMM), the Semiconductor Research Corporation (KSJP), and by NSF under DMC 84-51129, ECS 87-19298, and IRI-90-14490 (SSS).

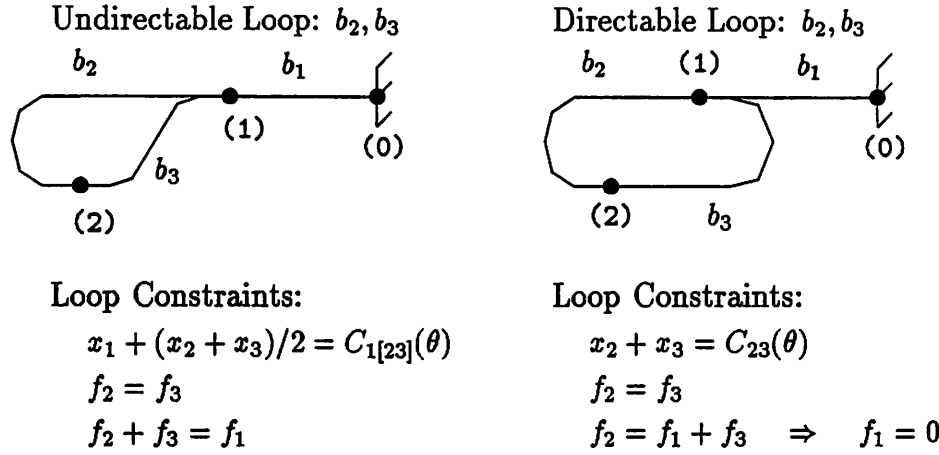


Figure 10: Two loops with different nodal incidence lists at node (1) illustrate the physical reason for undirectable loops. Indeed the simplest example consists of just one branch in a similarly constructed self loop. The loop displacement constraint conforms to intuition in that a frictionless hook placed in the loop and pulled to the left extends branches b_1 , b_2 , and b_3 in a manner which determines $(x_1 + (x_2 + x_3)/2)(\theta)$.

A Undirectable Loops

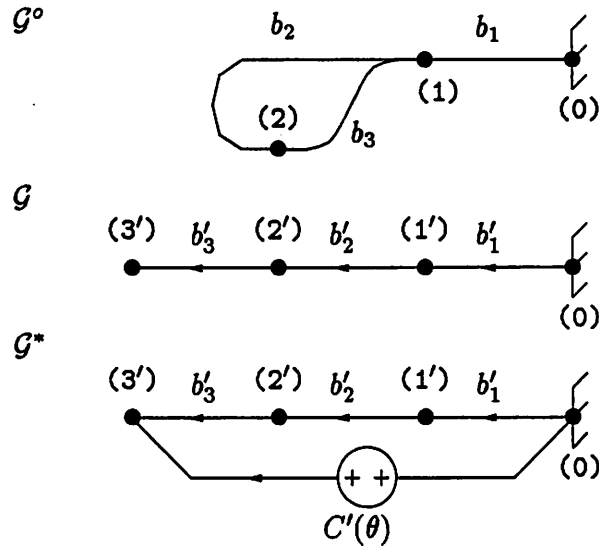
The possibility of encountering undirectable loops arose previously in Section 3.2 during the discussion of branch direction assignment. A graph consists entirely of directable loops if consistent directions can be assigned to all twigs and links, thereby forming \mathcal{G} . A simple illustration of an undirectable loop appears in Figure 10.

The finger-like analysis technique may be extended to encompass certain undirectable loops at the expense of introducing additional variables to bookkeep transformed branch and node variables belonging to the affected loop(s). The technique is illustrated in Figure 11. It is significant that there are no branches incident on the nodes of this undirectable loop except those belonging to the loop itself. Such a loop is termed an *isolated undirectable loop*.

