ADAPTIVE CENTRAL PATTERN GENERATORS FOR CONTROL OF TENSEGRITY SPINES WITH MANY DEGREES OF FREEDOM

by

BRIAN TIETZ MIRLETZ

Submitted in partial fulfillment of the requirements

for the degree of Doctor of Philosophy

Department of Mechanical and Aerospace Engineering

CASE WESTERN RESERVE UNIVERSITY

January, 2016

CASE WESTERN RESERVE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby approve the dissertation of

Brian Tietz Mirletz

candidate for the degree of **Doctor of Philosophy***.

Committee Chair

Dr. Roger D. Quinn

Committee Member

Dr. Kiju Lee

Committee Member

Dr. Joseph M. Prahl

Committee Member

Dr. M. Cenk Çavuşoğlu

Date of Defense

July 13, 2015

*We also certify that written approval has been obtained

for any proprietary material contained therein.

Contents

List of Tables

List of Figures

Acknowledgments

This work has been inspired, encouraged, and supported by countless friends, colleagues and family members over the years. I would like to start by recognizing Vytas SunSpiral, without whom I may not even know the word tensegrity. Vytas' vision and energy have been driving forces behind this project. As my advisor, Prof. Roger Quinn has done an excellent job helping me navigate through changes in project scope, coursework, fellowship applications, and the publication process. My committee members, Profs. Kiju Lee, Joseph M. Mansour, Joseph M. Prahl and M. Cenk Çavuşoğlu have all provided excellent advice. Prof. Roy Ritzmann made valuable suggestions on early iterations of this work, including the dissertation proposal.

Dr. In-Won Park constructed the final version the robot hardware and gathered the hardware data for the validation tests. Ross Carnahan made significant contributions to the carbon fiber Tetraspine, including the mold. Dr. Rich Bachmann and Mark Nandor also deserve thanks for coaching us through the carbon fiber process. Penny Lemire and Heather Mirletz both taught me the fundamentals required for Tetraspine's the knit stretch sensors. Dr. Adrian Agogino made countless helpful suggestions on machine learning techniques and ways to pitch the research. I greatly appreciate Tom Flemons' willingness to share his cutting edge models, and many fruitful conversations on tensegrity modeling.

Ryan Adams and Perry Bhandal both made numerous contributions to NTRT, Ryan developed the simulation architecture and the low level software that translates

the models into objects within Bullet, and Perry integrated critical libraries, and drove development of the multi-process learning scripts. I would like to thank Dr. Lee Brownston for teaching me most, if not all, of my good habits in C++ software development. Dr. Atil Iscen's machine learning code was an excellent starting point for my own learning routines. Dr. Dorothea Blostein, Jonathan Bruce, Dr. Ken Caluwaerts, Jérémie Despraz, Jeffrey Friesen, Dawn Hustig-Schultz, Andrew Sabelhaus, Steven Lessard, Alexander Xydes have all been valuable collaborators on NTRT and within the Dynamic Tensegrity Robotics Lab at NASA Ames Research Center. The Intelligent Robotics Group and Office of the Chief Technologist at NASA Ames helped make all of my trips there very productive and enjoyable, special thanks go to Dr. Terry Fong and Ingrid Desilvestre.

Back in Cleveland, thanks go to Dr. Kathryn Daltorio, Andrew Horchler, Alex Hunt, Matt Klein, Ron Leibach, Alex Lonsberry, Nick Szczecinski, Vickie Webster, and the rest of the Biologically Inspired Robotics lab for assistance with everything from discussing research, to printing posters, to computer maintenance, to stress relief.

Additional thanks to Dr. Roger Quinn, Dr. Kathryn Daltorio, Vytas SunSpiral, Dawn Hustig-Schultz, and Heather Mirletz for their comments on early drafts.

This work was directly supported by a NASA Space Technology Research Fellowship, #NNX11AN15H, and indirectly supported via the NASA Innovative Advanced Concepts Program.

Finally this would not have been possible without my parents, Doug and Pat Tietz encouraging me to study robotics, and all of their support throughout all three degrees at Case. Love goes to Heather Mirletz for her optimism, encouragement, and splendid diction.

Adaptive Central Pattern Generators for Control of Tensegrity Spines with Many Degrees of Freedom

Abstract

by

BRIAN TIETZ MIRLETZ

This work seeks to advance understanding of how to construct and control tensegrity spines: highly compliant robots with many degrees of freedom inspired by biological spines. Tensegrity describes systems of rigid bodies stabilized by cables, where on a simulated twelve segment robot up to 72 degrees of freedom require constraints or control. To coordinate the high number of actuators, central pattern generator (CPG) based controllers provide goal directed, adaptive trajectories for movement on rough terrain. As a result, this work details, to the author's knowledge, the first CPG based robotic control capable of goal directed locomotion on rough terrain. This control scheme is implemented on four different tensegrity spine designs.

Prior to this work, tensegrity spines were passive, hand constructed models; the open source NASA Tensegrity Robotics Toolkit (NTRT) was created to develop robots inspired by these and other tensegrity models using physics based simulation. This work also discusses the development and implementation within NTRT of a cable model with contact dynamics and realistic forces. Finally, the simulations are validated against the forces measured in the hardware implementation of a three segment tensegrity spine robot.

Chapter 1

Introduction

Many desirable applications of robotics, such as search and rescue, planetary exploration, and environments dangerous to humans are still largely inaccessible to current robots. Even though robots operate quite effectively in structured, largely predictable environments such as factories, single floor residences, and missions with detailed guidance from a human operator, robots have difficulty locomoting through unstructured terrain or with low bandwidth commands. In order to complete these more difficult missions, robots need to actively adapt to their surroundings, utilizing compliant yet rugged bodies to execute those adaptations.

1.1 Motivation

Anyone with sufficient experience in or exposure to dance, sports, or martial arts has likely encountered the phrase "use your core" or "move from your center." This is always a great mystery to beginners, who are still very focused on where to put their hands and feet. The answer was finally revealed to me about eight years into learning the swing dance Lindy Hop, when an instructor pointed at the bottom of his rib cage and said "your legs start here." Similarly, later in the lesson, the other instructor pointed at her sternum and said "your arms start here." In more biomechanical terms,

using flexibility of the torso is necessary for efficient, powerful, agile, and aesthetically pleasing movement.

This is an important anecdote to remember as we develop mobile robots for increasingly complex and dynamic tasks. The recent DARPA Robotics Challenge finals highlighted the difficulty of movement with a large, rigid torso [Guizzo and Ackerman, 2015]. Out of twenty three teams attempting to complete eight tasks in a simulated disaster scenario, most robots either fell or required human assistance at some point, and only one of the robots was able to recover from a fall without human assistance. Among quadruped robots, even the impressive cheetah inspired robot from MIT reaches some concerning pitch angles as it jumps hurdles [Park et al., 2015]. Additionally, while some robots can approach the efficiency of their biological counterparts on laboratory floors and treadmills, the most capable robots are orders of magnitude less efficient [Ruina, 2011].

Biological observations highlight how a flexible, actuated spine could improve a robot's performance in multiple ways. In evolutionarily 'lower' tetrapods such as lizards and crocodiles, the lateral bending (in the coronal plane) of the skeleton produces the advancement of the limbs for locomotion [Gracovetsky, 1985]. Mammals also have the ability to bend vertically (in the sagittal plane), which enables higher speed gaits such as galloping [Schilling and Hackert, 2006]. Galloping in particular utilizes the elasticity of the spine to transfer energy between legs [Alexander, 1988]. In addition to motion and efficiency, dorsal muscles of a dog's spine appear to contribute to stabilization in all three planes of the body, with roles (mobility and stability) changing somewhat between gaits [Schilling and Carrier, 2010].

Such improvements in agility and efficiency could enable robots to be more successful in applications where it is dangerous, difficult, or undesirable for humans to venture, such as search and rescue and planetary exploration. Constructing a robotic spine with a series of rigid bodies connected by rotary motors may not capture the

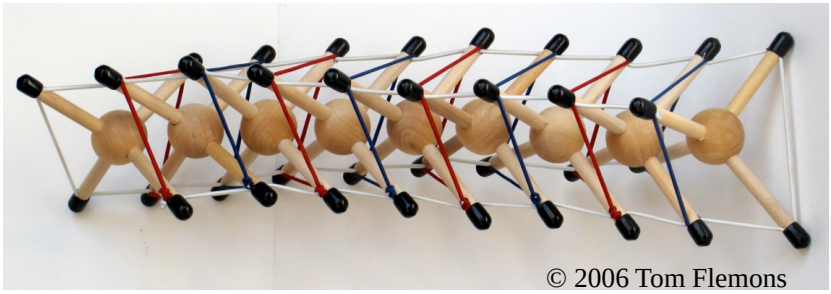

Figure 1.1: A tensegrity spine model using a tetrahedral complex as vertebrae. Eight strings connect each segment, four of which form a tensegrity saddle joint. Image courtesy of Tom Flemons, originally from [Flemons, 2007].

desired features of energy storage, be too heavy, or too difficult to control. The energy storage and control problems may be solved by increasing compliance in the entire system, allowing for the robot's body to accommodate low level details of movement [Paul, 2006], but a fully soft system may not have the strength to transmit the desired forces. Tensegrity structures appear to have the appropriate strength to weight ratio and force transmission capabilities [Skelton et al., 2001]. Tensegrity is a concept from art and architecture referring to systems stabilized with continuous tension networks [Snelson, 1965], where all of the rigid elements are in compression and forces are transmitted through elements in tension, such as cables. This can be used to model biomechanical systems on multiple scales [Ingber et al., 2014, Scarr, 2014]. A properly designed and controlled tensegrity spine may provide robots with many of the desired properties and behaviors of biological spines.

1.2 Scope of this dissertation

For the purposes of this dissertation, a tensegrity spine is defined as a tensegrity structure, built primarily along a single axis, which is capable of a stable, compliant response to forces in any direction. Prior to this work, tensegrity spines were passive models constructed from wood and elastic strings [Flemons, 2007]. An example is shown in Figure 1.1, with twelve segments composed of tetrahedral complexes, and

eight strings between segments.

There are three main possibilities for a tensegrity spine in robotics:

1. A 'spinal engine', aiding in the dynamic locomotion of a legged robot
2. A passive 'force integrator', replacing the rigid torso of a quadrupedal or bipedal robot
3. A stand-alone snake or legless lizard like robot

Each of the above shares common issues in mechanism design and control, but the stand alone robot is the most tractable given the state of prior art. The process of developing a stand alone spine will create the tools required to evaluate dynamic tensegrity spine designs and their controllers for the eventual application to legged robots. Therefore this dissertation will focus on the stand alone robot.

The next question is: what sort of controller should be used with a tensegrity spine? To enable command of a robot with appropriately high level tasks, it is logical to incorporate reflexes and online adaptations into the locomotor control. Traditional modeling control leaves too many possibilities for the number of degrees of freedom in a tensegrity spine for task level control [Rovira and Mirats Tur, 2009]. Central pattern generators (CPGs) are a biologically inspired, distributed control system that can accommodate low bandwidth commands from high level controllers, and reflexively adapt to feedback [Ijspeert, 2008]. This dissertation will explore using CPGs for adaptive control of tensegrity spines.

The purpose of this work is to develop adaptive controllers and simulation tools for tensegrity spines with many degrees of freedom. The controllers need to be capable of goal directed locomotion over rough terrain, with minimal descending commands. The simulation tools need to represent the real world sufficiently accurately to inform the design of future robots and controllers.

1.3 My Involvement and Contributions

I first encountered the concept of tensegrity in 2011 when I was introduced to Vytas SunSpiral through my NASA Space Technology Research Fellowship. Our initial goal was to control a spine or snake-like tensegrity robot with central pattern generators (CPGs), based on the intuition that these two oscillatory systems would synchronize. The compliance of its tensegrity structure should give the robot a high degree of locomotion competence at the structural level, which would complement the adaptability of the central pattern generator based controller.

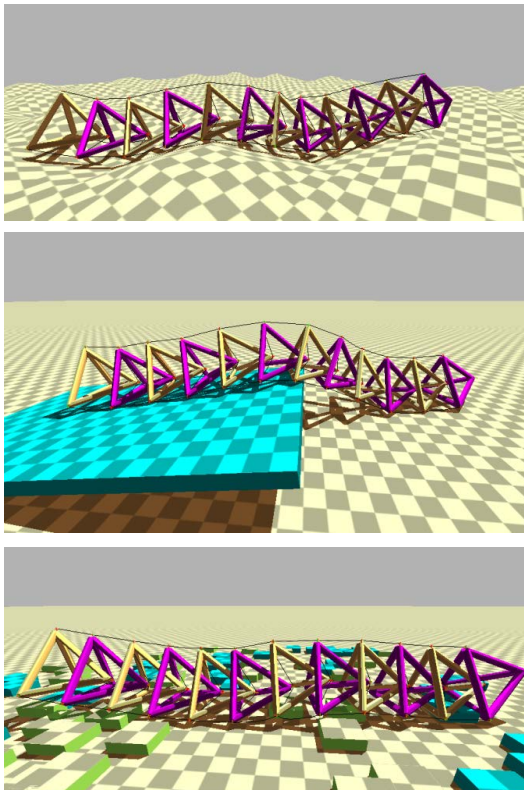

Figure 1.2: Tetraspine moving across: Top: 4 cm hills, Middle: A 7.5 degree ramp, Bottom 1 cm blocks. This simulation of Tetraspine used 10 cm rods in the tetrahedrons.

Since analytical design methods for tensegrity robots were not (and still are not) well defined, we decided to test the idea in a preliminary version of the NASA Tensegrity Robotics Toolkit (NTRT, Chapter 3). I started with simple sine waves providing

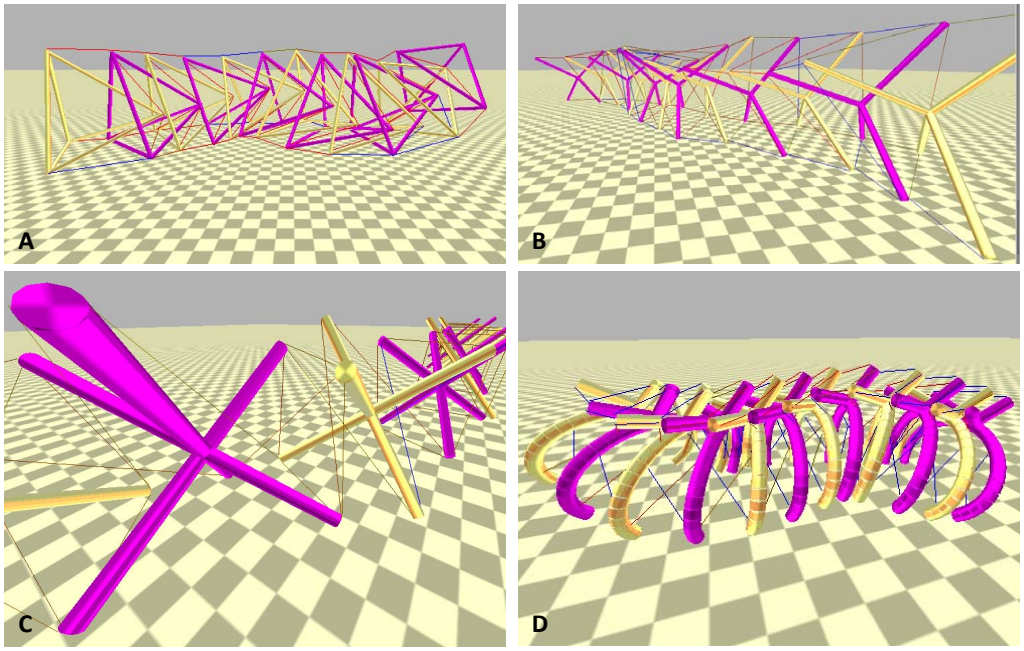

Figure 1.3: Four different tensegrity spines simulated in the NASA Tensegrity Robotics Toolkit. Locomotor gaits were tuned by machine learning for all structures, and all demonstrated multiple gaits. A: Tetraspine, with tetrahedral segments and six cables between each segment. B: Tetrahedal complexes for segments, with eight cables connecting each segment. C: Octahedral complexes for segments, with four passive and four active cables between segments. D: A spine with rigidly connected ribs connected by seven active cables and four passive cables.

a trajectory to distributed impedance controllers for the morphology that became Tetraspine, and eventually moved to an open loop CPG (Chapter 6). In simulation, movement was achieved with suspicious alacrity on rough terrain, including hills, blocks, a ramp, and a wall as shown in Figure 1.2. Due to the success in simulation we decided to pursue hardware in parallel, with one version of Tetraspine at CWRU and another version at NASA Ames. Ultimately the hardware was capable of locomotion, but not of the terrain handling we observed in simulation due to the spine's shorter length.

With results from our first hardware attempt, we were able to make improvements to NTRT including replacing the default position-based cables from the Bullet physics engine with a custom Hooke's law based spring model, resulting in more realistic though less impressive simulations. Based on Tetraspine's performance under these

new conditions, we concluded it would be worth investigating additional morphologies. Tom Flemons, a tensegrity researcher from British Columbia, has developed a series of passive tensegrity models inspired by vertebrate anatomy [Flemons, 2007, Flemons, 2012]. Chapter 5 shows the NTRT implementation of these spines, with the addition of a novel tensegrity spine I created. With four different morphologies, it would have been too time consuming to hand tune controllers for each experimental morphology. Thus, I developed an algorithm for automatically determining a CPG controller based on the morphology of a tensegrity structure, assuming reasonable symmetries between the actuators (Section 6.3). We then adapted machine learning techniques to parameterize these controllers, detailed in Chapter 4.1. I applied this technique to four morphologies controlling between forty four and eighty eight actuators with up to seventy two degrees of freedom, a record high for a tensegrity robot.

In addition to contributing to NTRT's open source release, I then added contact dynamics to our custom cable models. To my knowledge, this is currently the only open source cable model with both realistic forces and contact dynamics. This enabled reinvestigation of rough terrain locomotion with improved realism. As expected, the success of the tensegrity spines depended on the ratio between the size of the obstacle and the size of the spine. In order to enable spines to handle larger obstacles, I adapted feedback equations for a central pattern generator based controller in a novel way. I added descending inputs via an artificial neural network for goal direction, a step towards allowing for simple 'go that way' commands from an operator. The full control scheme is discussed in Chapter 6.

In order to verify the realism of these results, Dr. In-Won Park and I used the NASA Ames Tetraspine hardware to verify the forces predicted by NTRT. We showed that NTRT is capable of predicting the maximum tension in the hardware within 7.9%, giving us confidence in our ability to use NTRT to design future robots. Both

the NASA Ames and CWRU hardware, and the validation work are presented in Chapter 7. Finally, Chapter 8 includes my ideas for future directions for tensegrity spines, and other applications of tensegrity research that could utilize this work's contributions to control algorithms and simulation software for tensegrity robots.

Chapter 2

Background and Related Work

2.1 Structures

Having had the pleasure of working with a team that includes mechanical engineers, computer scientists, and tensegrity designers, I have learned the value of common technical vocabulary, thus a brief robotics background is included here.

The 'structure' is the robot's body, and how it interacts with the world. Robots have a wide range of structures, from a 'shoebox on wheels', to an arm, to a humanoid, to a variety of soft structures. Structures are typically simplified as rigid elements (sometimes referred to as links), and actuators (motors and similar).

Degrees of freedom (DOF) is an important concept to help describe dynamic structures such as robots. DOF refers to the number of variables required to describe the robot's current configuration, such as its position in space or the orientation of its links. The number of degrees of freedom is closely related to the number of actuators and constraints needed to control the robot. Position in three dimensions can be described in three variables (x, y, z). If the object has any shape (it is not a point), an additional three variables are required to describe its orientation (yaw, pitch, roll). Thus, an unconstrained rigid body has six degrees of freedom. Constraints can

reduce the DOF, for example a wheeled vehicle on flat ground is often described in two dimensions, with three degrees of freedom (x, y, and yaw). A structure fixed (with bolts or welds, etc.) to a base has zero degrees of freedom. A hinge joint provides a single rotational degree of freedom, a prismatic joint one linear degree, and a ball joint three rotational. A robotic arm with a chain of rigid bodies, fixed at the base, would have degrees of freedom exactly equal to its number of actuators. The tensegrity systems considered here have up to seventy two degrees of freedom, with between forty four and eighty eight actuators. Depending on the morphology, the motion of these degrees of freedom can be actively actuated or passively constrained.

2.1.1 Robotic Spines

While many existing robots are capable of versatile motion on rough terrain with a rigid torso [Raibert et al., 2008], other robots have gained mobility advantages by including a few degrees of freedom in the torso. Applications of body joints tend to fall into two broad categories. First, body joints can be actuated as needed to enhance behaviors like climbing [Allen et al., 2003, Daltorio et al., 2008]. Passive degrees of freedom can also provide additional adaptations. By tuning the stiffness of their robot's cable driven spine, Takuma et al. were able to optimize the speed of their quadruped robot [Takuma et al., 2010]. The caterpillar whegs robot had a spine composed of rubber spacers which increased passive compliance [Kern, 2012]. Servo driven cables provided turning and rearing. The next iteration of the design utilized a tensegrity joint design between segments (a compression spring between rigid elements, with the actuated cables as tensile members), and actuated each segment individually [Webster et al., 2013]. These designs demonstrate the ability of robots with body joints to adapt to their environment with minimal computation, but these types of joints play a limited role.

Alternatively, body joints can drive locomotion. Snake robots use a serial

chain of motors to generate whole body locomotion directly [Wright et al., 2007, Wright et al., 2012], or with passive wheels [Mori and Hirose, 2002]. Controllers for these robots mimic the gaits seen in biological snakes [Transeth et al., 2009]. Ijspeert et al.'s salamander robot uses a chain of motors in swimming and walking with biologically inspired central pattern generator based control [Ijspeert et al., 2007].

Quadruped robots have also been driven by a spine, increasing the range of motion of legs for increased speed and agility [Leeser, 1996]. Zhao et al. examined a planar quadruped performing a bounding gait on flat ground with a rigid, passive, and actuated spine, and found that the actuated spine increased speed over the other two spines [Zhao et al., 2013]. Eckert et al. experimented with actuated spines with multiple segments that move in the sagittal plane with at most one actuator, and found that a compliant spine can produce stable gaits up to 0.75 m/s on a 0.224 meter long robot [Eckert et al., 2015]. Duperret et al. showed a parallel elastic spine improves the agility of leaping in the quadruped Canid, when compared to the similar XRL robot with a rigid body [Duperret et al., 2014]. Spines developed for humanoid robots have so far focused on motion and stabilization of the trunk using cable driven [Mizuuchi et al., 2005, Liu et al., 2009, Pfeifer et al., 2013] or hydraulic mechanisms [Roos et al., 2006], but have not been used while walking.

This work seeks to combine compliant adaptability with active spine motion by using fully actuated, spine-like tensegrity structures.

2.2 Tensegrity Structures

A tensegrity structure is most generally defined as a set of rigid bodies (compression elements), stabilized by tension elements (typically cables or strings), in the absence of external forces [Skelton and De Oliveira, 2009]. A simple simulated tensegrity with three bars and nine cables in a prism configuration is shown in Figure 2.1. Initial work

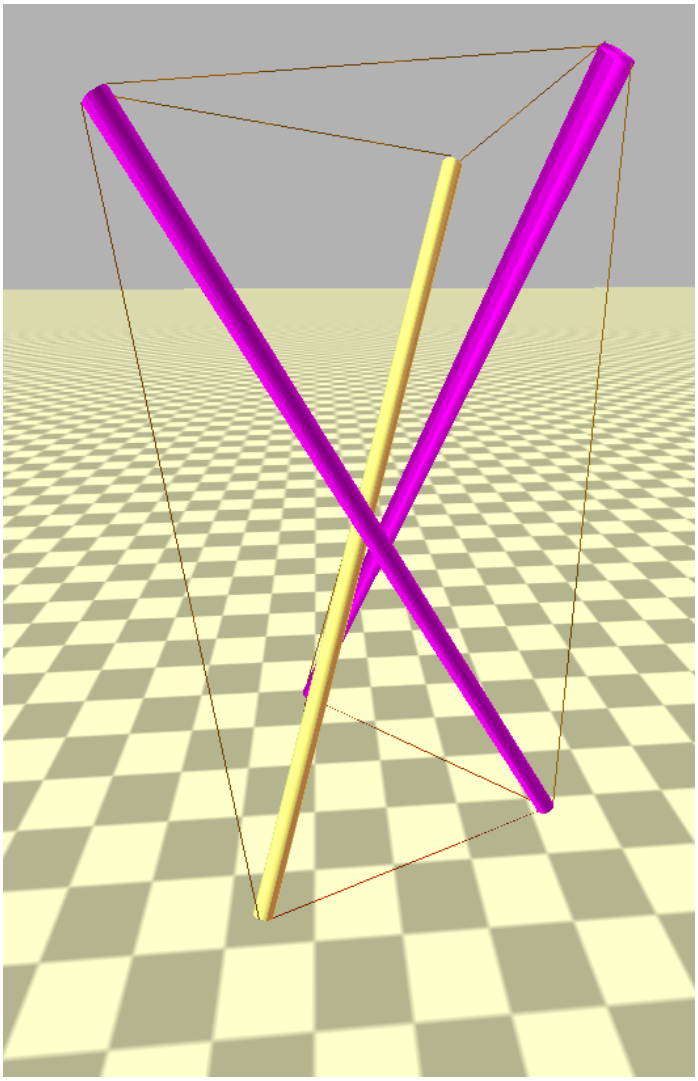

Figure 2.1: The simplest three dimensional tensegrity structure, a three bar prism, simulated in NTRT. The purple and yellow cylinders represent bars and red lines represent the cables.

on tensegrity structures was performed by Kenneth Snelson and Buckminster Fuller in the late 1940's, with patents filed in 1959 and 1960 [Fuller, 1962, Snelson, 1965]. Fuller went on to apply the work to large structures such as geodesic domes [Fuller, 1965], while Snelson created artistic sculptures [Heartney, 2009]. Contemporary architects have used tensegrity structures in their designs for applications ranging from bridges [Franklin et al., 2010] to football stadiums [Levy, 1991]. Current research in adaptive civil structures is examining tensegrity structures for improved building functionality or damage control [Sterk, 2006, Korkmaz et al., 2011].

With three rigid bodies, the tensegrity prism in Figure 2.1 has 18 degrees of freedom. While the cables do constrain the motion, the constraints are not rigid in the same way as a hinge or prismatic joint. In order to balance the forces and keep the structure's shape, the cables are elastic, meaning they will change their length when a force is applied. While in an absolute sense the rods also have some elasticity, they are orders of magnitude stiffer than the cables, so rigidity is usually a safe assumption. The cables' elasticity is assumed to be similar to that of a linear spring, where the force is linearly proportional to the distance away from its 'rest length.' A cable's elasticity is usually represented as $F = k\Delta x$, where F is the force, k is the spring's stiffness, and Δx is the distance from the rest length. However, unlike a spring, cables cannot be pushed (compressed). A slack cable applies no force (beyond its usually negligible weight), so if $\Delta x \leq 0$, $F = 0$. A more rigorous set of equations for computer simulation of tensegrity structures is discussed in Section 3.2. With their composition of rigid elements and cables, tensegrity robots are often considered a hybrid of soft and rigid robots [Lipson, 2014]. Constraining the rigid bodies with cables instead of rigid links provides desirable properties including:

- Most of a tensegrity structure is empty space, so tensegrity can provide minimal mass for a structure of a given strength, also known as a high strength to weight ratio [Skelton et al., 2001]

- Forces applied to the structure will be passively distributed throughout all of the tension elements, minimizing points of local weakness [Skelton and De Oliveira, 2009, SunSpiral et al., 2013]
- If one or more actuators fails, others can still move the structure in useful ways [Iscen et al., 2014, Paul et al., 2006]
- The tension elements store elastic energy within the structure [Juan and Tur, 2008]

2.2.1 Tensegrity in Biology

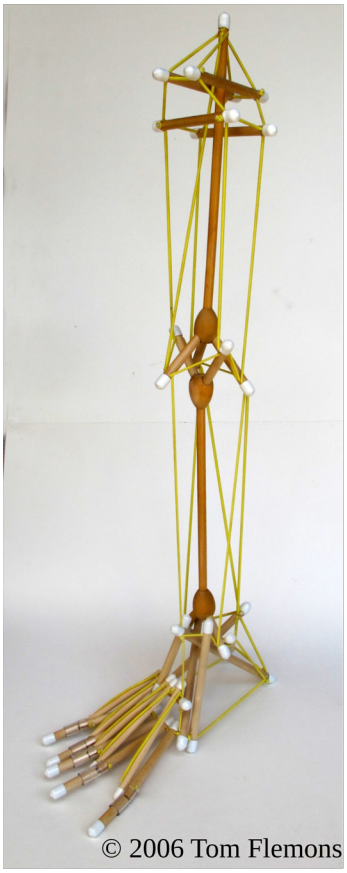

Figure 2.2: A tensegrity model of a human leg, which is able to support its own weight and transmit forces through the knee into the foot. Image courtesy of Tom Flemons, originally from [Flemons, 2012].

Many of the desirable properties of tensegrity structures are already present in biological systems. For exampe, the non-linear mechanical properties of the cell's cytoskeleton such as strain hardening (strengthening under sufficient stretch) can be modeled with tensegrity [Ingber et al., 2014]. These insights have implications for practical applications in tissue engineering, such as determining how cells will attach to substrates [Wang et al., 2002], and even changing the genetic expression of the cell (known as cell fate) [Ingber, 2003]. Tensegrity properties can also be observed in macroscopic biological structures, particularly through the connective tissue such as fascia [Myers et al., 2001]. Turvey and Fonseca have hypothesized that tensegrity based biomechanics would explain phenomena in haptic perception (forces experienced from touch) such as loss of limb awareness in microgravity and perception of phantom body parts [Turvey and Fonseca, 2014]. Passive tensegrity models of the spine [Levin, 2002, Flemons, 2007], knee [Flemons, 2012], and shoulder girdle [Levin, 1997], can capture geometric features of anatomy without the simplifying assumption of a stable platform within the body or a large reduction in the degrees of freedom (compared to more traditional models such as [Cholewicki and McGill, 1996]). A tensegrity model of the human leg is shown in Figure 2.2, for review of biotensegrity models see [Scarr, 2014]. Stability in this leg model is a result of the intuition that the tension network should hold the bones apart. The knee is capable of transmitting compressive forces and bending in the normal range of motion while constrained in others. However, these models are still in need of quantitative validation, particularly of their dynamic properties. These biologically inspired designs motivated robotic tensegrity models of the caterpillar [Orki et al., 2012], and manta-ray's wings [Moored et al., 2011] or tail [Bliss et al., 2008, Bliss et al., 2013], which are discussed further in section 2.5.1 and may assist with quantitative validation.

2.3 Controllers

A controller's role is to solve the problem: 'How do I get my robot to do what I want it to do?' For a robot, this means rotating or translating some parts of the robot relative to other parts via actuators, ideally coordinating the actuators in a desired pattern with higher level algorithms. If the motor commands are pre-specified and never change this is known as open-loop control. One level above pre-specified commands considers a desired position, force, or torque for the motor or some point on the robot (often referred to as an end-effector, such as a hand or foot), and uses feedback on that variable to determine motor commands. If only the variable is considered (say, position), then this is known as proportional control. If derivatives and/or integrals are considered the controller is referred to as proportional-integral-difference or PID control. Cruise control in a car is a PID controller, where the driver chooses the speed. The field of control theory deals with methods such as PID control to ensure a system's output matches the desired input for a variety of increasingly complex systems, including methods for optimization like minimizing energy consumption.

Then there is the question of how to determine the desired position, or setpoint, for proportional control. In some situations this is well known, but in others, additional variables need to be considered before choosing the setpoint. For example, in traffic, distance from the other cars becomes important to choosing a speed. If the setpoint needs to change over time, it is typically referred to as a trajectory. For a tensegrity structure, trajectories typically coordinate length changes between the cables for the desired movement of some point on the robot (center of mass or end effector).

This is where things get interesting (to me, at least). First, determining a trajectory for locomotion of tensegrity structure is complex due to the elasticity of the structure and the coupling between the actuators. Thus, machine learning algorithms are frequently used for determining a trajectory. The algorithms used in this dissertation are described in Chapter 4.1. Second, repeatedly using the same trajectory can

be effective in well known situations, such as walking on flat ground. These situations are often referred to as 'steady state'. However, in non-steady state conditions, such as rough terrain, the trajectory may need to be modified to get around obstacles or change the direction of locomotion to reach a new location. The bulk of Chapter 6 focuses on how to generate and adapt these trajectories, using inspiration from biological systems.

2.4 Tensegrity Simulation and Analysis

Much of the work on controllable tensegrity structures has focused on form finding: determine the equilibrium points of a structure given a set of compression and tensile members, their lengths and connectivity [Masic et al., 2005]. The lengths and connectivity can be represented as vectors of nodes (the points where the members connect) and their relationships can be specified by a connectivity matrix. Form finding led to applications in deployable structures - since the volume of tensegrity structures is mostly empty space, they can be packed and transported more densely than a traditional compression based structure [Skelton et al., 2001]. This is a particularly attractive property for planetary missions, since packing can minimize the launch weight and volume of the structure [SunSpiral et al., 2013]. The form finding algorithm addresses quasi-static movements between equilibrium points, but does not account for dynamic motions of the structure.

Some of the earliest theoretical work on dynamic tensegrity control looks to stabilize the vibration of the structure, such as as Djouadi et al.'s optimal control based algorithm for a tensegrity antenna-beam [Djouadi et al., 1998]. More recent work by Skelton and Friesen develops means to determine required length changes in actuators (inverse kinematics) to move tensegrity structures along known, desired trajectories [Skelton and De Oliveira, 2009, Friesen et al., 2014].

However, these algorithms depend upon well defined trajectories for the robot, and environmental contact points. These assumptions, especially about contact, are rarely valid for robots moving through natural unstructured environments. Physics-based simulation, which can provide more sophisticated environmental information, can help quantify the robots' potential performance in more complex environments. Many of the prior tensegrity simulations were dedicated to modeling only the equations of structures with axial bars only connected to cables [Rovira and Mirats Tur, 2009, Hirai and Imuta, 2012]. Finite element methods such as those used in [McGarry and Prendergast, 2004], also assume pre-defined contacts. Modern physics engines provide advantages such as the fast computation of contact dynamics [Boeing and Bräunl, 2007], and new robot morphologies can be easily implemented and tested in complex environments [Rieffel et al., 2010, Tietz et al., 2013, Mirletz et al., 2014]. Engines including ODE [Paul et al., 2005b, Rieffel et al., 2008, Rieffel et al., 2010], Simmechanics [Orki et al., 2012], and Bullet via the NASA Tensegrity Robotics Toolkit (NTRT) [Tietz et al., 2013, Caluwaerts et al., 2014, Friesen et al., 2014] have been used by tensegrity researchers for simulation. Additionally, the faster run time of these engines means that machine learning algorithms can be used to determine control strategies for structures [Paul et al., 2005b, Iscen et al., 2013b]. For more information on NTRT, see Chapter 3.

2.5 Prior Work: Tensegrity Robots

Tensegrity robots originated when form finding methods motivated exploring actuated tensegrity structures for cantilever like manipulators [Aldrich et al., 2003], since the trajectory of a chosen point on the stucture (end effector) could be represented by a series of equilibrium points. For locomotion, Masic and Skelton developed a

theoretical controller for a tensegrity tower that mimics the peristaltic motion of a worm [Masic and Skelton, 2004]. Paul et al. developed the first mobile tensegrity robot in hardware, a three strut prism capable of crawling [Paul et al., 2005b].

2.5.1 Tensegrity Robot Designs

Designs for tensegrity robots have fallen into two broad categories: those determined by geometry, and those determined by machine learning. The simplest tensegrity structure in three dimensions is a three strut prism, which has been investigated by [Paul et al., 2005b] for basic locomotion and trajectory following in simulation in [Rovira and Mirats Tur, 2009] (with hardware capable of shape changes discussed in [Mirats-Tur and Camps, 2011]). Additional struts can be added to the prism, as is typical for a tensegrity version of a Stuart platform parallel manipulator [Burt, 2013]. These prisms can also be chained together into the helical tensegrity spires, similar to a number of Snelson scupltures [Scarr, 2014], or connected at the rods to make a tower for peristaltic motion [Masic and Skelton, 2004]. Ball joints can be used between compressive nodes to further extend these structures into cylinders, toruses, or spheres [Nagase and Skelton, 2015].

More recent efforts towards the locomotion of tensegrity robots focus on the icosahedron configuration as a crawling or rolling tensegrity robot, as shown in Figure 2.3. Efforts by Koizumi et al. have produced slow 'flops', which deform the structure to change the face in contact with the ground, in a pneumatic tensegrity robot [Koizumi et al., 2012]. This configuration has also been used extensively by the NASA Ames Super Ball Bot project [Agogino et al., 2013], and has subjected the configuration to numerous drop tests [SunSpiral et al., 2013]. Controllers for this morphology are discussed in both sections 2.5.3 and 2.6.

Other tensegrity robots attempt to mimic the geometry and mechanics of a living organism. Orki et al. used an Assur truss to model caterpillar crawling in 2D,

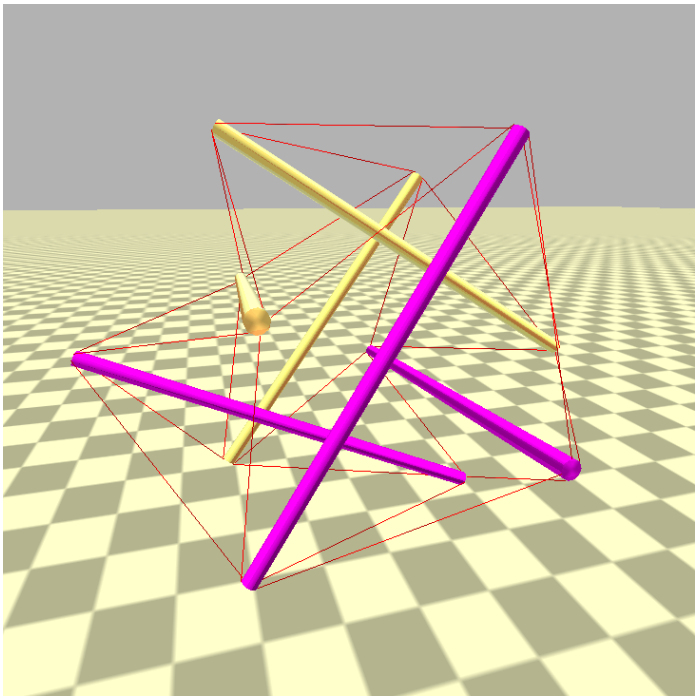

Figure 2.3: The tensegrity icosahedron with six bars and twenty four strings simulated in NTRT. The structure touches the ground at three points.

with struts representing legs and the internal compression, and cables representing muscles [Orki, 2012]. This model was capable of representing the internal pressure characteristics of a crawling caterpillar. Bliss applied a tensegrity truss to a swimming application similar to the tail of a manta ray [Bliss, 2011]. His hardware implementation used two actuated cables, which allowed for a pair of antagonistic actuators, a requirement of the oscillators he used for control [Bliss et al., 2013]. The robotic application of the passive, biologically inspired designs discussed in section 2.2.1 is the subject of Chapter 5.

A tensegrity structure's morphology can also be determined by evolutionary algorithms, which can explore a wider range of possible shapes than intuition alone. The resulting morphologies can either use a high degree of symmetry, as in Connelly and Black's catalog that was generated using group theory [Connelly and Back, 1998], or produce irregular structures by learning the connectivity matrix directly [Paul et al., 2005a]. Rieffel et al., extended this work by encoding

the connectivity into a graph [Rieffel et al., 2009].

2.5.2 Actuation Methods

Prior to discussing control algorithms in more detail, it is important to clarify assumptions about actuation. There are three possible options for actuating a tensegrity structure: actuate the tension elements, the compression elements, or a combination of both. Most researchers avoid actuating both, due to the control complication of full system actuation. While the struts are sometimes actuated in civil engineering applications, such as Fest et al.'s active structure [Fest et al., 2004], robotic applications typically opt for the increased speed available to actuators that change the rest length of tension elements.

A common method (used by [Paul et al., 2005b, Tietz et al., 2013, Bliss et al., 2013, Caluwaerts et al., 2014, Bruce et al., 2014], among others) is to place a motor at one end of the cable, and a tension element (spring) on the other. The tension element can either be a tensile spring, as in the ReCTeR robot from [Caluwaerts et al., 2014], or a compression spring as in [Bruce et al., 2014]. Other techniques for robots include placing linear actuators in line with the tension elements [Kim et al., 2014] or exploiting the oscillations of the structure using vibrational motors [Khazanov et al., 2013].

2.5.3 Machine Learned Controllers

The simplest possible controller for a tensegrity robot is open loop position control via a direct encoding of length changes for the actuated cables. This approach was used by [Paul et al., 2005b, Rovira and Mirats Tur, 2009] and [Iscen et al., 2013a]. The algorithms by Paul et al. and Iscen et al. both explicitly sought periodic actuation patterns, while Rovira and Mirats Tur revised the actuation pattern every 50 ms using a model based controller. As a result Rovira and Mirats Tur demonstrated trajectory

following control for a line, square, circle; a first for a tensegrity robot. However, for any other morphology their search algorithm would result in a state space explosion that could not be solved in real time, so most subsequent research has performed learning offline.