As a result of the transformation, an equivalent problem is formulated in terms of the primed set of variables. Note that $x' = Hx$ and $f' = H^{-1}f$, where H is a diagonal matrix of 1's and 1/2's. The tableau equations, $T'w' = u'$, again consist of three sets of algebraic relations: $A'f' = 0$ (vertex

laws), $-(A')^T e' + x' = 0$ (loop laws), and $\Gamma' x' + \Lambda' f' = \Phi u + \mu$ (device laws). The directed graph in Figure 11's third panel provides the incidence matrix A' and hence the first two algebraic relations. The device law relations, given originally in terms of the unprimed variables, $\Gamma x + \Lambda f = \Phi u + \mu$ are readily cast into the primed variables using H and H^{-1} .

This example may be extended to situations with multiple, isolated, undirectable loops. This follows by induction since each undirectable loop may be resolved by a transformation into a completely equivalent problem. The original graph \mathcal{G}^o is assigned directions in procedure consisting of choosing arbitrarily certain branch directions and propagating, consistently, direction assignments over the remaining branches of the graph. If an assignment inconsistency appears when closing isolated, fundamental loops, then each such loop is rendered directable by transforming it into an equivalent digraph. A comprehensive treatment for arbitrarily arranged undirectable loops is not yet available.



$$\begin{aligned}
 x'_1 &= x_1 & f'_1 &= f_1 \\
 x'_2 &= x_2/2 & f'_2 &= 2f_2 \\
 x'_3 &= x_3/2 & f'_3 &= 2f_3 \\
 x'_1 + x'_2 + x'_3 &= C'(\theta)
 \end{aligned}$$

Figure 11: The original undirected loop consisting of b_2 and b_3 is shown in the top panel, \mathcal{G}^o . The loop is broken at node (1), the unique node of this loop closest to the root (0). The branch and node variables associated with this loop have been relabeled with primes and a new node, (3'), has been introduced and marked as belonging to \mathcal{N}_M . The loop displacement constraint is thus introduced, forming digraph \mathcal{G}^* . Finally, the relations between the primed and original variables are shown (fourth panel).

B Tableau analysis with Mathematica

This appendix describes two Mathematica packages which can be used to perform tableau analysis on finger-like systems. The first package, `Tableau.m`, defines functions for describing and analyzing tableau equations. The second package, `Fingerlike.m`, provides an alternative interface to `Tableau.m` which simplifies the description and analysis of finger-like systems.

B.1 Tableau.m

The first step in performing a tableau analysis is specifying the network. This is done with the `BuildTableau` command:

```
BuildTableau[networkName,
             Nodes[list];
             branchName = Branch[from, to, rule];
             ...
            ]
```

The `Nodes` command specifies the list of nodes in the system. *The reference node must be the last node in the Nodes list.* The node names are used to specify branches.

The `Branch` statement assigns a rule to the branch and gives the branch a name for future reference. The name of a branch may be omitted. Branch rules have the form

$$\text{lhs} == \text{rhs}$$

that is, they are a Mathematica relationship. Within a rule, the following special variables are recognized:

<code>Displacement[branchName]</code>	displacement across a branch
<code>Force[branchName]</code>	force across a branch
<code>Displacement[]</code>	displacement across the current branch
<code>Force[]</code>	force across the current branch

Branch rules must be linear in the `Force` and `Displacement` variables. All other terms are considered to be independent inputs.

To solve for the branch displacements and forces in an actuator network, use the `SolveTableau` command:

```
{e, x, f} = SolveTableau[networkName]
```

where e , x and f are

e node displacements
 x branch displacements
 f branch forces

If the actuator network is not well-posed, an error message is printed.

B.2 Fingerlike.m

The `Tableau.m` package can be used to analyze mechanical tendon networks. Several special branch rules have been added to simplify the network building process. These rules are available via the `Fingerlike.m` package:

`Spring[from, to, K]` linear spring between two nodes
`Force[from, to, u]` force generator
`Displacement[from, to, u]` displacement generator

Force and displacement generators may be either dependent or independent. The argument u should be an expression (*not* a relation) which evaluates to the force in the branch.

`Fingerlike.m` also defines some commands for extracting information from the tableau solution that is useful in finger-like analysis:

`FingerlikeTorques[networkName, branchList, thetaList, {e,x,f}]`

Calculate the torques conjugate to `thetaList` as a function of the branch forces of `branchList`. `{e,x,f}` may be omitted, in which case it is recalculated.