Offline methods to determine a trajectory for the cables include genetic algorithms [Paul et al., 2005b], where control parameters are iteratively modified either with small mutations of existing parameters, or crossover with existing parameter sets to produce (hopefully improved) new controllers. These algorithms can work effectively when only a subset of cables is actuated [Paul, 2006]. More details on genetic algorithms can be found in section 4.1.3. Iscen et al. determined controls for the icosahedron configuration by using a co-learning algorithm [Iscen et al., 2013a]. In this application, co-learning assigns each of the 24 strings (or sometimes a group of strings) a controller and creates 'teams' of controllers to coordinate the entire robot. Controllers are then evaluated by their average score after participating in multiple 'teams' and controller parameters are modified by mutation. When provided with state information such as contact and goal direction, the teams can be trained to move the robot towards a goal location by continuously tipping the center of mass of the robot onto a new face of the icosahedron [Iscen et al., 2014]. While Iscen et al.'s controller was robust to perturbations of short duration during locomotion (such as a large impulse throwing it off the shortest path), the controller needed to be retrained for new terrains. Thus, other researchers have developed closed loop strategies for tensegrity control.

2.5.4 Morphological Communication and Distributed Control

An effective way to close the loop in a tensegrity robot is to distribute controllers and take advantage of the structure's passive dynamics, minimizing communication

between disparate elements. For example, Khaznov et al. were able to make an icosahedron tensegrity robot travel across the floor without rolling by vibrating at different frequencies through online learning using an off-board motion tracking system [Khazanov et al., 2013]. A more active strategy is a spiking neural network, as applied to tensegrity robots by Rieffel et al. [Rieffel et al., 2010] for the structures from [Rieffel et al., 2009]. The inputs of the network are the tensions in the cables, and a spike results in a 50% change in rest length. Rieffel et al.'s controllers relied on the physical structure to communicate control information, a concept known as morphological communication. The weights of the network and the positions of the actuated cables were tuned with a genetic algorithm. Rieffel et al.'s control is closely related to the idea of morphological computation, which relies on compliance and nonlinearities in the physical structure to handle typically complex control problems [Pfeifer and Gómez, 2009, Hauser et al., 2011, Hauser et al., 2012]. The icosahedron tensegrity configuration has also been used for morphological computation for locomotion, performing a gait change after changing the equilibrium length of a few springs [Caluwaerts et al., 2013].

In contrast to much of the prior work, the controller's output could be the tension of the cables. Impedance control (originally developed for manipulators by [Hogan, 1985]), allows for the specification of a stiffness along a trajectory, and thus allows for both the tension and length of cables to be considered. Orki et al. adapted impedance control to tensegrity structures with a scalar equation for the force in each cable [Orki et al., 2012]. Orki et al. then use a binary descending control to command the cables to lengthen or shorten for motion resembling caterpillar crawling. The advantage of the impedance control approach is that impedance control explicitly considers the forces of the actuators, which allows for distributed reflexes for terrain adaptation.

2.6 Central Pattern Generators

The previously discussed algorithms have not dealt with generating online adaptations for a complex robot. Animals are capable of generating adaptive behavior through the many degrees of freedom in their bodies, so control ideas from neuroscience may be effective for tensegrity robots. To handle the challenge of coordinating many degrees of freedom, motor control in animals is distributed across neural centers, ranging from higher centers in the brain, to local control circuits [Grillner et al., 2007]. Well designed low level controllers for robots will allow for descending commands to be relatively simple, easing the job of an operator or high level controller.

The quintessential low-level neural circuit for locomotor control has been described as a central pattern generator [Grillner, 1985b], which is a group of neurons capable of generating a rhythmic pattern. Networks of coupled oscillators forming a central pattern generator (CPG) are typically found in the spinal cord in vertebrates [Grillner, 2006] or in thoracic ganglia in invertebrates [Büschges et al., 2008]. CPGs have been found to control multiple types of locomotion, including walking [Brown, 1911], swimming [Grillner et al., 1981], and flying [Wilson, 1961, Stevenson and Kutsch, 1987], as well as non-locomotor functions such as digestion [Robertson and Moulins, 1981]. Feedback is used to modulate the CPG and adapt the rhythm to the environment [Delcomyn, 1980, Grillner, 1985a]. Additional behaviors can be generated by descending commands from the brain, which change the CPG's rhythmic patterns, such as the transition from walking to swimming in the salamander [Cabelguen et al., 2003].

Computational models of central pattern generators need to represent both oscillators and the connectivity between them. These can range from using detailed models of individual neurons as a 'node' [Hellgren et al., 1992, Daun-Gruhn and Tóth, 2011], to neuron models based on firing frequency [Ekeberg, 1993], to abstract dynamical systems that produce oscillations [Matsuoka, 1985]. For review see [Ijspeert, 2008].

These computational CPGs produce what is known as a stable limit cycle, a path through phase space that is robust to perturbations. The drawback is system complexity; the detailed neuron models require at least two differential equations, including several exponential and hyperbolic terms [Daun-Gruhn and Tóth, 2011], and have a strong dependence on coupling in order to produce sustained oscillations. Simpler nodes have two linear differential equations [Matsuoka, 1985], but again are highly dependent upon coupling. These drawbacks can make CPGs for robots hard to design and tune.

An important simplification for engineering use is to assume a node has all of the properties it needs to oscillate, representing a population of neurons [Buchli et al., 2006]. Thus a designer can focus on the network dynamics, and oscillator parameters can have clear effects on the output. One example is an amplitude controlled phase coupled oscillator implemented by Crespi and Ijspeert at École Polytechnique Fédérale de Lausanne in Switzerland (EPFL) on a lamprey-like swimming robot [Crespi and Ijspeert, 2006] (see equations 6.2, 6.3, and 6.4 in section 6.2). This CPG successfully simplified the parameters to clear amplitudes and phase differences. The same oscillator with a different network topology was also used in their salamander [Ijspeert et al., 2007] and boxfish inspired robots [Crespi et al., 2007]. CPGs in those robots also simulated a descending command from the mesencephalic locomotor region of the brain to change gaits and turn, similar to salamanders [Cabelguen et al., 2003].

Properly formulated oscillator equations can even adapt to their environment online using local feedback, and continue using that pattern if feedback is lost [Righetti et al., 2009]. This entrainment behavior was used on a swimming tensegrity by Bliss et al. [Bliss et al., 2013]. Feedback can also come from non-rhythmic sources. Utilizing feedback from a simulated gyroscope, Gay et al. implemented the phase coupled oscillators of the salamander/lamprey research on a quadruped

with a rigid torso [Gay et al., 2013]. The feedback parameters were learned offline though particle swarm optimization, which enabled the simulated robot to traverse a 55 degree downward slope and randomized rough terrain.

One of the problems with existing tensegrity controllers was that controllers needed to be re-tuned for each terrain type. CPGs could solve this problem by allowing for online adaptations via feedback functions. Additionally, properly designed low level controllers can mimic reflexes, to complement the mechanical compliance of the tensegrity structure. Therefore this dissertation will use CPGs with feedback and impedance controllers with a compliant tensegrity spine for locomotion on irregular terrain. Machine learning methods will help determine parameters for these controllers.

Chapter 3

The NASA Tensegrity Robotics Toolkit

A primary focus of this dissertation was developing the NASA Tensegrity Robotics Toolkit as a platform for simulating tensegrity robots and developing control algorithms for those robots[1]. This chapter summarizes the architecture and features of NTRT that were developed for and used in this work.

Note that capabilities and accuracy of NTRT have improved dramatically through this work, and many papers that comprise this dissertation were written before relevant features of the toolkit were implemented. For reference, Appendix B summarizes the features and their implementation dates, as well as the dates that relevant data was recorded from the simulator. Generally speaking, the control algorithms discussed in Chapter 6 have been successful throughout the improvements to the simulator, with appropriate retuning of parameters.

[1]Information, source code, and documentation for NTRT can be found at http://irg.arc.nasa.gov/tensegrity/NTRT

3.1 The Bullet Physics Engine

Rigid body physics simulation engines handle the task of integrating the equations describing the motion of objects and detecting and resolving their collisions. Depending upon the constraints of the system, it may be more effective to write simulator equations directly in terms of the positions of joints, rather than world space (Cartesian coordinates). Physics simulators can thus either be implemented in Cartesian space, or joint space. Given that there are not rigid constraints between rigid bodies in tensegrity structures, in NTRT they are simulated in Cartesian space. Additionally, implementing terrain interaction is easier when simulating in Cartesian space. The NTRT simulator uses Bullet Physics version 2.82 as its underlying physics engine [Coumans, 2012]. The initial choice of Bullet was motivated by:

1. Bullet's inclusion of soft body dynamics
2. The desire to use all open source code with NTRT
3. The large community of developers that use Bullet, and an active support forum

(1) and (3) were the key reasons for choosing Bullet over the Open Dynamics Engine (ODE), another common choice within robotics. [Erez et al., 2015] provides a recent comparison between the rigid body capabilities of these and other engines.

Simulations in NTRT are run at a fixed timestep (between 1000 and 4000 Hz for the work in this dissertation), at which they can either run in real time with rendering (60 Hz), or faster than real time without rendering. For example, a 60 second simulation without rendering or cable contact dynamics can run in approximately 7 seconds on an Intel Core^{TM} i7 CPU. The use of numerical integration for CPGs also slows runtimes, the slowest simulations with contact cables and complex terrains tend to be about 1/2 of real time (two real seconds to one simulation second).

We construct models in NTRT using a set of builder tools that takes a tagged set of Cartesian coordinates (nodes) and their connectivity (pairs), and generates

a tensegrity structure according to physical properties of the rods (radius, density, friction, restitution) and cables (stiffness, damping, motor properties). Models can either use objects of uniform mass, such as a series of compounded rods, or place point masses at nodes to simulate mass concentrations such as motors and increase the accuracy of the inertia matrices used by the simulator. Examples of model parameters are given in Appendix A. Units used within the simulator are scaled to keep the numbers within reasonable ranges for numerical integration, and gravity is scaled accordingly. Simulations were run at cm scale unless otherwise noted (gravity at 981 cm/sec), data presented in this dissertation has been scaled as necessary.

3.2 NTRT's Cable Models

My initial work in [Tietz et al., 2013] used Bullet's default soft-body cables, which use a position based animation method. Later work determined that these cables were not realistic. In particular, the cables' elasticity coefficient did not have a force term, therefore they could not simulate the forces required to appropriately scale the world. Additionally, their behavior was wildly different with varying timesteps. Therefore we implemented our own custom cable models. Changing the cable model has not reduced the speed of the spine like robots, but it has made crossing rough terrain realistically difficult. Comparison data is provided in Chapter 6.

3.2.1 Internal Dynamics

NTRT's custom cable model uses a Hooke's law based spring and cable model originally implemented in [Caluwaerts et al., 2014]:

$$\mathbf{f} = \begin{cases} 0 & : \|\mathbf{x}\| \leq \ell_i \\ (k(\|\mathbf{x}\| - \ell_i) + b\|\dot{\mathbf{x}}\|)\hat{\mathbf{x}} & : \|\mathbf{x}\| > \ell_i \end{cases} \tag{3.1}$$

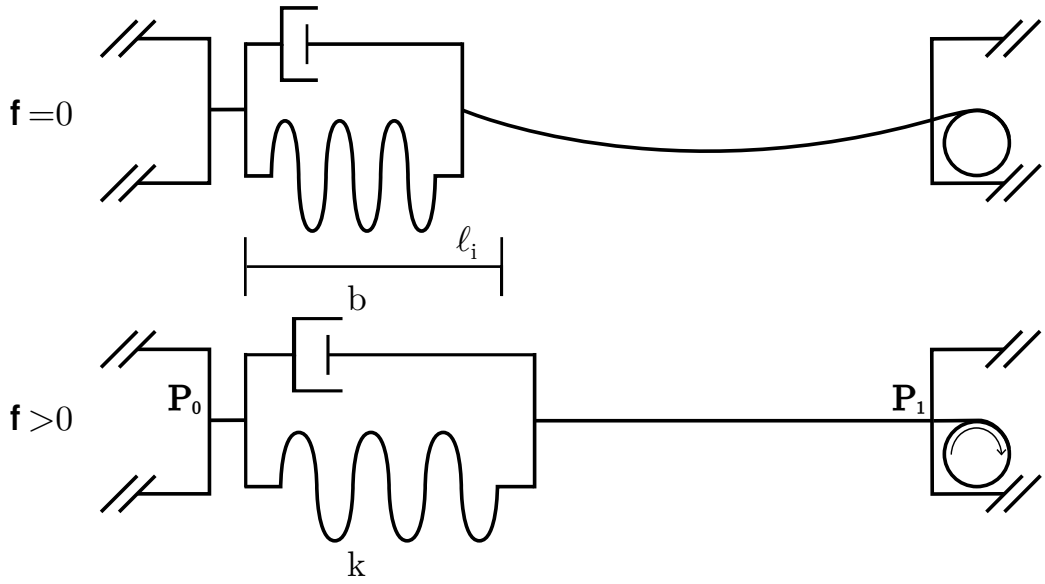

Figure 3.1: A graphical representation of equations 3.1 and 3.2. The top picture represents a slack cable. In the bottom the motor has spun clockwise, tightening the cable and applying equal and opposite forces to the rods.

$$\mathbf{x} = \mathbf{p}_{i,0} - \mathbf{p}_{i,1} \tag{3.2}$$

where k is the spring stiffness, b is a linear damping term, x is the Euclidean distance between the attachments of the spring-cable assembly ($\mathbf{p}_{i,0}$ and $\mathbf{p}_{i,1}$), and ℓ_i is the cable length. Actuation is accomplished by changing ℓ_i, which is subject to speed and force constraints comparable to the actuators in hardware designs. Our simulation with this cable model was validated by our prior Planetary Lander work to be accurate within 15 mm for a robot with 1 m struts when performing a roll [Caluwaerts et al., 2014]. Further validation on tensegrity spine models is discussed in Chapter 7.

Initially our actuator model was ideal (flat torque speed curve) and inertia free, so that as long as these constraints were not violated, the rest length would change as desired. Subsequently, I implemented a model of an actuator with inertia and friction, using the following equation for a brushless DC motor:

$$\ddot{\theta} = (\tau_a - b * \dot{\theta} + r_0 * T)/I \tag{3.3}$$

where θ is the angle of the rotor, τ_a is the torque applied by the stator, b is a damping constant (units of $length^2 * mass/second$), r_0 is the radius of the rotor and spool, T is the tension in the cable, and I is the inertia of the rotor. Setting $r_0 = 1$ models a linear actuator, τ_a is typically constrained by a linear torque-speed curve. θ is integrated semi-explicitly within the model's class. The motor model can either be backdrivable (movable by external forces alone) or non-backdrivable.

3.2.2 Collision Handling

The above cable model was originally implemented without contact dynamics, so it only interacted with the two rigid bodies to which it was attached. When restoring the contact dynamics, a major design consideration was whether or not to add mass to the cables. [Servin et al., 2011] provides an equation for determining the maximum force a given cable can simulate in a numerically stable fashion, based on the vibrational modes of the cable and its numerical stability:

$$f_{crit} = \frac{LM}{4N^2h^2} \tag{3.4}$$

In a typical tensegrity simulation: a reasonable maximum force, f_{crit} is 100 N, the length of the cable $L = 0.3$ m, the mass of the cable $M = 0.004$ kg, and the timestep $h = 0.001$ s. N represents the number of mass nodes. Under these conditions, a stable simulation could only use three mass nodes, which is not sufficient to properly determine contacts. Thus, I developed a massless contact model using behavior inspired by [Servin and Lacoursière, 2007] and [Servin et al., 2011]. Collision detection is handled by ghost objects within Bullet, placing a small cylinder between the two endpoints for collision detection. As additional contacts are accumulated, additional

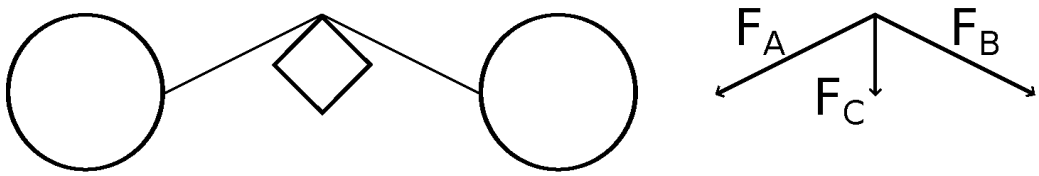

Figure 3.2: Left: Two rods and a cable encounter a square block, creating a single point of contact. The cable in this case would be modeled with two cylinders for collision detection. Right: A graphical representation of equation 3.5 for this situation. The contact force $\mathbf{f}_c$ is applied to the block, while cable forces $-\mathbf{f}_a$ and $-\mathbf{f}_b$ are applied to the rods.

cylinders are added to a compound object, subject to a maximum resolution. Thus, Bullet determines contact points, and NTRT's code resolves the contacts. Forces are applied to detected contacts by the following equation:

$$\mathbf{f}_c = \mathbf{f}_a + \mathbf{f}_b \tag{3.5}$$

where $\mathbf{f}_a$ and $\mathbf{f}_b$ are the forces along the length of the cable between the two nearest end or contact points and $\mathbf{f}_c$ is applied to the contacted object along the contact normal. A diagram of the application of this equation is shown in Figure 3.2. Figure 3.3 shows a NTRT simulation of a similar situation.

After they are placed, contacts are subject to an overall cable length minimization similar to [Servin et al., 2011], which accounts for sliding. Contacts are removed when they violate the resolution constraint, their force would be applied 'outward' of the geometry (i.e. the cable would push), or when contact is lost. Additional details of the algorithm are specific to Bullet, and are discussed in Appendix B.

3.3 Additional Features

NTRT also includes:

1. Machine learning algorithms, discussed in Section 4.1.

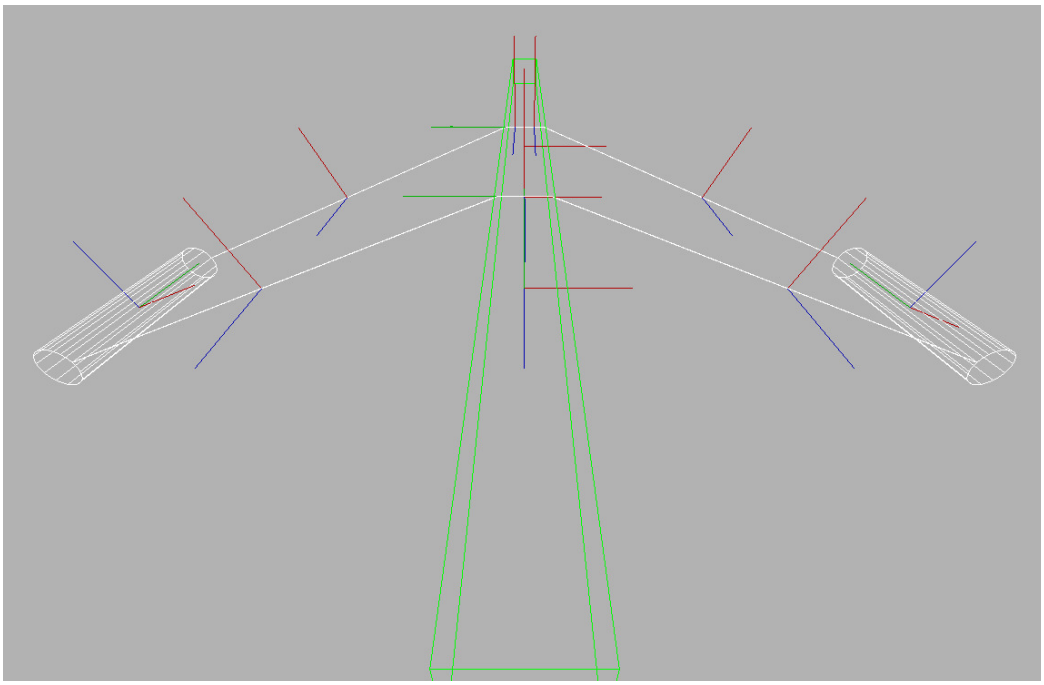

Figure 3.3: An analogous situation to Figure 3.2 simulated in NTRT, except the cable contacts the rod in two places due to its orientation. The axes indicate the geometric center of each collision object.

2. Classes to implement Impedance Control and PID Control (Chapter 6).

3. Numerical integration using ODEInt [Ahnert and Mulansky, 2011], for the Central Pattern Generator models discussed in Section 6.2.

Chapter 4

Machine Learning and Metrics

4.1 Learning Algorithms

This section discusses how I approached parameterizing controllers for tensegrity robots, and tradeoffs between the algorithms I used. The specific structures and controllers for my tensegrity work are discussed in the next chapters. However, the metrics and learning algorithms can generalize to the control of most types of robotic locomotion.

Determining an effective controller is one of the challenges of using robots with many degrees of freedom, such as tensegrity robots. While these controllers can be hand tuned, this typically requires a reduction of the dimension of the problem. To preserve the advantages of many degrees of freedom, controllers are frequently designed using machine learning or optimization techniques. Machine learning has two major advantages:

1. Machine learning algorithms can 'discover' designs and parameterizations that a human might not think of intuitively.

2. The algorithms can be run in the background, freeing the designer to work on other things, such as writing.

The disadvantage is that machine tuning a controller for locomotion is difficult, since the algorithm might not get enough information to differentiate between modes of failure. For example, the robot falling over may produce an indistinguishably low score of an objective function, compared to the robot getting stuck on a terrain feature. Without meaningful score differences between scenarios, many algorithms have trouble determining what changes to make to parameters. On the other hand, highly random algorithms frequently need a large number of function evaluations in order to find an effective controller (known as the tradeoff between exploration and exploitation).