`FingerlikeExtensions[networkName, branchList, {e,x,f}]`

Calculate the extensions across the listed branches. `{e,x,f}` is optional.

`FingerlikeMatrices[tauList, forceList]`

Calculate the matrices $G(\theta)$ and $S(\theta)$ that satisfy $\tau = S(\theta) + G(\theta)f$.

The optional `{e,x,f}` argument allows a previously stored solution of the tableau equations to be used.

B.3 Program listings

The program listings for the tableau analysis package are contained on the subsequent pages. In the listings the branch displacements and forces are referred to as v and i , respectively.

```

(*
 * Tableau.m - tableau analysis for linear dynamic circuits
 *
 * R. Murray and K. Pister
 * June 18, 1990
 *)

Needs["Struct"];

(* Make sure we don't overwrite the Null symbol (for debugging) *)
SetAttributes[Null, Protected];

(* Default options associated with a system *)
defaultOptions = {
  nodeList->{}, (* node names *)
  branchList->{}, (* branch names *)
  hingeRule->{}, (* rule for hinging together nodes *)
  incidenceMatrix->{}, (* incidence matrix *)
  gammaMatrix->{}, (* displacement coefficients *)
  lambdaMatrix->{}, (* force coefficients *)
  inputVector->{} (* input coefficients *)
};

(* Return the incidence matrix of a tableau system *)
IncidenceMatrix[name_Struct] := incidenceMatrix /. StructToRules[name];

(* SolveTableau - solve the tableau equations *)
SolveTableau[name_Struct] :=
Block[
  {sysopt = StructToRules[name], A,T,w, e,v,i},

  (* Get the reduced incidence matrix *)
  A = Drop[(incidenceMatrix /. sysopt), -1];

  (* Put together the Tableau matrix *)
  T = stackRows[
    stackCols[zero[nodeCount-1], zero[nodeCount-1, branchCount], A],
    stackCols[-Transpose[A], IdentityMatrix[branchCount], zero[branchCount]],
    stackCols[zero[branchCount, nodeCount-1],
      (gammaMatrix /. sysopt), (lambdaMatrix /. sysopt)] ];

  (* Now find the solution *)
  If[ Det[T] == 0,
    Print["SolveTableau: tableau equations are singular"];
    Return[Null] ];

  w = LinearSolve[T,
    Join[Table[0, {nodeCount+branchCount-1}], inputVector /. sysopt]];

  (* Partition the result and return it *)
  e = Take[w, nodeCount-1];
  v = Take[w, {nodeCount, nodeCount+branchCount-1}];
  i = Take[w, -branchCount];

  {e,v,i}
];

```

```

(*
 * BuildTableau - build a system up from statements
 *
 * Statements:
 *   Nodes           declare the nodes in the system (must be first)
 *   Branch          declare a linear branch rule
 *
 * Specific types of systems can be implemented by defining functions
 * which call Branch with a current/displacement relationship.
 *)

(* Variables used while we are building systems *)
system = Null; (* system being build *)
branchCount = 0; (* number of branches *)
nodeCount = 0; (* number of nodes *)
ruleCount = 0; (* current branch number *)

(* Rules for different passes of the compiler *)
buildRule := Hold[Set[name_, branch_[args_]]] :> Hold[branch[args, name]];
countRule := { buildRule, Hold[Nodes[list_]] :> Hold[Null] };

(* Function for building up a Tableau equation *)
SetAttributes[BuildTableau, HoldAll];
BuildTableau[name_Symbol, body_CompoundExpression] :=
  Set[name, BuildTableau[body]];

BuildTableau[body_CompoundExpression] :=
Block[
  {},

  (* Freeze the first level expressions so that we can evaluate at will *)
  holdBody = wrapHold[body];

  (* Initialize global variables *)
  system = RulesToStruct[defaultOptions];
  nodeCount = branchCount = 0;

  (* First pass - figure out how many branches there are *)
  Apply[Release, Hold[holdBody /. countRule]];
  branchCount = Length[branchList /. StructToRules[system]];

  (* Now go through and evaluate the original expression *)
  ruleCount = 0;
  Apply[Release, Hold[holdBody /. buildRule]];

  (* Replace the tableau matrix with its transpose *)
  SetStructValue[system,
    incidenceMatrix -> Transpose[incidenceMatrix /. StructToRules[system]] ];

  system
];

(* Store the list of nodes *)
Nodes[names_List] :=
Block[
  {},

  (* Check to make sure all of the names are symbols *)
  If[ Not[And @@ Map[SameQ[Head[#], Symbol]&, names]],
    Print["Nodes: invalid node list"];