This work uses algorithms that explore the state space more randomly, with the objective function generally being distance traveled. Using highly exploratory algorithms enables handling high dimensional spaces, such as fully actuated structures. Thus, future efforts will understand what tradeoffs are being made when grouping or reducing the number of actuators.

For this section, a 'trial' refers to a single instance of the simulation, running from zero to N simulation seconds. A 'generation' refers to a group of trials with a fixed set of controllers. Finally, a 'run' refers to a complete evolution, which has ideally converged on a maximum.

4.1.1 Monte Carlo

If simulation times are sufficiently short, it is possible to randomly sample the parameter space and locate the neighborhood of good solutions. This is called Monte Carlo sampling. Since the parameter estimation is random, trials can be run entirely in parallel, there is no need to synchronize for a mutation step. Monte Carlo provides a significant advantage to cluster computing, since the speed of the algorithm scales linearly with the number of processors. This algorithm allows for a rapid (though not exhaustive or optimal) search of a complex parameter space, allowing us to quickly

identify parameters for effective locomotion. Domain knowledge is sometimes necessary to bound solutions, such as setting the maximum frequency and amplitude for cable trajectories. A typical result of a Monte Carlo run is shown in Figure 4.1. Most of the solutions are terrible, moving less than 14 cm in 30 sec. However, around one in a thousand is good enough for effective locomotion. These can be tuned further using other algorithms.

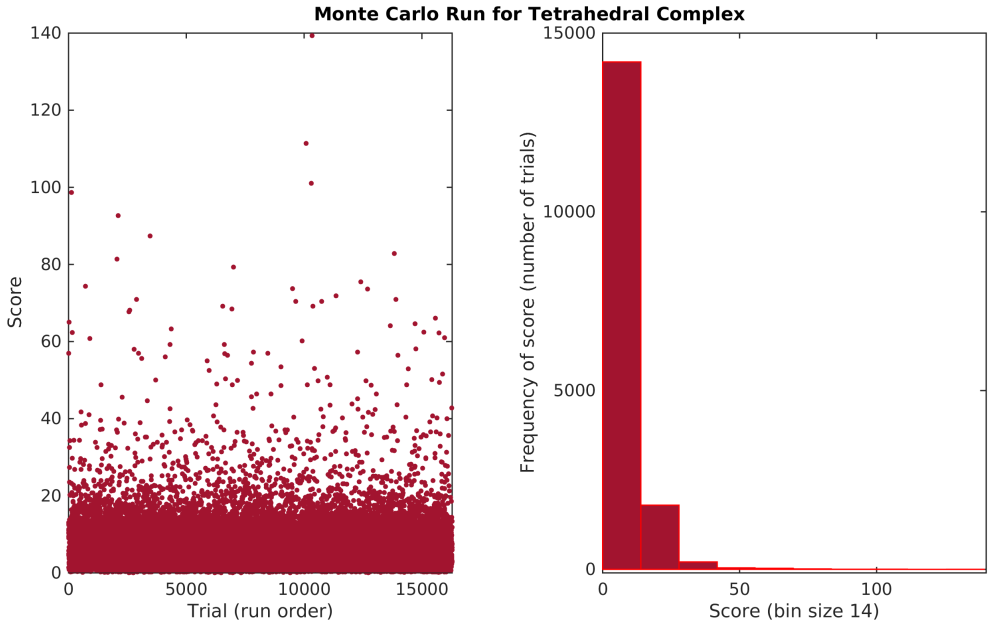

Figure 4.1: The results of a typical Monte Carlo learning run. As expected, there is no correlation between trial number and score. Scores above a certain threshold become the seed of the subsequent learning run.

4.1.2 Gaussian Sampling

In order to optimize the controllers determined by Monte Carlo, I typically sampled nearby control parameters using a Gaussian distribution. Thus, the prior controllers become the 'seed' of the Gaussian sampling step. If the new controller is better, sampling continues around the new controller's parameters. A single random controller is unlikely to be good in this application, so seeding with prior controllers is strongly

recommended. Parameters to be mutated were chosen by chance (typically 50%) and mutated simultaneously, with no upper limit on the number of parameters to mutate. The operator needs to determine the deviation of the Gaussian used to sample. Performing Gaussian sampling after Monte Carlo is sometimes called two-tier Monte Carlo.

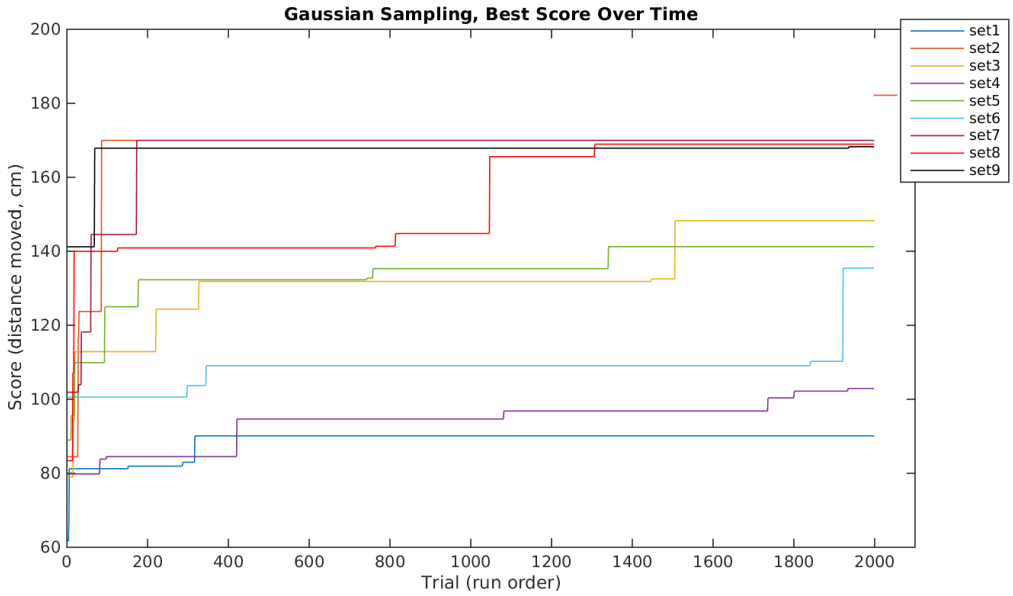

Figure 4.2: Sampling around the best controllers from Figure 4.1. The vertical jumps indicate a new best controller.

Gaussian sampling can also be thought of as evolution with a population of two, the better of which is mutated. The results for eight controllers (optimized in parallel) are shown in Figure 4.2. Note how the improvements occur more frequently than 'good' parameters in Monte Carlo, especially at the beginning.

4.1.3 Evolution

A large category of algorithms fit into the category of 'evolutionary algorithms'. The general idea is that a population of controllers (generally, agents or organisms) is iteratively evaluated by a fitness function. Depending on the task, in this work

population sizes for evolutionary algorithms varied from 2 to 100. The best controllers are then allowed to reproduce (influence the next generation) by any of a number of methods, detailed below, and the worst controllers are discarded. There are three methods by which controllers can be chosen for the next generation.

Elitism - directly copying the best controllers over to the next generation. Depending on the objective function these controllers may or may not need to be re-evaluated in the new generation.

Mutation - changing some (or all) parameters of the best controllers by a small variation. This is similar to Gaussian sampling, but any number of controllers can be mutated in a generation since the population is larger.

Crossover - combining groups of parameters (an analogy is often made to genes in the literature) in the controller. There are multiple methods for crossover, the most common of which is picking a location in the parameter vector and using one parent's vector before that location and the second's after (one-point crossover). An alternative is selecting random parameters from each parent (uniform crossover).

Population members can either be a seed from a prior method like Monte Carlo or hand tuning, randomly determined, or some combination. Typically an artificial neural network (discussed in detail in Chapter 6) can be evolved starting with a random seed, whereas a CPG needs to be evolved from an existing good seed.

Other algorithms group parameters for evolution. For example, Iscen et al.'s coevolution algorithm tested controllers for specific cables, and used an historical average to choose the best controller for each individual cable [Iscen et al., 2013a]. This resulted in faster evolution and a better final controller than a centralized scheme. I adapted this by grouping parameters by function, for example impedance controller, feedback, and CPG. Thus the best historical average within each functional group can be used.

Specific patterns for an evolution run are included in Appendix A.

4.1.4 Implementation within NTRT

Two iterations of machine learning algorithms were implemented within NTRT. The first generation utilized C++ learning modules. As implemented, multi process learning within C++ was possible by spawning multiple processes locally on the same machine by hand, but crossover for evolution algorithms could not be multi-threaded, since each thread was fully independent. In March and April 2015, Perry Bhandal and I transferred the learning algorithms to Python, and provided parameters to the C++ controllers through a JSON (JavaScript Object Notation) interface. This eased the implementation of automated multi-process learning, which was crucial to speed up the algorithms for learning across multiple terrains with the contact cables. In this way, the trials for an entire population could be parallel processed, and then synchronized for the mutation step.

4.2 Metrics and Objective Functions

4.2.1 Objective Functions

For most of this work, the most practical objective function was distance traveled for a set length of trial. This can either be distance in any direction, or distance towards a specific location (goal). In both cases this is measured 'as the crow flies' comparing the starting and ending positions of the robot, rewarding controllers that travel relatively straight paths. There are several key caveats to this. First, the length of the trial matters. If a goal directed trial is too short for the robot to arrive at the goal, the learning algorithm has no knowledge of whether the robot will in fact get to the goal, as the robot could leave its current path after the trial. Second, there is the question of historical averaging: Should controllers keep scores from previous generations? Historical averaging is particularly valuable for non-deterministic evaluations, so the

best controllers continue to incorporate information about possible worlds. Averaged evaluations can consist of multiple goal positions, starting robot orientations, and terrains. However, it is difficult to optimize for all of these criteria in one run, so we organized the sub-problems as follows:

1. Use Monte Carlo to determine a controller that coordinates effective locomotion in a structure (any direction).

2. Improve the performance of that controller on flat ground using Gaussian sampling (any direction).

3. Add a feedback controller, tune the performance on rough terrain using evolution (any direction).

4. Determine a controller that can reach a single goal on rough terrain using evolution.

5. Evolve from a single goal to multiple goals on rough terrain.

Terrain Types

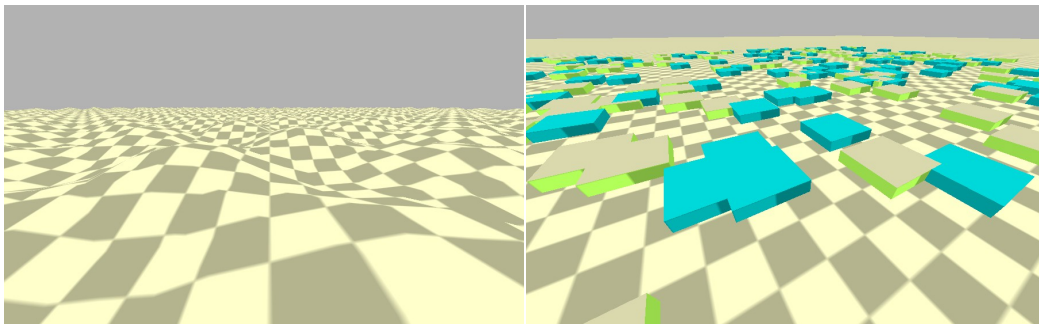

Figure 4.3: The two primary rough terrain types used throughout this work. Left, sinusoidal hills. Right, randomly placed blocks.

Terrains considered in this work included flat ground, hills, and a field of blocks. Hilly terrain was composed of sinusoidal hills with an amplitude of 2 cm, and was

represented as a single triangular mesh within Bullet. The block field consisted of 500 randomly placed blocks on otherwise flat ground. The blocks were 5 cm wide, 0.5 cm tall and were fixed to the ground within a 200 cm by 200 cm area around the origin. These terrains are shown in Figure 4.3.

These terrains were chosen since they represent slightly different problems. While the hills are a larger perturbation, they are perfectly smooth and were implemented to be deterministic. The sharp corners on the blocks are more likely to catch cables. The randomness of the blocks provides some assurance that the controllers have not been overfit. The size of the terrain features was chosen as the smallest features that impeded the existing spines and controllers, more details are given in Chapter 6.

In the original work, Tetraspine was also capable of traversing a wall and going up a ramp via manual steering and adjustment of cable tensions. These obstacles are not feasible to learn without autonomous goal direction, as a controller without a goal input would always go in an easier direction. Given that a goal directed controller is included in this work, future work could re-implement these behaviors.

4.2.2 Cost of Transport

While distance traveled is a good metric for evaluating a new structure, it is not the only metric that should be considered. Therefore we also computed the efficiency metric cost of transport (COT) and included it in the results. It was not included in the optimizations. Cost of transport is defined using Equation 4.1, as discussed in [Tucker, 1970]:

$$COT = \frac{W}{mgd} \tag{4.1}$$

Where W is the work put in to the system, m is the mass, g is gravity, and d is the distance traveled. Cost of transport is unitless, and provides a means to compare different mobile systems. Given equation 4.1, a lower COT indicates a more efficient

system.

This metric highlights a tradeoff between the overall capability of the system and its performance on flat ground. Bicycles are great on roads, but become much more difficult to use in the woods without a path, where walking or running is much more effective.

The most efficient legged robots are typically designed by studying the passive dynamics of walking, taking advantage of the resonance modes of these systems on flat ground, and then seek to minimize control input by using reflexes [Bhounsule et al., 2014]. Their approach achieved the record cost of transport in a legged robot of 0.19, comparable to human walking (0.2). Of note this record cost of transport was achieved on a linoleum lab floor, a controller that was robust enough to walk on Cornell stadium's track for 65 km had a higher cost of transport (0.28). For efficient running, Seok et al. focus on an electromechanical optimization for their cheetah like robot, and utilize custom, regenerative motors and low inertia legs [Seok et al., 2013]. A recent implementation can run at 4.5 m/s with a COT of 0.5 [Park et al., 2015].

Within legged robots, the most capable robots, such as those from Boston Dynamics, have an estimated total cost of transport of up to 15 [Ruina, 2011]. Honda's Asimo is estimated to have a total cost of transport of 3.2 [Collins et al., 2005]. The mechanical cost of transport in simulation for our current tensegrity systems range from 1.55 to 3.2, depending on the structure and controller [Iscen, 2014, Mirletz et al., 2014]. This is within the expected range for robots that have not yet been optimized for efficiency, and these gaits have been able to perform agile goal directed locomotion through rough terrain [Iscen et al., 2014], whereas the more efficient robots have only shown basic turning on relatively flat ground.

To compute the energy input to the tensegrity spines with the basic actuators, we used the sum of the work to shorten the strings: Tension times distance shortened at

each timestep. For energy calculations on the basic (ideal) motor model, we assumed that relaxation was passive, similar to biological muscles or a backdriven motor. The cost of transport for a system similar to our hardware implementation may be higher if the motors are actively unspooled during relaxation. With the kinematic (brushless DC) motors, we integrate power (applied torque times speed) over time.

Chapter 5

Tensegrity Spine Structures

My exploration of tensegrity spines has been an iterative process. The four spines presented here represent different design goals, which is the focus of this chapter. Quantitative results for all spines are discussed in the next chapter, alongside the relevant controllers.

Given their software implementation, all spines could be lengthened or shortened with a single parameter (see 6.3 for details on how the controller adapts.) Spines were typically tested with twelve segments in computationally simple scenarios (no contact dynamics in the cables), and six for more complex scenarios. The length of a given spine in segments will be given in each iteration of the results.

5.1 Tetraspine

For our initial morphology, our primary goal was to simplify the morphology to determine the feasibility of a tensegrity spine as a robotic structure. Inspired by the structures in [Flemons, 2007], Tetraspine (Figure 5.1) uses tetrahedrons as 'vertebrae', with six cables connecting each segment, the minimum for a tensegrity spine as defined for this dissertation. The arrangement of the cables allows for active control of pitch, yaw, and axial translation, and each cable has a clear line of action within

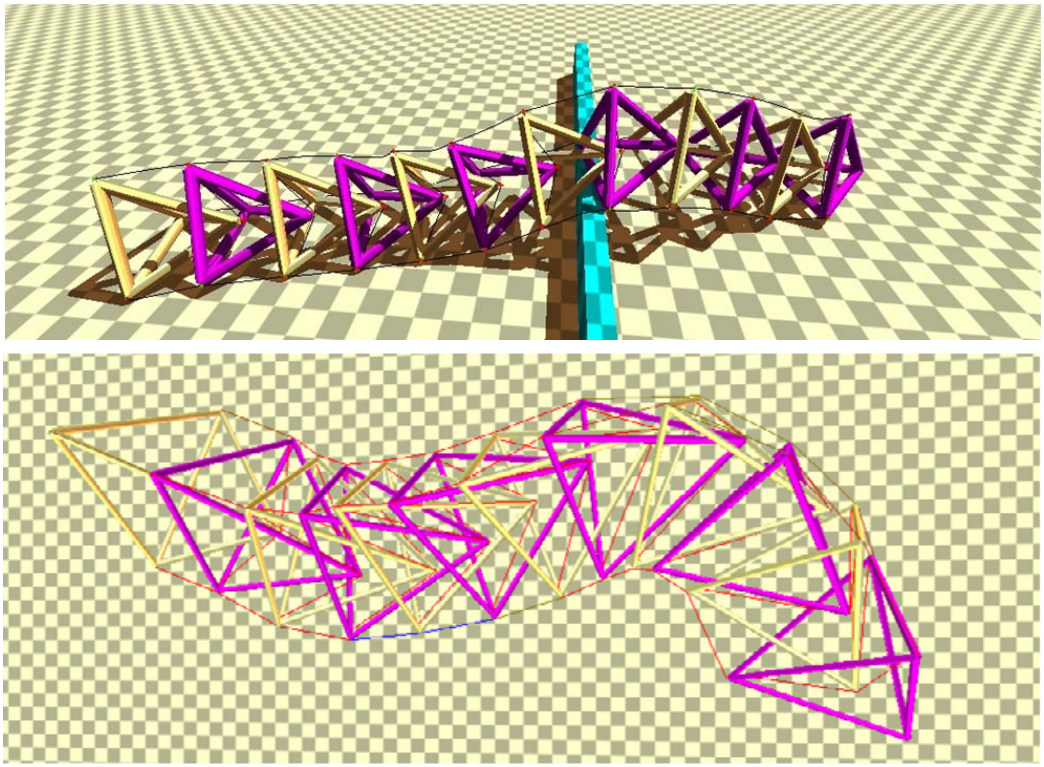

Figure 5.1: Tetraspine, a tensegrity spine robot with tetrahedrons as segments, and six cables connecting each segment. From [Tietz et al., 2013, Mirletz et al., 2014]

these degrees of freedom. Three cables extend from each outer vertex to the segment behind it, and three cables extend from the tip of each segment to the outer vertexes of the segment in front of it. Convex tetrahedrons for segments also allow a lot of room for mounting motors, sensors, and control boards for a hardware implementation, as in Chapter 7. This morphology successfully demonstrated the first locomotion of a tensegrity spine robot both in simulation and hardware [Tietz et al., 2013].

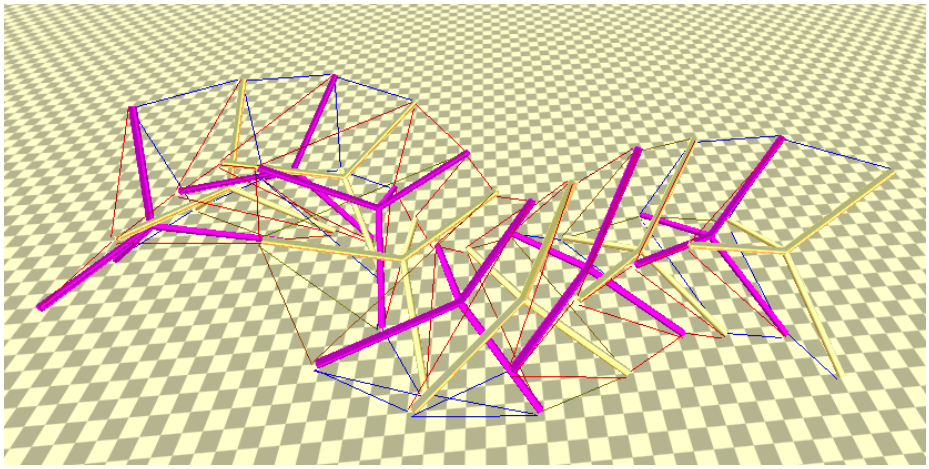

Figure 5.2: The tetrahedral complex spine, with eight cables between segments, simulated in NTRT performing a sidewinding motion.

5.2 Tom Flemons' Designs

Once NTRT was sufficiently developed with tools to assist the creation of structures and design of controllers, I implemented Tom Flemons' spine design as shown in Figure 5.2. This structure uses a tetrahedral complex as its vertebrae, with four cables composing a tensegrity saddle joint [Flemons, 2007], and four cables running along the outside. All eight cables are actuated in this case. Changing the morphology of the structure from a convex tetrahedron to a tetrahedral complex and the addition of two cables adds two important elements to the structure, when compared to Tetraspine. First, the cable routing allows control of all six degrees of freedom. Second, the center of mass of the structure is now over its line of support, which decreases the tension required to hold the structure's shape by a factor of 500 when compared to Tetraspine: 10 N on Tetraspine vs 0.2 N on the tetrahedral complex. Note that the force required to hold a static shape with Tetraspine increased to 25 N once we accounted for motor placement. We have not yet done detailed simulations with motor placement on the tetrahedral complex, though mechanisms are being designed for a hardware implementation [Sabelhaus et al., 2015]. This design has stood out with the fastest locomotor speeds and the highest efficiency.

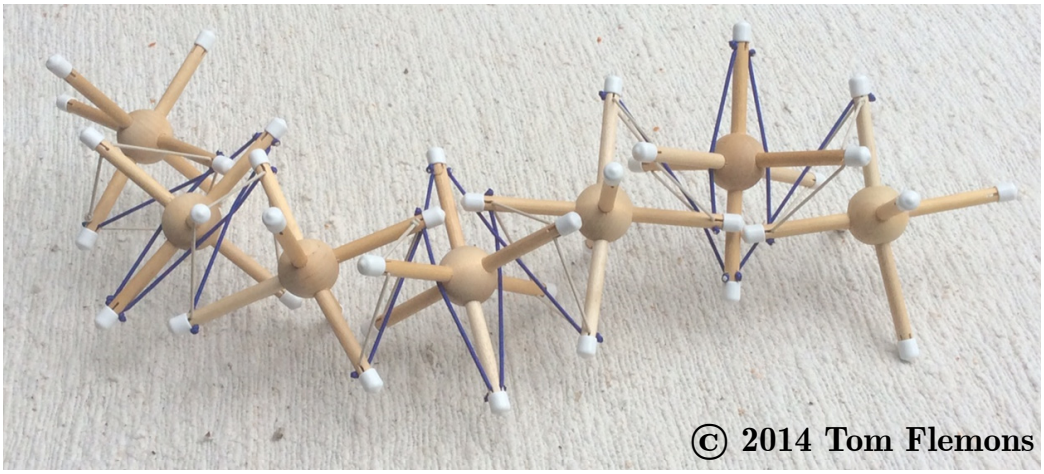

Figure 5.3: Octahedral complexes form the vertebrae, with four passive (white) and four active (blue) connections between segments. The simulated spine has twice as many actuated cables. Every saddle joint has a pair of actuators on either side. Image courtesy of Tom Flemons, originally from [Mirletz et al., 2014]

One possible disadvantage of the tetrahedral complex shape is the high number of actuators per segment. In order to reduce the number of actuators and maintain stability we decided to place passive cables at each of the tetrahedral saddle joints. However, in the previous morphology this would severely limit the range of motion, so Mr. Flemons suggested we try cross-linking through the saddle joints. Four cables compose the saddle joint, while another four cross link on either side of the saddle to provide antagonistic actuation. We also transitioned to an octehedral complex as the vertebrae shape, to avoid interference between the cables and vertebrae (this was initially implemented prior to the cables' contact dynamics). See Figure 5.3 for a passive model with half the number of cross linked cables (blue cables). Note how the angle of the spine changes between each vertebrae. This structure is bistable, each of these can be 'switched' to the other position and stay there as well. When resting on the ground, only one rod from each vertebrae touches.

This structure was my first attempt with unactuated cables, as opposed to the Tetraspine cables with their velocity setpoint of zero. This morphology also successfully demonstrated locomotion in simulation.

5.3 Novel Designs

In order to do additional analysis on how the shape of the structure affects its motion, I designed one novel structure. We hypothesized that snake like slithering motions would be easier on a closely packed vertebrae structure with longer attachments, like the ribs on a snake skeleton. I created the shape by attaching 3/4 of an ellipse in place of one of the rods of the tetrahedral complex.

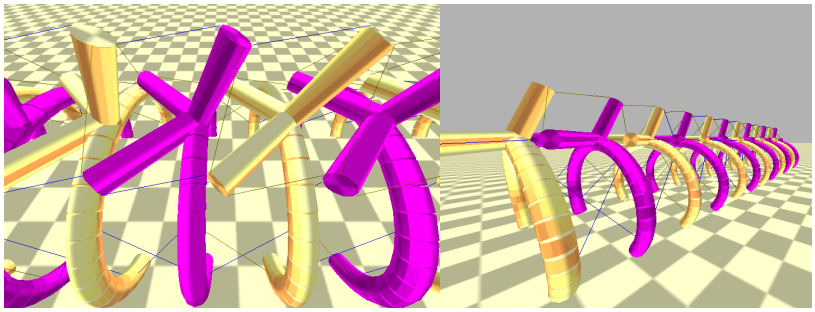

Figure 5.4: Ribs are rigidly attached to three rods in a configuration that mimics three rods of the tetrahedral complex. cables that only touch ribs are passive, the other seven are active. Originally from [Mirletz et al., 2014]

Actuator locations were chosen by examining anatomy textbooks and mimicking muscles [Gray, 1918, Myers et al., 2001]. Five cables surrounding the spine mimic the Interspinales and Intertransversarii muscles. Our initial tests with these muscles proved unstable, so we added four passive cables between each segment's ribs, mimicking fascia in the thoraxic region. The passive cables were half of the stiffness of active cables (500 N/m), and were not pretensed. This was still insufficient, so we added two more muscles connecting the transverse processes to the inferior ribs. Upon reexamination of the anatomy books, we realized these may play a role similar to the Levatores costarum. This spine displayed two forms of locomotion, one of which was slithering.

Chapter 6

Control Methods and Simulation Results

This chapter details the evolution of the tensegrity spine controller over the course of this project, starting with the simplest controller and ending with a controller capable of goal directed locomotion over rough terrain. Occasionally this includes the literal evolution of the controllers via machine learning, as good controllers from a previous iteration make good seeds as new components are added. Rather than presenting all of the controllers and then all of the results in a separate chapter, this chapter presents simulated, full-system results as closely as possible to the relevant controller.

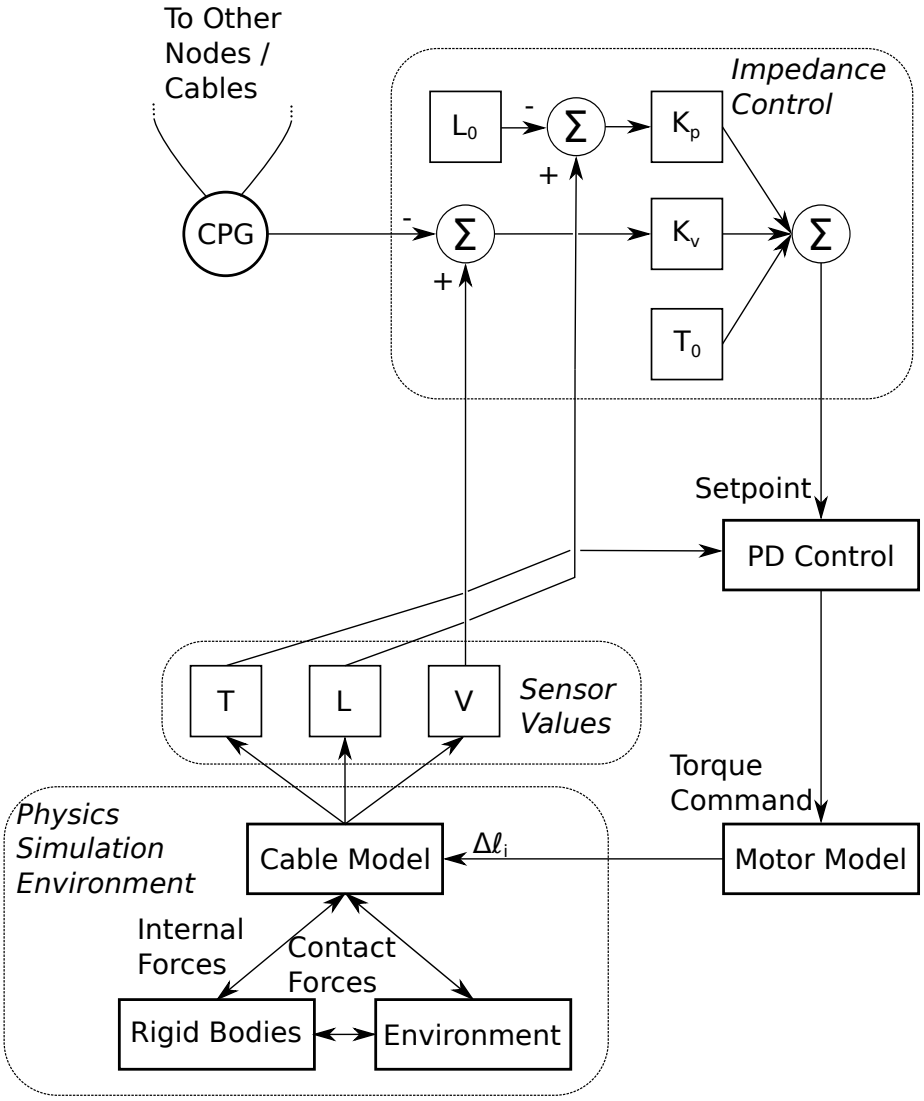

Figure 6.1: The control system for a single cable. A block diagram for impedance control is shown in the upper right hand corner, graphically representing Equation 6.1. CPG connectivity is shown in Figure 6.6.

6.1 Impedance Control

As discussed in Section 3.2, in NTRT we assume the robot's cables are actuated. Given typical sensors and actuators, this means that there are two variables to control: the cable's length and its tension. In order to incorporate both, we use a distributed

form of impedance control, adapted for tensegrity by [Orki et al., 2012]. Impedance control provides tunable stiffness through changing its gains. The equation is as follows:

$$T = T_0 + K(L - L_0) + B(V - V_0) \tag{6.1}$$

Where T is the tension setpoint, T_0 is a tension offset, K is a position gain on the difference between the cable's current actual length L (x in (Equation 3.1)) and desired length L_0. B serves a similar function for V and V_0 (velocity, equivalent to $\dot{x}$), where V_0 is the trajectory: a control input from the CPGs or sine waves [Tietz et al., 2013]. In Orki et al.'s implementation V_0 is always zero [Orki et al., 2012]. All variables except V and V_0 should be greater than or equal to zero for stability, given the convention that lengthening is positive. L_0 is typically the initial length of the cable, assuming the robot starts in a desirable shape. If the result is $T < 0$, the controller will set $T = 0$. The tension setpoint is then sent to a PD controller for low level motor control. Typical PD control values were $P = 20000$, $D = 5$. A block diagram is shown in Figure 6.1. Typical impedance controller gains are given in the caption of Figure 6.2.

The trajectory is controlling the velocity for three reasons. First, since I am targeting dynamic locomotion, moving the cables is desirable, hence not setting $V_0 = 0$ as in [Orki et al., 2012]. Second, keeping L_0 constant allows for information about the shape to be maintained in the controller, oscillations are then maintained around the equilibrium of the length and tension offsets. Finally, avoiding a slack cable is desirable for maintaining coherent sensor information. The tension offset can be used to increase the equilibrium point of oscillations for this purpose, and for minor shape changes. In practice, effective amplitudes for V_0 are far larger than the motors are capable of, thus the oscillations increase and decrease the tension in the cable.

Figure 6.2, shows how one of the top outside cables behaves while Tetraspine

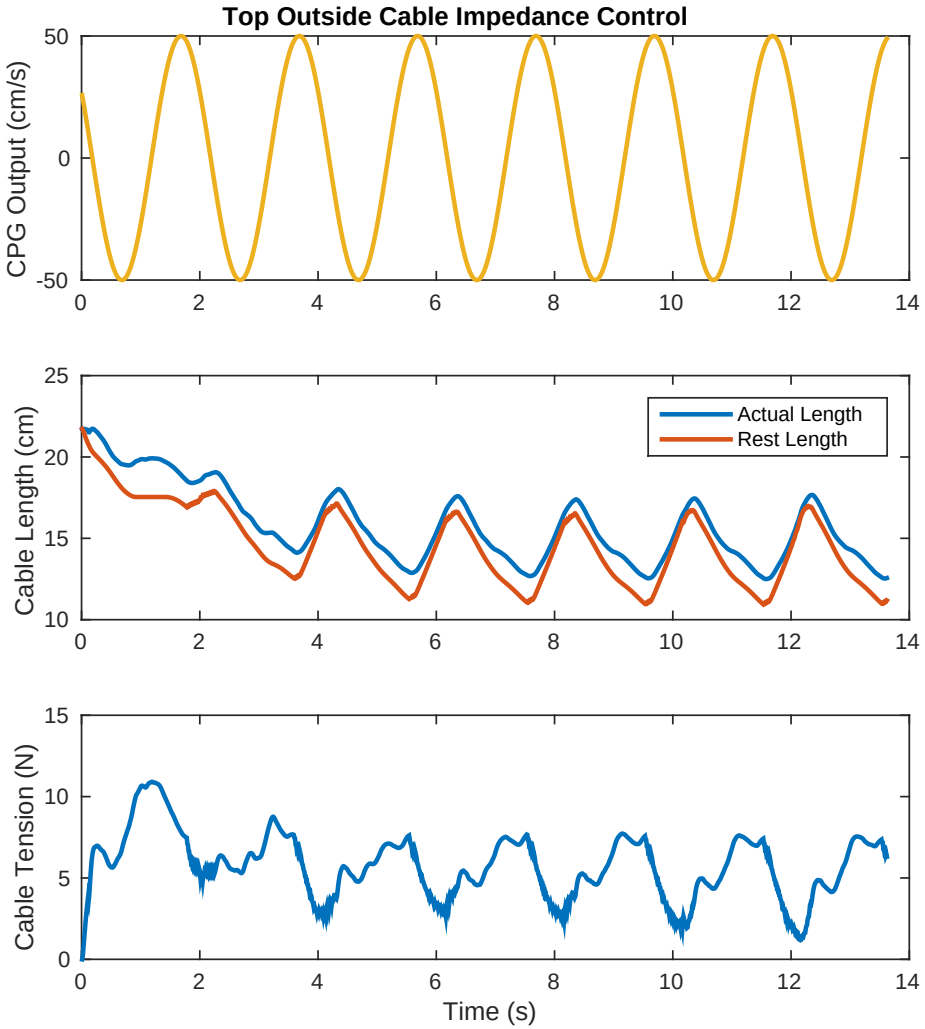

Figure 6.2: A twelve segment Tetraspine (38 cm to a rod) crawling over flat ground. The top cable is shown responding to an impedance controller where $T_0 = 35$ N, $K = 300\ N/m$, and $B = 50\ N/m - s$. The CPG's output specifies the impedance controller's trajectory (V_0). The difference between actual length (L) and rest length (L_0) is the stretch in the cable, which correlates with the tension. The cables take about four seconds to reach the desired length as specified by the impedance controller's parameters. The tension offset is high enough to prevent the cable from going slack, and the high amplitude of the velocity causes a 6.5 cm oscillation in the length of the cable.

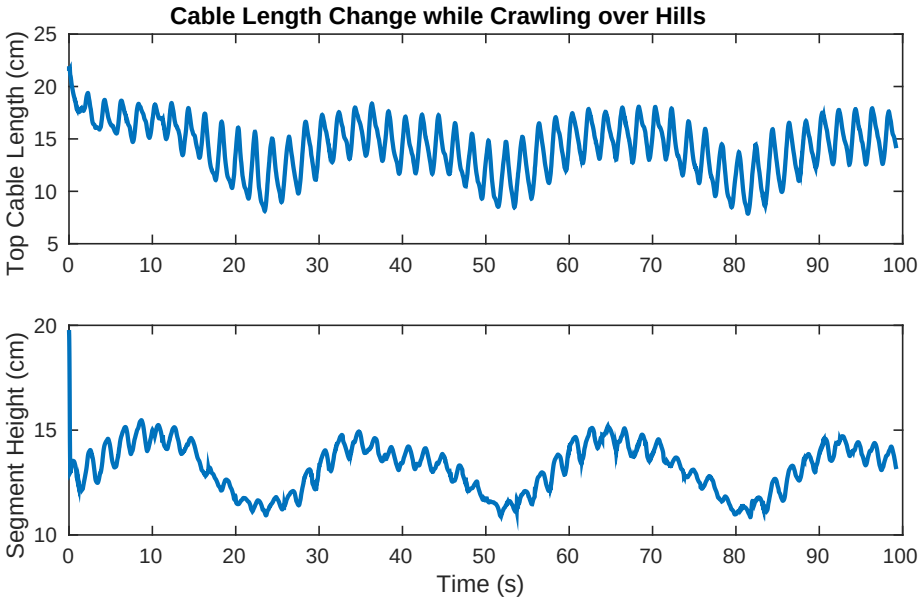

Figure 6.3: Tetraspine (38 cm to a rod) crawling over 5 cm bumps. The impedance controller adapts the length of the cable (top) to the terrain (visible in the height of the segment, bottom), while still tracking the signal provided by the CPG and producing locomotion. On the hills, this robot travels at 0.79 cm/s or 0.2 body lengths per minute.

crawls across flat ground. The hand tuned, sine wave input simulates the output of a CPG, and is sufficient for forward locomotion on flat ground and 5 cm hills (15% of the segment height). The 'step height' (amount each segment is raised during locomotion) is about 1 cm, so whole body adaptations are required for this obstacle. The impedance controller is capable of making the required adjustments for the 5 cm hills, as shown in Figure 6.3. As the robot crawls over the hills, the rest lengths around which the oscillations occur conforms to the terrain. For more complex terrain types we have to add adaptations to the trajectory as well.

In my initial work on Tetraspine, I used the tension setpoint in order to change the shape of the robot for turning and rearing, shown in Figure 6.4. If the impedance controller's tension offset T_0 is higher on the left inside and outside cables than on the right side, we get a curve similar to the top picture in Figure 6.4. Tetraspine will then start turning toward the left. That approach can be reversed for right turns.

Similarly, we can increase the tension in the top outside cables to produce rearing behaviour, which is useful to prepare to ascend an obstacle.

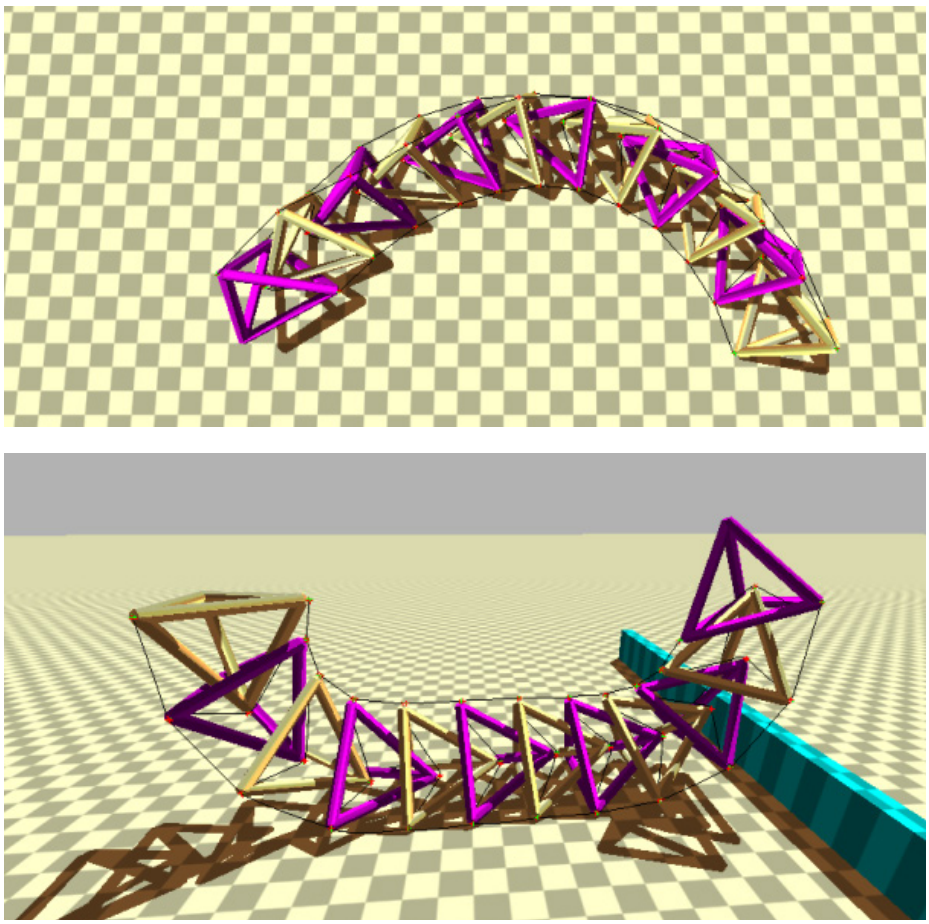

Figure 6.4: Overall body shape is controlled controlled by adjusting cable tensions. Top: Tetraspine executing a left turn. Bottom: The simulated robot rearing before climbing a wall. From [Tietz et al., 2013].