```

```

Return[Null]];

SetStructValue[system, nodeList->names];
nodeCount = Length[names];
];

(* Hinge a bunch of nodes together *)
Hinge[names_List] :=
Block[
  {base = names[[1]],
   (*! This needs some error checking !*)

   (* Create a rule for node name replacement *)
   SetStructValue[system, hingeRule -> Map[Rule[#, base]&, Rest[names]]];
}

(* Store the rule associated with a branch *)
Branch[fromNode_, toNode_, equation_, branchName_:=Null] :=
Block[
  {sysopt = StructToRules[system], branchEqn, branchLabel = branchName,
   fromIndex, toIndex, arow, mrow, nrow, urow, hingeRule},

  (* Make sure we know how many nodes there are *)
  If[branchCount == 0,
   (* First pass; just store the rule name *)
   (* Don't use return because this boots us out of compound *)

   (* Check to make sure the branch name is valid *)
   If[branchLabel == Null, branchLabel = Unique["branch"] ];
   If[Not[SameQ[Head[branchLabel], Symbol]],
    Print["Branch: invalid branch name: ", branchLabel];
    Return[Null];
   ];

   (* Save the branch name in our list *)
   SetStructValue[system,
    branchList -> Append[branchList /. sysopt, branchLabel]],

  (* Else *)
  If[ nodeCount == 0,
   Print["Branch: missing node list"];
   Return[Null] ];

  (* See if there is a hinge rule *)
  hingeRule = hingeRule /. sysopt;

  (* Figure out the indices of the nodes *)
  fromIndex = listOffset[fromNode /. hingeRule, nodeList /. sysopt];
  toIndex = listOffset[toNode /. hingeRule, nodeList /. sysopt];
  If[fromIndex == 0 || toIndex == 0,
   Print["Branch: can't find node ", fromNode, " or node ", toNode];
   Return[Null]];

  (* Keep track of the fact that we got a new rule *)
  ++ruleCount;

  (* Add the entry to the incidence matrix *)
  arow = Table[0, {nodeCount}];
  arow[[fromIndex]] = -1; arow[[toIndex]] = 1;

```

```

(* Figure out the branch rule *)
If[(branchEqn= branchRule[equation, ruleCount]) == Null, Return[Null]];

(* Set the relevant options in the system options list *)
SetStructValues[system,
  incidenceMatrix -> Join[incidenceMatrix /. sysopt, {arow}],
  gammaMatrix -> Join[gammaMatrix /. sysopt, {branchEqn[[1]]}],
  lambdaMatrix -> Join[lambdaMatrix /. sysopt, {branchEqn[[2]]}],
  inputVector -> Append[inputVector /. sysopt, branchEqn[[3]] ]
];
];
];

(* Allow the use of Displacement[branch] and Force[branch] *)
SetAttributes[{Displacement, Force}, Listable];

(* Figure out the linear rule associated with a branch *)
branchRule[eqn_, offset_] :=
Block[
  {Mrow, Nrow, branches = branchList /. StructToRules[system], expr},

  (* First convert the equation into an expression *)
  If[Not[SameQ[Head[eqn], Equal]],
   Print["branchRule: rule is not of the form lhs == rhs"];
   Return[Null] ];

  (* Convert equation to expression and set up defaults *)
  expr = eqn[[1]] - eqn[[2]] /. {
   Force[] -> Force[branches[[offset]]],
   Displacement[] -> Displacement[branches[[offset]]] };

  (* Make sure we were given a linear equation *)
  If[(Max @@ Exponent[expr, Displacement[branches]]) > 1 ||
   (Max @@ Exponent[expr, Force[branches]]) > 1,
   Print["branchRule: rule is not linear"];
   Return[Null] ];

  Mrow = Coefficient[expr, Displacement[branches]];
  Nrow = Coefficient[expr, Force[branches]];
  {Mrow, Nrow, Mrow.Displacement[branches] + Nrow.Force[branches] - expr}
];

(*
 * Utility functions
 *
 * We collect here various utility functions that may not exist on
 * all systems.
 *)

SetAttributes[wrapHold, HoldAll]
wrapHold[expr_] := Map[ Hold, MapAt[Hold, Hold[expr], {1, 0}], {2}] [[1]]

(* Find position a single element in a list; return offset or 0 *)
listOffset[expr_, list_] :=
Block[
  {pos = Position[list, expr]},
  If[SameQ[pos, {}], 0, pos[[1,1]]]