6.2 Central Pattern Generators

As discussed in the preceding section, I used central pattern generators to generate adaptive trajectories for the impedance controller. Due to their relative simplicity and physical meaning of their parameters, the specific CPG equations we chose are amplitude controlled phase coupled oscillators, also used in Ijspeert et al.'s salamander

robot [Ijspeert et al., 2007]:

$$\dot{\theta}_i = 2\pi v_i + \sum_j r_j w_{ij} sin(\theta_j - \theta_i - \phi_{ij}) \tag{6.2}$$

$$\ddot{r}_i = a_i(\frac{a_i}{4}(R_i - r_i) - \dot{r}_i) \tag{6.3}$$

$$V_i = r_i(cos(\theta_i)) \tag{6.4}$$

The phase of the CPG is determined by equation 6.2, where v_i is a frequency term, and coupling is determined by r_j, the amplitude of the coupled node, a coupling weight w_{ij}, and the phase difference between nodes, and a phase offset ϕ_{ij}. The amplitude is determined by equation 6.3, and is constant after it reaches the setpoint R_i. a_i is a positive constant. The final output is on the velocity of the impedance controller, as determined by equation 6.4. v_i and R_i can be specified as one parameter each, or as the combination of an offset, a gain on a descending command, and the command itself [Ijspeert et al., 2007]. Equations were integrated using ODEInt, part of the Boost C++ libraries [Ahnert and Mulansky, 2011].

6.2.1 Hand Tuned Results

Prior to setting up the machine learning architecure, I hand tuned controllers for the tensegrity spines. These fell into two categories: (1) tuning the parameters of the CPG directly, or (2) tuning a set of sine waves (amplitude, frequency, phase differences) as an approximation. The data in Figures 6.2, 6.3, and 6.5 are from a set of sine waves. This gait, with the adaptability provided by the impedance controllers, is sufficient for Tetraspine to slowly traverse hills.

The CPG parameters in [Tietz et al., 2013] were hand tuned. The connectivity of the CPG used each outside cable as a node, and connected the cables to equivalent nodes on other segments. Each node was connected to five other nodes. Inside cables

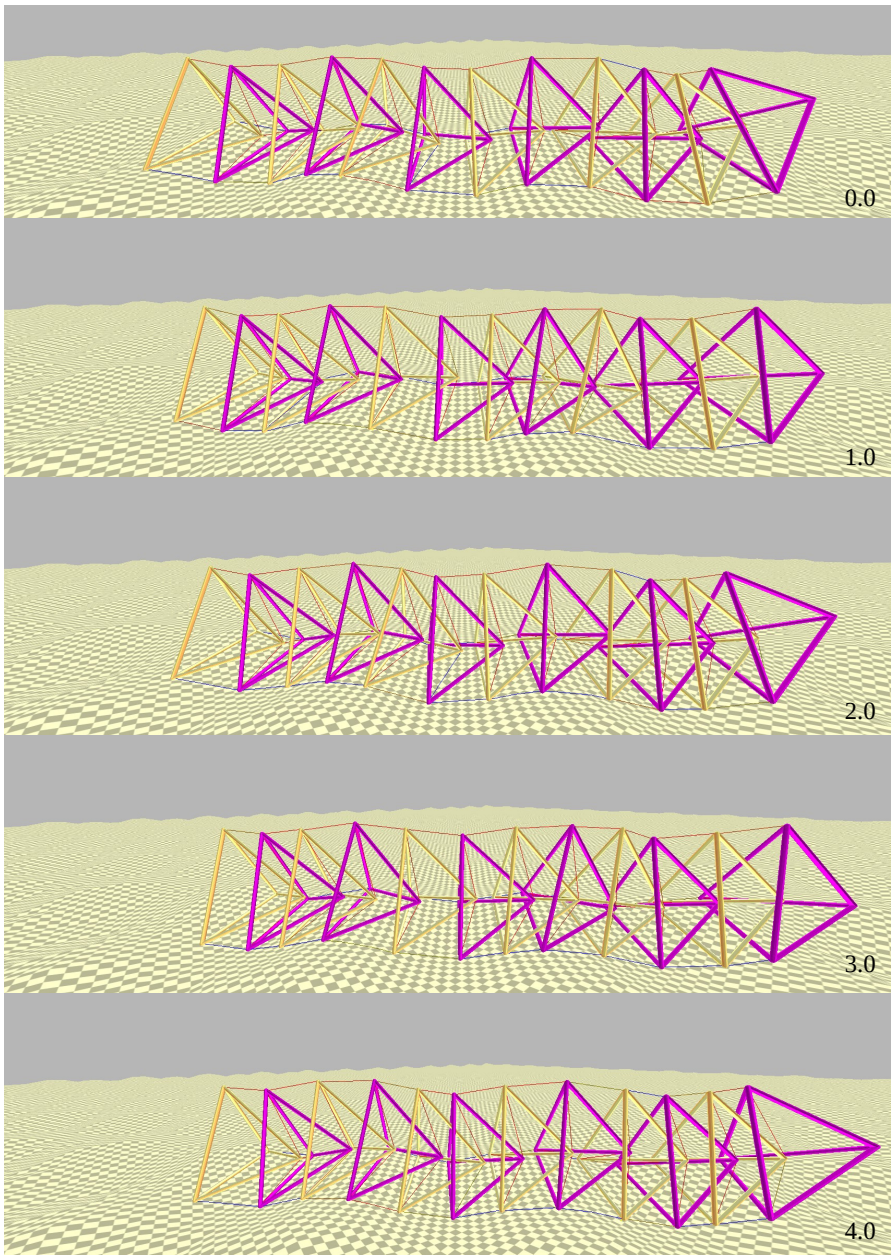

Figure 6.5: A sine wave controller for Tetraspine making slow progress over hills (0.2 body lengths per minute). This is the same controller as Figures 6.2 and 6.3.

had V_0 set to zero. Machine learning determined a sidewinding gait for Tetraspine that was 43% faster than the hand tuned gait with the same CPG morphology, even allowing the unrealistic cables. Thus all subsequent gaits have been determined by machine learning.

6.3 Automated CPG Morphologies

While Tetraspine's morphology had obvious directions for actuation and control, the other morphologies I investigated did not. Since I had already seen that machine learned gaits could be better than hand tuned, I decided to use machine learning to determine the coupling of the CPGs, in addition to the parameters. The coupling of the CPGs (w_{ij} and ϕ_{ij} from equation 6.2) determine how the oscillations coverage to a steady state oscillation, specifying the gait of the robot. The maximum possible coupling would be "all to all," where every node is connected to every other node in the CPG. However, letting machine learning decide the appropriate coupling out of "all to all" quickly becomes intractable, leading to $n(n-1)$ couplings, where n is the total number of nodes. If there are k segments with m nodes each, then this becomes $km(km-1)$. Thus, I developed rules based on the morphology of the spine to reduce the number of possible couplings, and reduce the corresponding machine learning time.

As a starting set of coupling rules, I decided that each node (actuator) would be coupled to nodes (actuators) with shared rigid bodies. This is similar to nearest neighbor coupling, which is the basis for models of swimming and walking networks in the salamander [Bicanski et al., 2013]. Since the rigid bodies are in a chain, this assumption means there are at most three rigid bodies worth of nodes in any repeated coupling set. For example, in Tetraspine with the outside cables forming the CPG, this leads to twenty four connections per segment ($m(3m-1)$), as opposed to the

twelve in the hand tuned version. Couplings are bi-directional, and can have different sets of parameters for each direction. In the case of Tetraspine, this would lead to forty eight parameters for the edges. Therefore, in addition, I specified that symmetric couplings would be the same, consistent with the prior work in [Tietz et al., 2013]. This additional assumption reduces the number of possible couplings to $m(3m+1)/2$. An example with a two node, three segment system is shown in Figure 6.6.

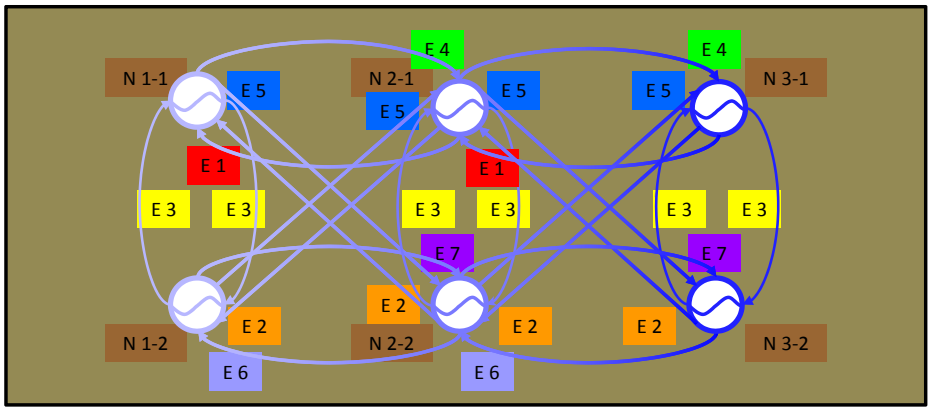

Figure 6.6: The coupling rules used on the CPGs, as applied to a hypothetical two dimensional tensegrity spine with four rigid bars, and six cables connecting them (two per segment). This gives the CPG two nodes per segment, with three segments. When scaled up, this system leads to $m(3m+1)/2$ possible couplings, where m is the number of nodes. In this case, there are a total of ten edges per node, but the edges are sorted by geometry into seven possible edges. These rules reduce the rate of scaling to half of that for specifying each coupling separately. [Mirletz et al., 2014].

Given the CPGs morphologies, I then used the Monte Carlo method for parameter estimation. Trials were scored by the distance the center of mass of the tensegrity spine moved in 30 or 60 seconds of simulation time, depending on computational intensity. After 10,000 - 20,000 trials we took the best sets of parameters and ran Gaussian sampling on them with a standard deviation of 0.005. If a better set of parameters was found, sampling continued around the new best set. The resulting best parameters are discussed below. On a desktop computer purchased in 2011 (Intel Core^{TM}i7 at 3.4 GHz), this search procedure takes 24 - 48 hours.

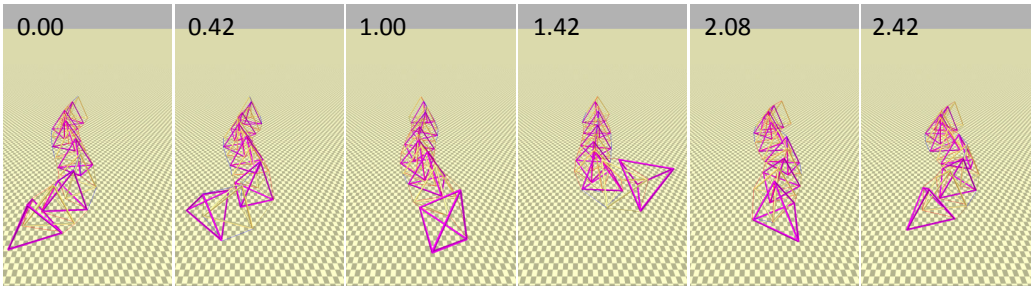

Figure 6.7: Tetraspine executing a form of sidewinding, determined by machine learning. The vertex of the rear segment functions as a tail pushing the robot to the right, unlike my hand tuned initial work where we treated it as the head [Tietz et al., 2013]. Only the outside cables were included in the CPG, the velocity setpoint for the inside cables was zero. The time in seconds at each step is shown in the upper left corner. From [Mirletz et al., 2014].

6.3.1 Results on Various Morphologies

For each morphology from Chapter 5 we recorded the best machine learned gait, its speed, and its cost of transport (COT). Note that these gaits have been exclusively optimized for speed, COT is currently only for comparison.

Our first attempt with this parameter estimation paradigm used the same CPG as our initial Tetraspine work, with four connections per segment. We also maintained the original coupling weights (0.5). This CPG morphology resulted in nine total parameters for estimation, as we left amplitude and frequency offsets and gains separate in this case since the equations were set up for descending commands, as in [Ijspeert et al., 2007]. However, commands were held constant throughout the trials.

The resulting sidewinding gait[1] travels at 9.46 centimeters per second with a COT of 3.2. Scaling our prior results, this is 43% faster than the hand tuned gait of [Tietz et al., 2013] (6.6 cm/s).

Six key frames of video for this gait are shown in Figure 6.7. A second gait, similar to the hand tuned gait in [Tietz et al., 2013] but traveling in the opposite direction, traveled at 7.78 cm/s with a COT of 2.4. Both of these gaits contradicted

[1]Video of the tetraspine sidewinding is available at `https://www.youtube.com/watch?v=I-7jCGT-WKk`

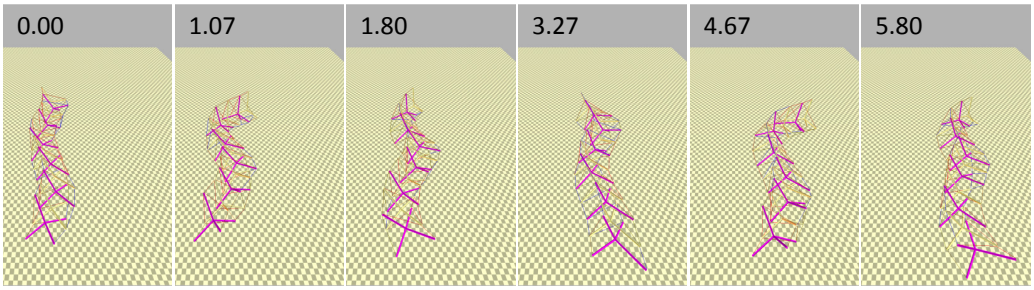

Figure 6.8: The tetrahedral complex based spine also learned a gait similar to sidewinding. Large twists of the body are possible since all eight cables are actuated, moving the robot to the right and down. Other gaits (not shown) included rolling. From [Mirletz et al., 2014].

our intuition for hand tuning the robot, where we assumed based on the shape, that the end segment with the extra vertex would function as a head. Machine learning indicates it is more effective as a tail.

As simulated, one rod of the tetrahedral complex measured 17.3 cm. The best gait for the 'tetrahedral complex' spine, shown in Figure 6.8, was another form of sidewinding[2], and is the first to use the automated morphology algorithm (all following controllers use it). This gait moved at 6.13 centimeters per second, with a COT of 2.13, the lowest cost of transport of all four morphologies. A lower COT for sidewinding relative to other gaits matches results from biological snakes [Secor et al., 1992].

For the Octahedral Complex, the gait in Figure 6.9 utilizes one rod of each segment as a leg in a walking like gait with a speed of 4.6 centimeters per second, and a cost of transport of 3.4. A similar gait[3] which was slightly slower (3.2 cm/sec) had a cost of transport of 2.6. These differences illustrate the common tradeoff between speed and efficiency in locomotor systems.

Parameter estimation for the ribbed spine returned two gaits of interest. One, pictured in Figure 6.10 is a crawling motion similar to slithering. The fastest gait[4]

[2]Video of the tetrahedral complex sidewinding is available at `https://www.youtube.com/watch?v=Wyt7B7-OebI`

[3]Video of the lowest COT gait for the octahedral complex based spine is available at `https://www.youtube.com/watch?v=Wk9x_40xWjM`

[4]Video of the gait pictured in Figure 6.10 is available at `https://www.youtube.com/watch?v=`

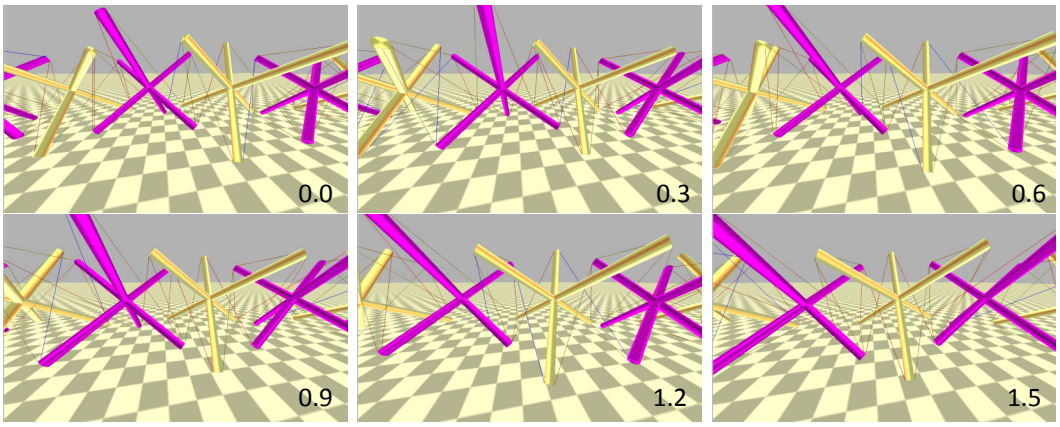

Figure 6.9: When on a flat surface, the octahedral complexes have one contact per vertebrae. The learned gait uses these contacting rods as legs and alternates sides in a walking pattern. While the robot mostly moves to the left, the motion is not purely two dimensional, the segments alternate slightly away from the central axis. In this plot phase of the CPG can be seen in the color of the cables, red indicates high tension cables, blue indicates low tension. This is also true in the previous plots, but the cables were typically too small to see. The time in seconds at each step is shown in the lower right corner From [Mirletz et al., 2014].

traveled at 5.37 centimeters per second, with a cost of transport of 6.0. While this gait is neither the fastest, nor most efficient, it has the advantage that the 'head' segment stays relatively stable during locomotion. This stability provides a potential advantage for sensors such as cameras or laser range finders, used in higher level navigation behaviors.

j0UXr59C9mY. The fastest gait can be viewed at https://www.youtube.com/watch?v=PL22dswN2RA

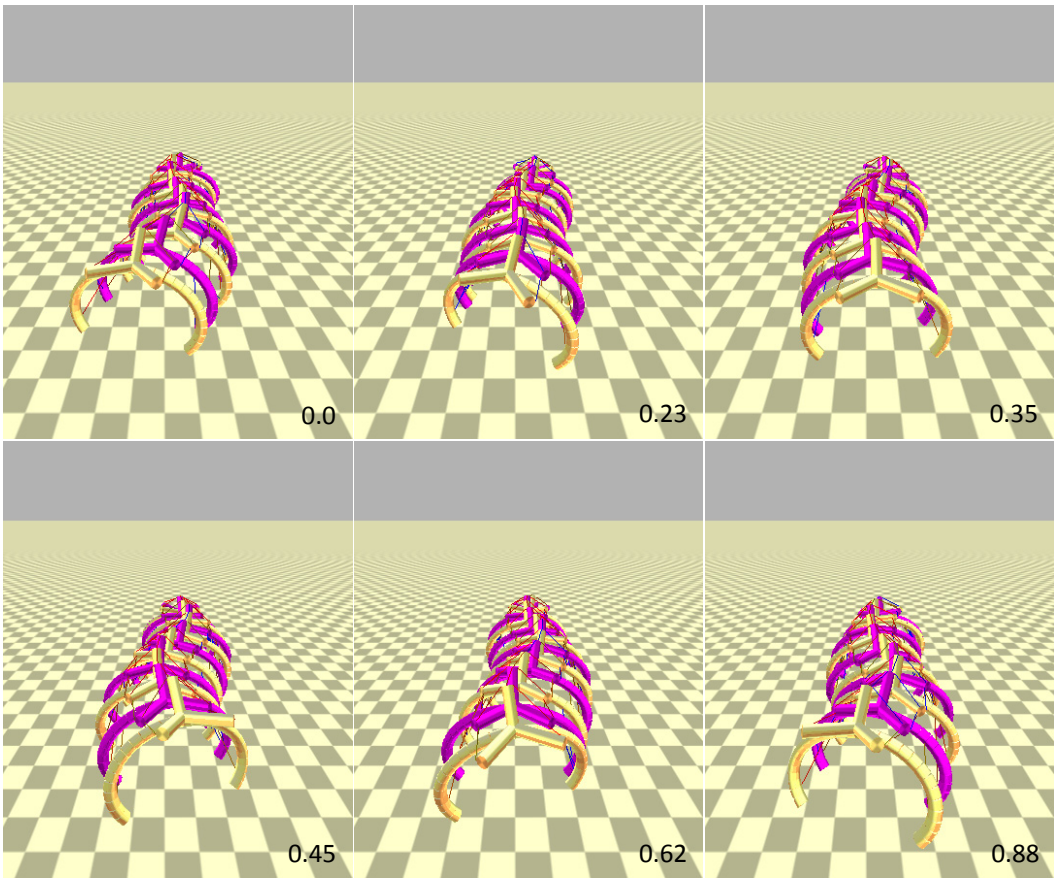

Figure 6.10: The second fastest gait of the ribbed tensegrity is shown. This gait exploits turning the body left and right to move forward (down on the page), similar to lateral undulation.

6.4 Feedback for CPGs

Once I developed the contact dynamic modules for the cables, I was able to test the spines on rough terrain. In order to make things realistically difficult, I also added the brushless DC motor model at this point. As expected, the spines quickly became stuck in some terrains, particularly the block field. While the impedance controllers can generate some adaptations (as in Figure 6.3), the resulting gaits are extremely slow. Thus, I looked for ways to incorporate environmental feedback into the CPG as well, so the trajectory can be adapted to rough terrain. Additionally, properly specified feedback allows the oscillations of the CPG to synchronize with the structure and the environment.

The specific CPG equations we use are a combination of the adaptive phase-coupled oscillator equations of Righetti et al. [Righetti et al., 2006] (feedback on the frequency) and Gay et al. [Gay et al., 2013] (feedback on the amplitude and phase):

$$\dot{r_i} = \gamma(R_i + k_r F_r - r_i^2) r_i \tag{6.5}$$

$$\dot{\theta_i} = \omega_i + k_\theta F_\theta + \sum_j r_j w_{ij} sin(\theta_j - \theta_i - \phi_{ij}) \tag{6.6}$$

$$\dot{\omega_i} = k_\omega F_\omega sin(\theta_i) \tag{6.7}$$

$$V_i = r_i(cos(\theta_i)) \tag{6.8}$$

where r_i is the amplitude of the wave, ω_i is the frequency, θ_i is the phase, and V_i is the input to the impedance controller. The amplitude (equation 6.2) is set by convergence parameter γ, and setpoint R_i. The phase relates to connected nodes through weight w_{ij}, phase offset ϕ_{ij}, and the other node's amplitude r_j in equation 6.3. The terms k_r, k_θ, k_ω are scalar gains on feedback functions F_r, F_θ, and F_ω, which, similar to [Gay et al., 2013], are the outputs of an artificial neural network. The network contains two inputs (tension and length), four hidden nodes, and the

three output functions to the CPG. Each neuron contains a sigmoid function, and is the weighted sum of nodes in higher layers. The same network weights are used for each cable's feedback function. An updated control diagram is shown in Figure 6.11. Note that due to differences in r and ω, parameters need some additional tuning to transfer from the previous CPG implementation.

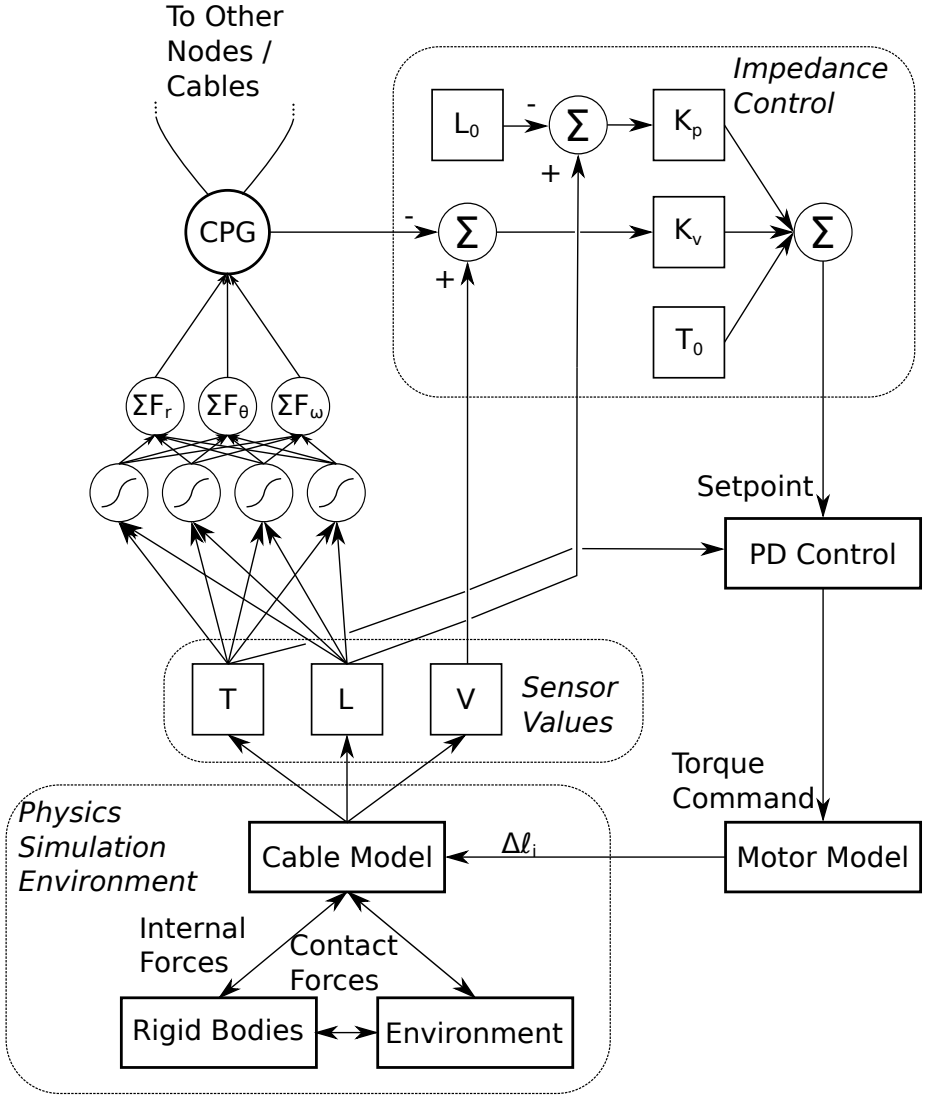

Figure 6.11: Feedback to the CPG is provided through an artificial neural network (center). The same network is re-used at each cable.

In order to determine a controller for rough terrain, we tuned and tested the tensegrity spine on three types of terrain: flat ground, sinusoidal hills with an am-

plitude of 2 cm, and a field of 500 randomly placed blocks as described in Chapter 4. Imagining the length of one rod as a 'leg' this means the hills are 20% of the leg height (blocks are 5%), and 98% of the flat ground step height. Trials were evaluated according to distance traveled in 60 simulation seconds. If multiple terrains were used, scores were averaged.

Given the large number of parameters, we started tuning our system by selecting the best gaits for an open loop CPG on flat ground from 24,000 Monte Carlo trials, with the new equations and a shorter spine (6 segments for computational efficiency). The eight best were selected for a Gaussian sampling hill climbing optimization on all three terrains. The hill climbing step improved the results between 100% and 200% over Monte Carlo, but most of the improvement was on flat ground, indicating the need for feedback to the CPG (the feedback gains were zero during the preceding steps).

To parameterize the artificial neural network, we used a genetic algorithm with crossover, mutation, and elitism. The population consisted of 60 members, the best 15 of which survived to the next generation. Fitness was again determined by average score between the three terrain types (hills, blocks, flat ground). Evolution for the feedback parameters ran for 14 generations (1,935 trials), the CPG parameters were held constant during these trials.

6.4.1 Improvements on Rough Terrain

Once the feedback functions were tuned, we compared the performance of the closed loop CPG controller to our previous open loop configuration, impedance control was used in both cases. The differences between the two CPG signals are shown in 6.12 when the feedback function is activated mid trial (10 seconds). The feedback decreases the frequency and increases the amplitude of the signal. Flat ground and hilly terrain results were both deterministic, and are summarized in Table 6.1, block field results

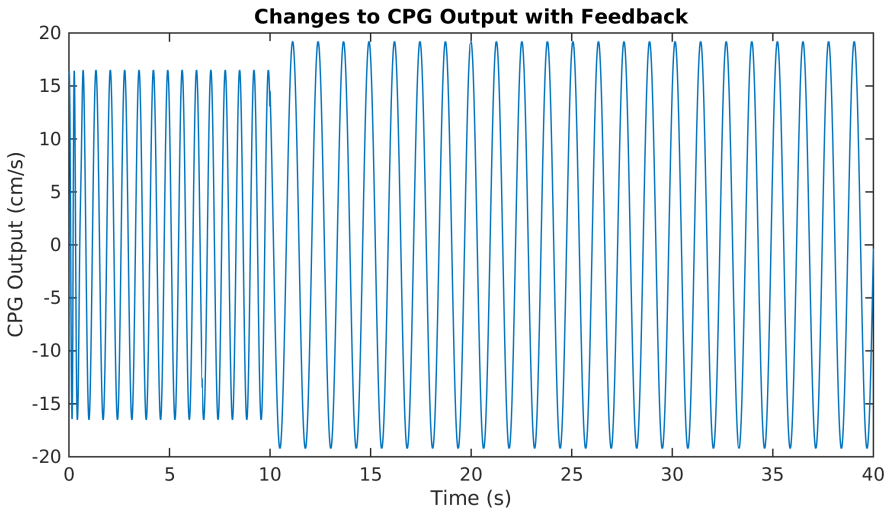

Figure 6.12: The feedback functions are activated at 10 s while a six segment tetrahedral complex spine craws across flat ground. The CPG equations are able to smoothly integrate this new information. The result is a lowered frequency but increased amplitude.

are summarized in Figure 6.14. While for this gait, feedback slows locomotion on flat ground by about 1%, the robot's ability to handle rough terrain improves significantly. This morphology and controller move 7 times faster on hills than Tetraspine with sine wave inputs and impedance controllers, relative to its size (1.36 body lengths per minute).

Table 6.1: Distance traveled in 60 seconds with deterministic terrain types

Terrain	**Without Feedback**	**With Feedback**
Flat Ground	492.8 [cm]	487.1 [cm]
2 cm Hills	30.4 [cm]	105.3 [cm]

To determine the robustness of these results, we set up twenty different block fields and tested the robot's performance with and without feedback, using the same random seed for each controller[5]. Motion on the blocks is shown in Figure 6.13. Since the robot starts at the center of the square block field, if it traveled at least 150 cm, it is guaranteed to have 'escaped' the block field. The controller with feedback moved

[5]Video: `https://www.youtube.com/watch?v=94yqYPUYjH0`

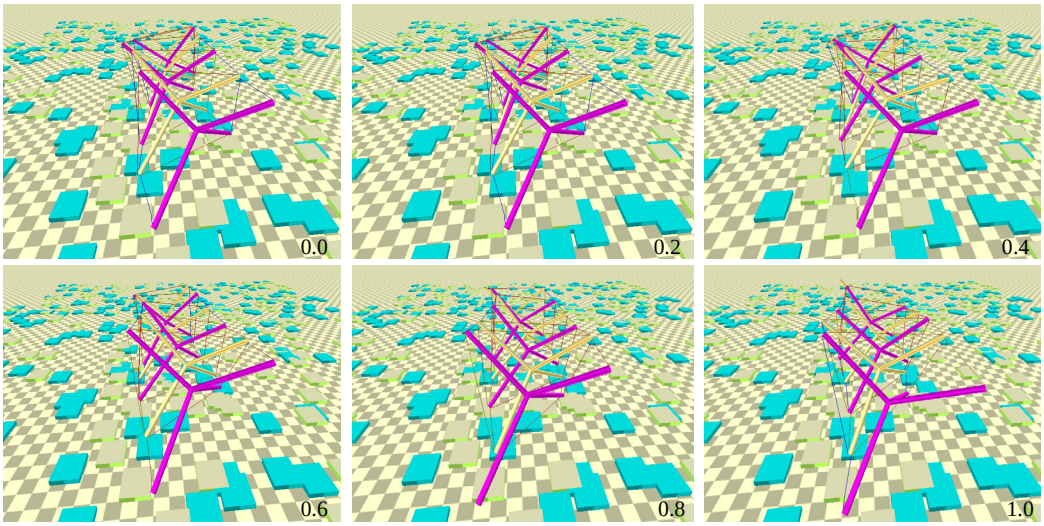

Figure 6.13: A motion sequence of the tetrahedral complex crawling over blocks with the feedback control system. This spine is 77.5 cm long.

further than the open loop controller in all cases, and 'escaped' in 17 out of 20 trials.

Performance details are shown in Figure 6.14.

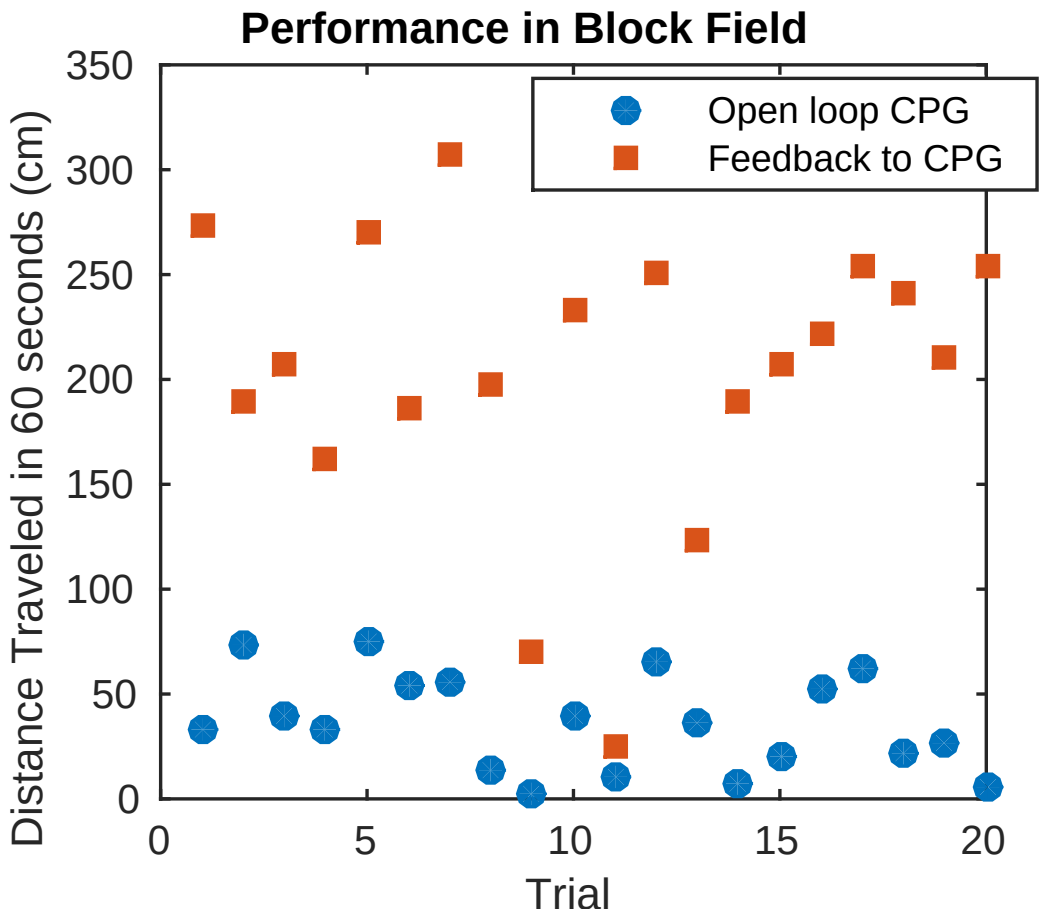

Figure 6.14: Tests of locomotion on a random block field with and without feedback at the CPG level, using the same random seed. From [Mirletz et al., 2015b].

6.5 Goal Directed Controllers

Up to this point in the work, the controllers have been scored by their distance traveled in any direction. However, the ideal robot can be sent in a specific direction with simple descending commands. Thus, I investigated methods for steering the robot towards a goal location.

Figure 6.15 shows results from the final controller with only the feedback controller from the previous section, no goal direction. The robot tends to move perpendicular to its length, in a clockwise arc. The robot's path changes on each terrain type (flat ground, block field, and hills). Thus, the ideal controller will be able to direct the robot to the same locations across multiple terrains.

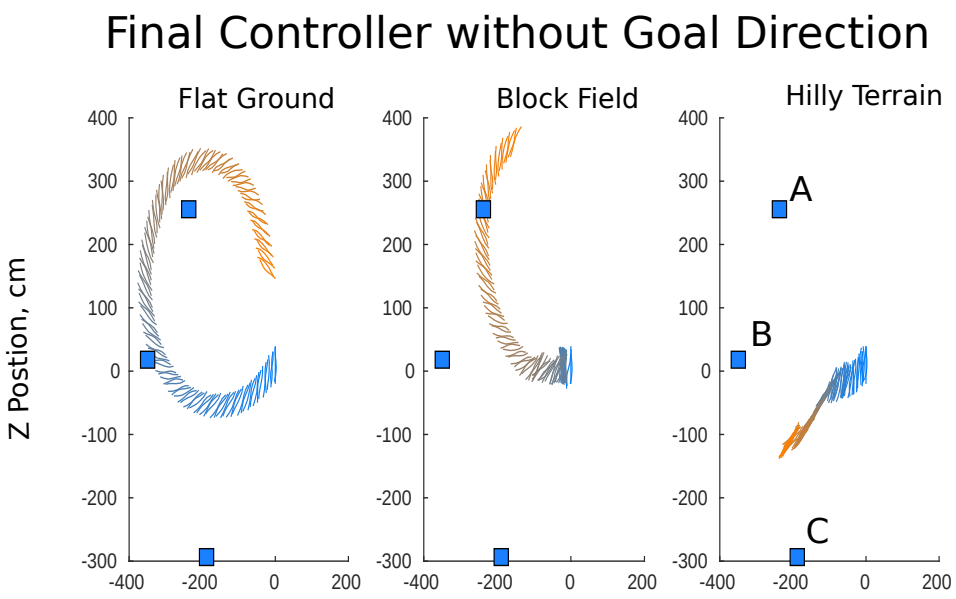

Figure 6.15: The final controller on three terrain types without goal direction input. The three square blocks indicate the possible goal positions during training (the CPG, feedback function, and impedance controller were all optimized for this task during learning). The small lines indicate the positions of the center of mass of each segment of the robot. Color indicates the timestep, with blue at 0 s, fading to red by the end of the trial at 120 s. With no goal direction, the robot finishes in three different positions after 120 s.

One of the challenges for learning a goal directed controller is the choice of objective function. I used the objective function of distance moved towards the goal. The goal was represented by a large rigid body in the simulation, so I could compare the center of mass of the robot's 'head' with the goal position. This means that not moving is better than moving away from the goal, which can cause CPGs that don't move to be favored over those that do. Machine learning required careful control over the seeding of controllers, the choice of initial goal location, and the complexity of trials.

My first attempt was to use a neural network mapped from the goal direction to the same CPG feedback functions as the feedback network. I used the same network for each segment, so this network had two inputs, 24 hidden neurons and 24 outputs. This worked for a single goal, but had trouble generalizing to multiple goals and multiple

orientations of the robot. Thus, I added two more inputs for the robot's orientation (4 total). In essence, this is similar to having a compass heading, and a compass heading of your destination, both of which update over time. In hardware, a compass and GPS could provide this information, or any high level mapping controller.

While this was successful on flat ground, interference between the two types of feedback equations prevented it from being successful on rough terrain. Thus, using inspiration from the early work on Tetraspine, I changed the goal direction network to map from the existing inputs to the tension gain on the impedance controller. In essence, this technique uses the neural network to generalize the modifications I was making to Tetraspine's tension setpoints by hand (as in Figure 6.4). The resulting network has four inputs, eight hidden layer neurons, and eight outputs.

The 'seed' for the goal directed controller was the rough terrain controller from section 6.4. Based on the natural direction of motion of the rough terrain controller, I trained the goal directed controller to move toward a goal roughly straight to the left, 350 cm away (-x in Bullet's coordinates, location B in Figure 6.15). I started evolution just tuning the goal direction controllers from a random seed, with a population of 80, running each trial for 120 sec to give the robot sufficient time to get to the goal on hilly terrain. After running this overnight (15 generations), I took the best controllers and attempted to co-evolve a full system. Ultimately, for this particular controller, centralized evolution of the full system proved more effective, and a controller capable of a single goal on hills and flat ground (only trained on hills) was trained in 45 generations. Generalization to multiple goals (A, B, and C) took an additional 26 generations of training on two additional goal positions, placed 350 cm away in different directions. The controller also trained on flat ground with the same locations in this final stage, scores were averaged across these six trials.