```

```
];

(* Stack matrix columns together *)
stackCols[mats_] :=
Block[
  {i,j},
  Table[
    Join[ Flatten[Table[{mats}[[j]][[i]], {j,Length[{mats}]}], 1] ],
    {i, Length[ {mats}[[1]] ]}]
];

(* Stack matrix rows together *)
stackRows[mats_] := Join[Flatten[{mats}, 1]];

(* Matrix of zeros *)
zero[nr_, nc_] := Table[0, {nr}, {nc}];
zero[nr_] := zero[nr, nr];
```

```

(*)
* Fingerlike.m - fingerlike kinematics
*
* Richard M. Murray
* June 24, 1990
*
*)

Map[Needs, {"Tableau`", "Struct`", "Jac`"}];

(* Figure out the torque on the skeleton due to the actuator network *)
FingerlikeTorques(system_Struct, skeletalBranches_List, theta_List,
tableau_:=Null) :=
Block[
  {w = tableau, e,v,i, Pe, skeletalOffsets},

  (* First solve the Tableau equations (if necessary) *)
  If[ w == Null,
    If[ (w = SolveTableau[system]) == Null, Return[Null] ]];
  {e,v,i} = w;

  (* Now extract out the pieces that we need *)
  skeletalOffsets =
  Map[listOffset[#, branchList /. StructToRules[system]]&, skeletalBranches];
];

If[ Min @@ skeletalOffsets == 0,
  Print["FingerlikeTorques: can't find all skeletal branches"];
Return[Null] ];

(* Figure out the kinematics *)
(* For fingerlike systems Pe can be calculated from branch displacements *)
Pe = v[[skeletalOffsets]];
Transpose[Jac[Pe, theta]] . i[[skeletalOffsets]]
];

(* Figure out the muscle extensions on a given set of branches *)
FingerlikeExtensions(system_Struct, branchNames_List, tableau_:=Null) :=
Block[
  {w = tableau, e,v,i, branchOffsets},

  (* First solve the Tableau equations (if necessary) *)
  If[ w == Null,
    If[ (w = SolveTableau[system]) == Null, Return[Null] ]];
  {e,v,i} = w;

  (* Now extract out the pieces that we need *)
  branchOffsets =
  Map[listOffset[#, branchList /. StructToRules[system]]&, branchNames];
  If[ Min @@ branchOffsets == 0,
    Print["FingerlikeExtensions: can't find all branches"];
    Return[Null] ];

  (* Extract the relevant branch extensions *)
  v[[branchOffsets]]
];

(* Separate fingerlike torque into G and S *)
FingerlikeMatrices[tau_, f_] :=
Block[
  {expandTau, G, S},

```

```

  expandTau = Expand[tau];
  G = Map[Coefficient[#, f]&, expandTau];
  S = tau - G.f;
  {G, S}
];

(*
* Build Tableau statements for fingerlike systems
*
* Spring[from, to, K]          attach a spring between nodes
* Force[from, to, F]          independent/dependent force generator
* Displacement[from, to, D]    independent/dependent disp. generator
*)

Spring[from_, to_, K_, args_] :=
Branch[from, to, Force[] == K Displacement[], args];

Force[from_, to_, F_, args_] :=
Branch[from, to, Force[] == F, args];

Displacement[from_, to_, X_, args_] :=
Branch[from, to, Displacement[] == X, args];

(*
* Utility functions
*
* We collect here various utility functions that may not exist on
* all systems or are internal to other Packages.
*)

(* Find position a single element in a list; return offset or 0 *)
listOffset[expr_, list_] :=
Block[
  {pos = Position[list, expr]},
  If[SameQ[pos, {}], 0, pos[[1,1]]]
];

```

```
(*
 * Jac.m - jacobians and Lie derivatives
 *
 * John Hauser
 * 1989
 *)

BeginPackage["Jac`"]

Jac::usage = "Jac[f,x] computes the derivative of f with respect to x.";
Lie::usage = "Lie[f,g,x] computes the Lie bracket of f and g wrt x.";
LieD::usage = "LieD[f,h,x] compute the Lie derivative of h wrt f.";
Adj::usage = "Adj[v1,v2,x,k] calculate the kth bracket of v2 wrt v1.";

Begin["Private`"]

Jac[f_, x_] :=
  If[
    VectorQ[ f ],
    Table[ D[ f[[i]], x[[j]] ], {i, Length[f]}, {j, Length[x]} ],
    Table[ D[ f, x[[j]] ], {j, Length[x]} ]
  ]

Lie[v1_, v2_, x_] := Jac[v2,x].v1 - Jac[v1,x].v2

Adj[v1_, v2_, x_, k_] :=
  If[ k==0, v2, Lie[ v1, Adj[ v1, v2, x, k-1 ], x ] ]

LieD[f_, h_, x_] := Jac[h,x].f

End[]
EndPackage[]
```



```
(*
 * Struct - package for handling lists of name/value pairs
 *
 * RMM 8/8/90
 *
 * This package mimics the functionality of the Options interface in
 * a limited way. Namely, it maintains variables which are lists of
 * rules and provides functions for changing the values of the rules.
 * Internally, rules are stored in a list of pairs format.
 *
 * I wanted to use Attributes instead of Struct/Member, but the name was
 * already taken.
 *)

BeginPackage["Struct`"]

(* Convert a list of rules to a list of pairs *)
RulesToStruct[rules_List] := Struct @@ Map[ruleToPair, rules];
ruleToPair[Literal[Rule[name_, value_, hidden___]]] := {name, value};

(* Convert a list of pairs into a list of rules *)
StructToRules[list_Struct] := List @@ Map[Rule @@ #&, list];

(* Set the value of an element in a structure *)
SetAttributes[SetStructValue, HoldFirst]
SetStructValue[list_Symbol, Literal[Rule[name_, value_, hidden___]]] :=
  list[[ findName[list, name], 2 ]] = value;

SetAttributes[SetStructValues, HoldFirst]
SetStructValues[name_Symbol, rules___] :=
  Map[SetStructValue[name, #]&, {rules}]

findName[list_, name_] := Position[list, name][[1,1]]

EndPackage[]
```

C Mathematica Listings

This section contains the Mathematica listings used to analyze the examples presented in the body of the paper. These files use the tableau analysis package described in Appendix B.

C.1 example1.m

```
(*
 * example1.m - Mathematica solution to example #1
 *
 * RMM 31 May 91
 *
 *)

<<Fingerlike.m

(* Tendon displacement functions *)
P1[theta_] := R1 theta + L1
P2[theta_] := -(R2 + R2p theta) theta + L2

BuildTableau[ex1,
  Nodes[{n1,n2,n3,n4,n5,base}];

  (* Branch laws for upper tendon *)
  b1 = Displacement[base, n1, P1[theta]];
  b2 = Force[base, n3, F1];
  b3 = Spring[n3, n4, K];
  b4 = Displacement[n4, n1, L1];

  (* Branch laws for lower (non-compliant) tendon *)
  b5 = Force[base, n5, F2];
  b6 = Displacement[n5, n2, L2];
  b7 = Displacement[base, n2, P2[theta]];
];

A = IncidenceMatrix[ex1]; (* get the incidence matrix *)
w = SolveTableau[ex1]; (* solve the tableau equations *)

tau = FingerlikeTorques[ex1, {b1,b7}, {theta}];
```

C.2 example2.m

```

(*)
* example2.m - Mathematica solution to example #2
*
* RMM 31 May 91
*
* This system is similar to that of example #1, except that nodes 1 and
* 2 are combined into a single node (n1). As a consequence, a loop is
* generated and an extra node must be added (a1).
*
*)

<<Fingerlike.m

(* Tendon displacement functions *)
P1[theta_] := R1 theta + L1
P2[theta_] := -(R2 + R2p theta) theta + L2

BuildTableau[ex2,
  Nodes[{n1,a1,n3,n4,n5,base}];

  (* Branch laws for upper tendon *)
  b1 = Displacement[base, n1, P1[theta]];
  b2 = Force[base, n3, F1];
  b3 = Spring[n3, n4, K];
  b4 = Displacement[n4, n1, L1];

  (* Branch laws for lower (non-compliant) tendon *)
  b5 = Force[base, n5, F2];
  b6 = Displacement[n5, a1, L2];
  b7 = Displacement[a1, n1, L1 - L2 + P1[theta] - P2[theta]];
];

A = IncidenceMatrix[ex2]; (* get the incidence matrix *)
w = SolveTableau[ex2]; (* solve the tableau equations *)

tau = FingerlikeTorques[ex2, {b1,b7}, {theta}];

```

C.3 example3.m

```
(*
 * example3.m - Mathematica solution for example #3
 *
 * RMM 18 Feb 91
 *
 * This system contains an example of branch laws which depend on
 * displacements at other locations in the network.
 *)

<<Fingerlike.m

BuildTableau[ex3,
  Nodes[{n1,n2,n3,n0}];

  b1 = Force[n0, n1, F1];
  b2 = Force[n0, n1, g12 Displacement[b5]];
  b3 = Spring[n1, n3, K1];
  b4 = Displacement[n0, n3, theta];

  b5 = Force[n2, n0, F2];
  b6 = Force[n2, n0, g21 Displacement[b1]];
  b7 = Spring[n3, n2, K2];
]

A = IncidenceMatrix[ex3]; (* get the incidence matrix *)
w = SolveTableau[ex3]; (* solve the tableau equations *)

tau = Simplify[ FingerlikeTorques[ex3, {b4}, {theta}] ];
{G, S} = FingerlikeMatrices[tau, {F1,F2}];
```

C.4 buchner.m

```

(*)
* buchner.m - Mathematica solution to Buchner finger example
*
* RMM 31 May 91
*
* For simplicity, we assign all tendons zero length. Since we are
* interested only in differential relationships, this eliminates
* extra constants.
*)

<<fingerli.m

(* Define the displacement functions used below *)
P1 = -R11 phi1 - (R12 - R12p phi2) phi2 - R13 phi3; (* ED *)
P2 = (R21 + R21p phi1) phi1 + (R22 + R22p phi2) phi2 + (* FDP *)
    (R23 + R23p phi3) phi3;
P3 = (R31 + R31p phi1) phi1 + R32 phi2; (* FDS *)
P4 = R41 phi1 - R42 phi2; (* Int *)

P5 = -(R51 phi1 - (R52 + R52p phi2) phi2 + R11 phi1 + (* Lum-ED *)
    (R12 - R12p phi2) phi2);
P6 = -(-R11 phi1 - R12 phi2 - R41 phi1 + R42 phi2);

BuildTableau[buchner,
  Nodes[{n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,n11,n12,n13,n14,n0}];

  (* Muscle elements *)
  b1 = Force[n0, n12, F1]; (* extensor digitorum *)
  b2 = Force[n0, n10, F2]; (* flexor digitorum profundus *)
  b3 = Force[n0, n9, F3]; (* flexor digitorum superficialis *)
  b4 = Force[n0, n8, F4]; (* palmar and dorsal interosseus *)
  b5 = Force[n10, n11, F5]; (* lumbricals *)

  b6 = Displacement[n5, n1, 0]; (* ED tendon segments *)
  b7 = Displacement[n6, n5, 0];
  b8 = Spring[n12, n6, Ka]; (* ED spring divider *)
  b9 = Spring[n12, n7, Kb];
  b10 = Displacement[n13, n4, 0];

  b11 = Displacement[n8, n4, 0]; (* Int tendon segment *)
  b12 = Displacement[n9, n3, 0]; (* FDS tendon segment *)
  b13 = Displacement[n14, n5, 0]; (* Lum tendon segment *)
  b14 = Displacement[n10, n2, 0]; (* FDP tendon segment *)

  (* Displacement generators across mechanical interface nodes *)
  b15 = Displacement[n0, n1, P1]; (* ED *)

```

```
b16 = Displacement[n0, n4, P4]; (* Int *)
b17 = Displacement[n0, n2, P2]; (* FDP *)
b18 = Displacement[n0, n3, P3]; (* FDS *)

(* Displacement generators due to actuator loops *)
b19 = Displacement[n11, n14, P5]; (* Lum-ED loop *)
b20 = Displacement[n7, n13, P6]; (* Int-ED loop *)
];

(* Create lists of useful quantities *)
phi = {phi1, phi2, phi3}; (* configuration variables *)
F = {F1, F2, F3, F4, F5}; (* input forces *)
ActuatorBranches = {b15, b16, b17, b18, b19, b20};

(*
A = IncidenceMatrix[buchner]; (* get the incidence matrix *)

(* This calculation takes a *long* time *)
tau = FingerlikeTorques[buchner, ActuatorBranches, phi];
{G, S} = FingerlikeMatrices[tau, F];

(* Save the results so we don't have to rerun this section *)
Save["buchner.out", A, tau, G, S]
*)
```

References

- [1] J. C. Becker, N. V. Thakor, and K. G. Gruben. A study of human hand tendon kinematics with applications to robot hand design. In *IEEE International Conference on Robotics and Automation*, pages 1540–1545, 1986.
- [2] H. J. Buchner, M. J. Hines, and H. Hemami. A dynamic model for finger interphalangeal coordination. *J. Biomechanics*, 21:459–468, 1988.
- [3] R. G. Busacker and T. L. Saaty. *Finite graphs and networks: An introduction with applications*. McGraw-Hill, 1965.
- [4] T. J. Carew. The control of reflex action. In Eric R. Kandel and James H. Schwartz, editors, *Principles of Neural Science, 2nd ed.*, chapter 35, pages 457–468. Elsevier, 1985.
- [5] L. O. Chua, C. A. Desoer, and E. S. Kuh. *Linear and Nonlinear Circuits*. McGraw-Hill, 1987.
- [6] W. K. Durfee, M. B. Wall, D. Rowell, and F. K. Abbott. Interactive software for dynamic system modeling using linear graphs. *IEEE Control Systems*, 11(4):60–66, 1991.
- [7] S. C. Jacobsen, H. Ko, E. K. Iversen, and C. C. Davis. Antagonistic control of a tendon driven manipulator. In *IEEE International Conference on Robotics and Automation*, pages 1334–1339, 1989.
- [8] D. Karnopp and R. Rosenberg. *System Dynamics: A Unified Approach*. Wiley, New York, 1975.
- [9] A. J. Pellionisz and R. Llinàs. Tensor network theory of the metaorganization of functional organization of functional geometries in the cns. *Neuroscience*, 16:245–274, 1985.
- [10] L. P. A. Robichaud, M. Boisvert, and J. Robert. *Signal Flow Graphs and Applications*. Prentice-Hall, 1962.
- [11] J.L. Shearer, A.T. Murphy, and H.H Richardson. *Introduction to System Dynamics*. MIT Press, Mass., 1974.
- [12] D. E. Thompson and D. J. Giurintano. A kinematic model of the flexor tendons of the hand. *J. Biomechanics*, 22:327–334, 1989.

-
- [13] L-W. Tsai and J-J. Lee. Kinematic analysis of tendon-driven robotic mechanisms using graph theory. *Journal Mechanisms, Transmissions, and Automation in Design*, 111:59–65, 1989.
- [14] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley, 1988.

The first and second authors may be reached via electronic mail at the following addresses:

dcdeno@united.berkeley.edu
murray@design.caltech.edu