Figure 6.17 shows the results on hills for the three trained positions. Due to the geometry of the goal block and the robot, if the center of mass of one of the robot's

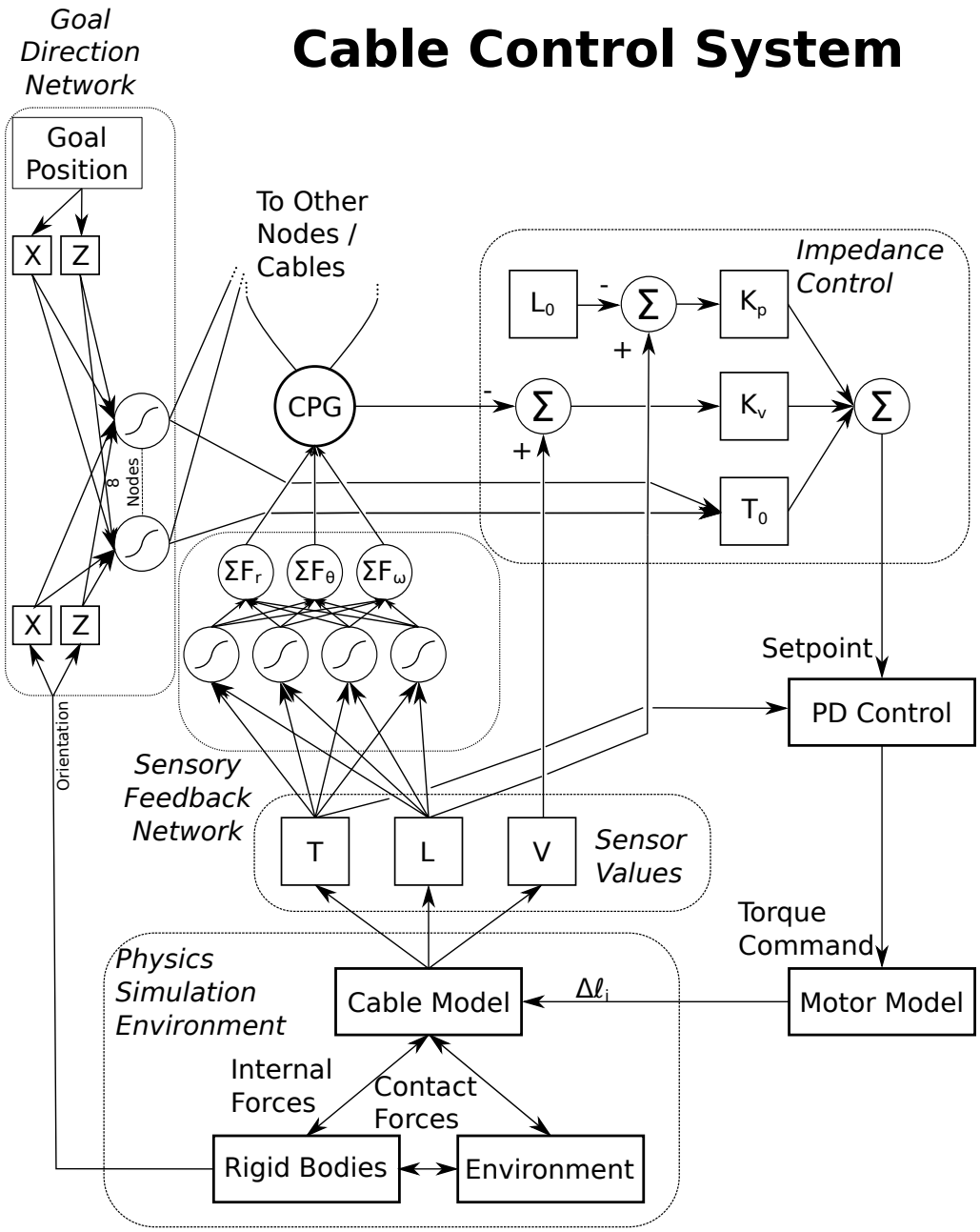

Figure 6.16: The final control system for a single cable. The new component is an artificial neural network on the left side of the figure which maps the goal location and the robot's orientation to the tension input of the impedance controller. The network has four inputs, eight hidden nodes, and eight outputs: the tension setpoint of each cable's impedance controller. Connections between nodes are weighted linear functions, which sum into a sigmoid function. Each final output node is another sigmoid, which combines the outputs of the hidden nodes. The same network is used for each segment, resulting in the goal directed feedback providing commands like 'tense all of the top cables.'

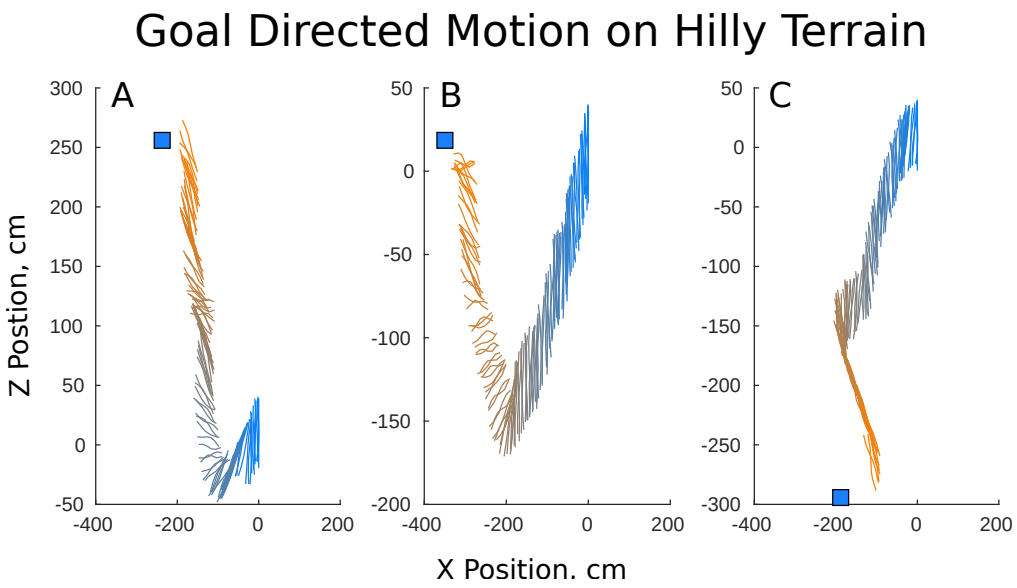

Figure 6.17: Goal directed motion on hilly terrain. The goal position is indicated with the large blue box, and the center of mass of each of the robot's segments is indicated with the colored lines. The color fade again occurs from 0 to 120 seconds.

segments passes within 20 cm of the goal block's center of mass, the robot will contact the goal block and achieve the maximum score. This distance is slightly larger on the hills, since the goal block falls over. The robot reaches the goal at the location B (center) on the hilly terrain, for location A the robot comes within 20 cm, and for the location C it 'misses' by 60 cm.

The geometry of the hills forces the robot to move differently than it would on flat ground. This is because moving perpendicular to the robot's length causes it to cross the hills. In particular, with the goal at location C, the robot spends a noticeable portion of the trial moving along its length, rather than perpendicular to its length. The goal directed controller is just under two times faster than the undirected controller of the prior section, covering 201.9 cm in the first 60 seconds of the middle trial.

The goal direction is more precise on flat ground. Comparing Figure 6.15 with Figure 6.18, the shape changes yielded by the goal direction network change the radius of the arced path taken. For flat ground and on the blocks, the robot was tested on an

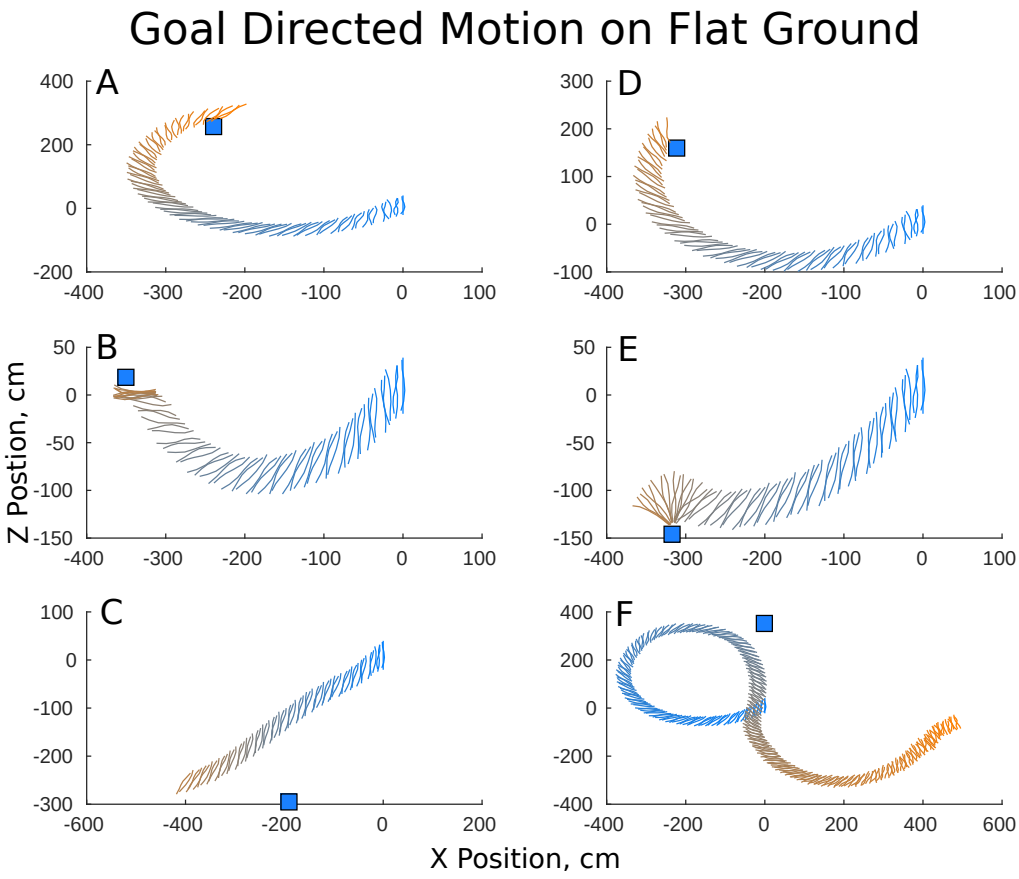

Figure 6.18: Goal directed motion on flat ground. I tested on three additional untrained locations (D, E, and F) for flat ground and the blocks. For the first five plots (A through E), blue is 0 s, and red is 82 s. For location F (lower right, red is 267 seconds. The robot contacts the goal block in four out of six locations, two of which were untrained.

additional three goal locations on which it was not trained (locations D through F). In these trials, it hit the goal block in two trained locations (A, B) and two untrained locations (D, E). Some of the time the robot will hit the goal block before the end of the trial. The goal block is heavy enough that hitting the block stops the robot's continued motion, as in the trial with location B in Figure 6.18 (middle left) and location A (top left) in Figure 6.21. While the robot misses the goal in the bottom two trials (C and F), the longer run time of location F's trial (bottom right) shows that with the proper inputs, the goal direction controller can also produce counter clockwise turns. This controller displays its fastest motion on flat ground, moving up

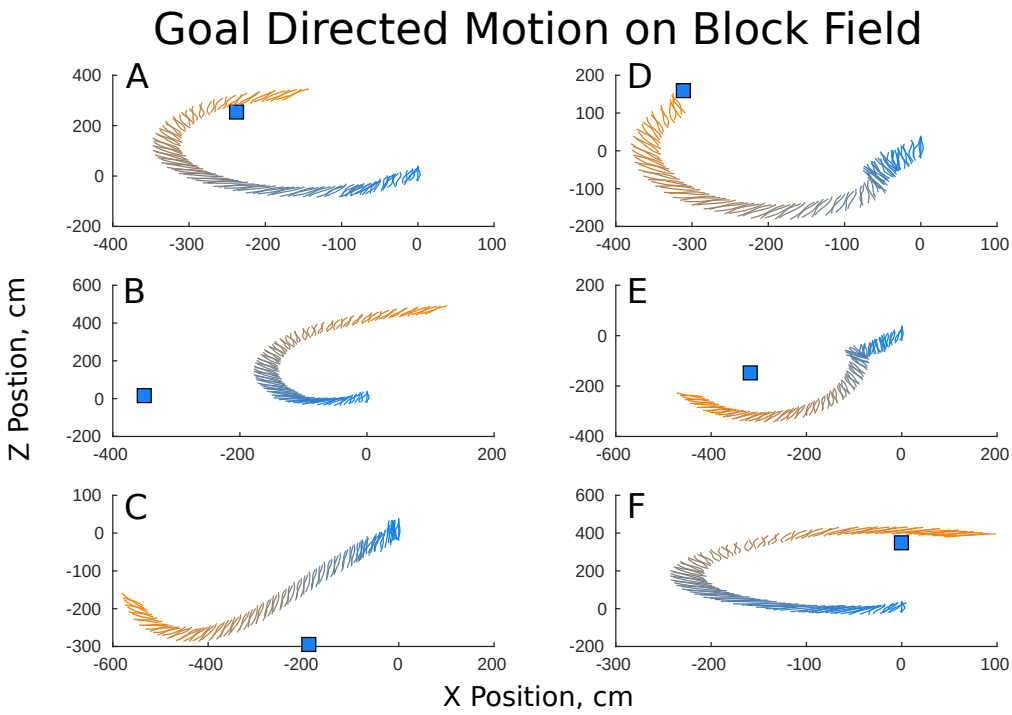

Figure 6.19: Goal directed motion on a block field. All six trials used a block field with the same random seed. The robot contacts the goal in three out of six trials. The trial length was 100 seconds in all cases.

to 501.34 cm in 60 seconds. However, the behaviors designed for rough terrain led to a high cost of transport of 11.2. Note that the DC motor model makes more accurate assumptions about energy use than prior work, such as including the applied torque when lengthening.

The block field was particularly difficult due to the random placement on the blocks. To avoid over-fitting to a particular block field I left them out of the learning entirely, as deterministic objective functions converge more quickly. My hope was since the hills were larger than the blocks, if the robot was capable of performing passably precise paths, it would be able to move off blocks and then to the goals.

Between the training for motion over rough terrain, and training in the block field in section 6.4, the robot was able to 'escape' in all twelve tests of the goal directed controller (six locations, two block fields). Note that using the goal direction network enables the robot to 'escape' the block field 28 seconds faster than the controller with

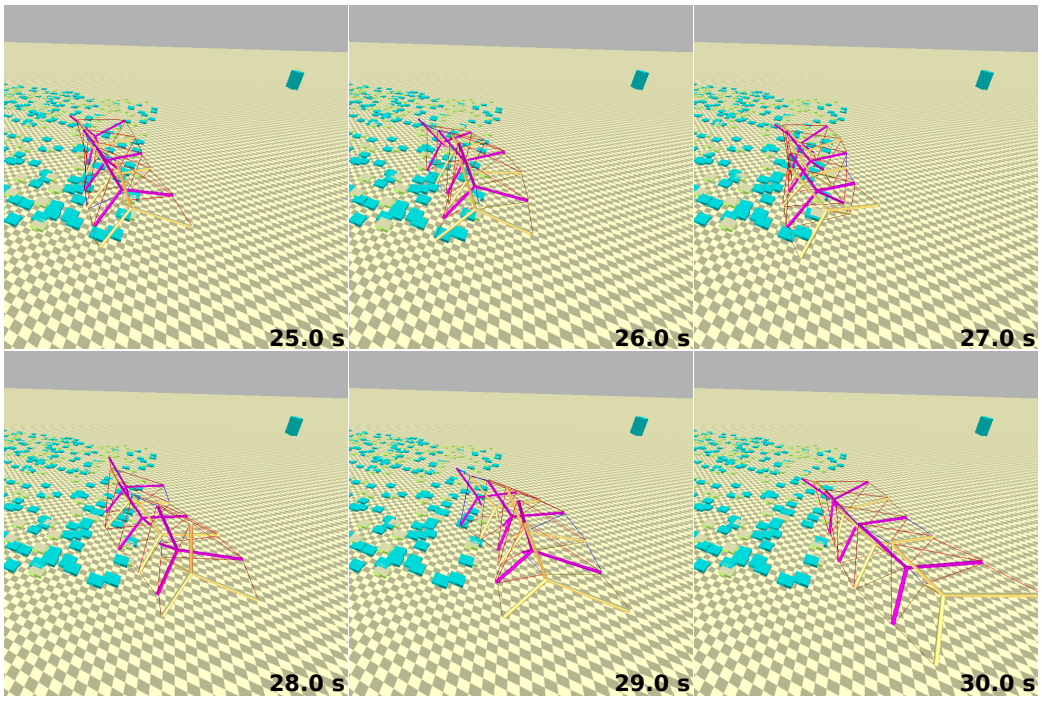

Figure 6.20: Images from 25 s to 30 s of the trial with location A on the blocks (top left of 6.19). The robot successfully departs the blocks, and then crawls on flat ground to the goal (the large block in the upper right corner of each image).

no goal information. Six locations with a single block field are shown in Figure 6.19, and the same six locations with a different block field are shown in Figure 6.21. The robot contacts the goal block in six out of twleve trials. Each goal block set produces a success in one of the two cases where the controller missed on flat ground (C and F). Thus, the controller reaches all six goal locations in at least one trial.

Tests on sloped terrains (global ramps) showed that with this controller the simulated robot was capable of climbing up to 7.5 degree slopes, and can reach the goal on small slopes of 2 degrees.

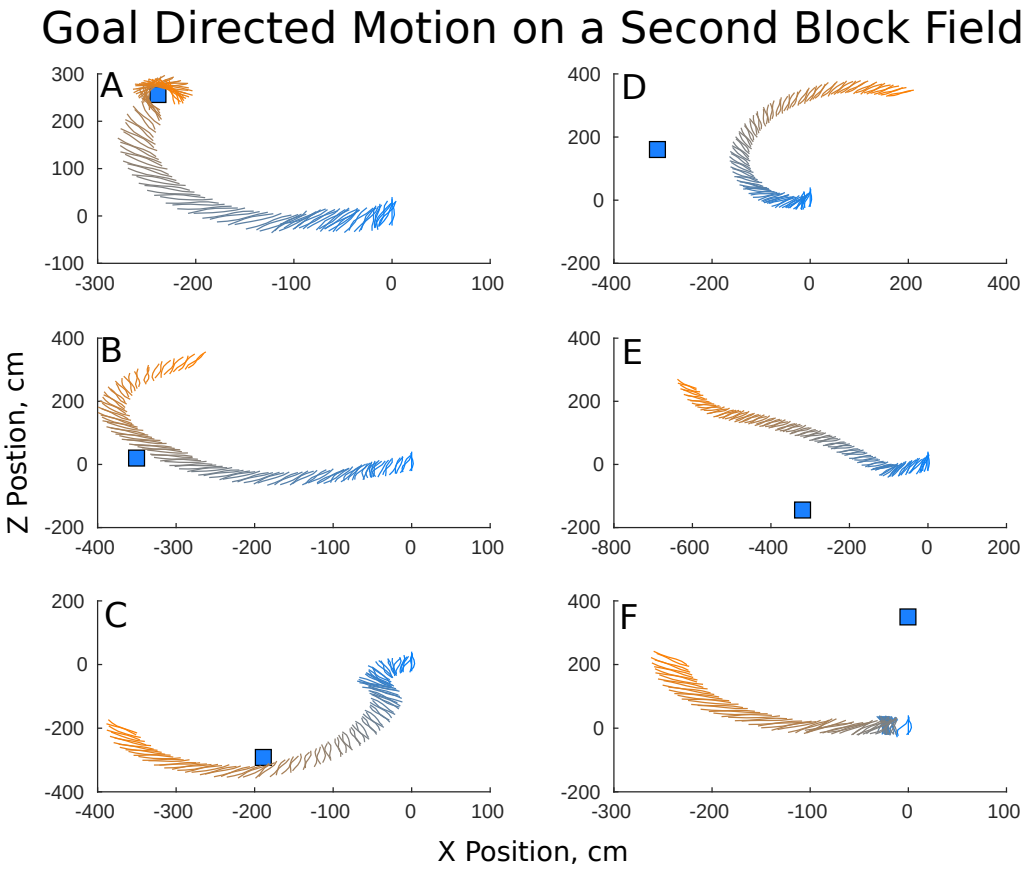

Figure 6.21: Goal directed motion on a second block field. All six trials use the same random seed, which was different from Figure 6.19. The robot reaches the location A again, as well as locations B and C (left side).

Chapter 7

Robot Hardware and Simulator Verification

In order to ensure the prior results could be realized in robotic hardware, I conducted hardware trials on two versions of Tetraspine, each of which consisted of three segments. The original version of Tetraspine was constructed by the senior project team at the University of Idaho in 2012. Dr. In-Won Park upgraded this robot to make it capable of locomotion during 2013, but due to a lack of force sensors it was only capable of position control. I constructed my first version in the CWRU Biorobotics Laboratory with assistance from Ross Carnahan and Richard Bachmann [Tietz et al., 2013]. Based on his experience with the University of Idaho Tetraspine, Dr. In-Won Park designed a new implementation and built Tetraspine 3 at NASA Ames, and we performed validation tests together using its high quality load cells as force sensors [Mirletz et al., 2015a]. These validation tests complement those performed in [Caluwaerts et al., 2014], where the positions of an icosahedron tensegrity robot (a prototype planetary lander) were validated for one step of a rolling motion to be within 15 mm on a robot with 1 m struts. This section discusses the construction of each Tetraspine prototype, followed by the validation trials that showed NTRT can

be used to design future robots with reasonable safety factors.

7.1 Carbon Fiber Spine

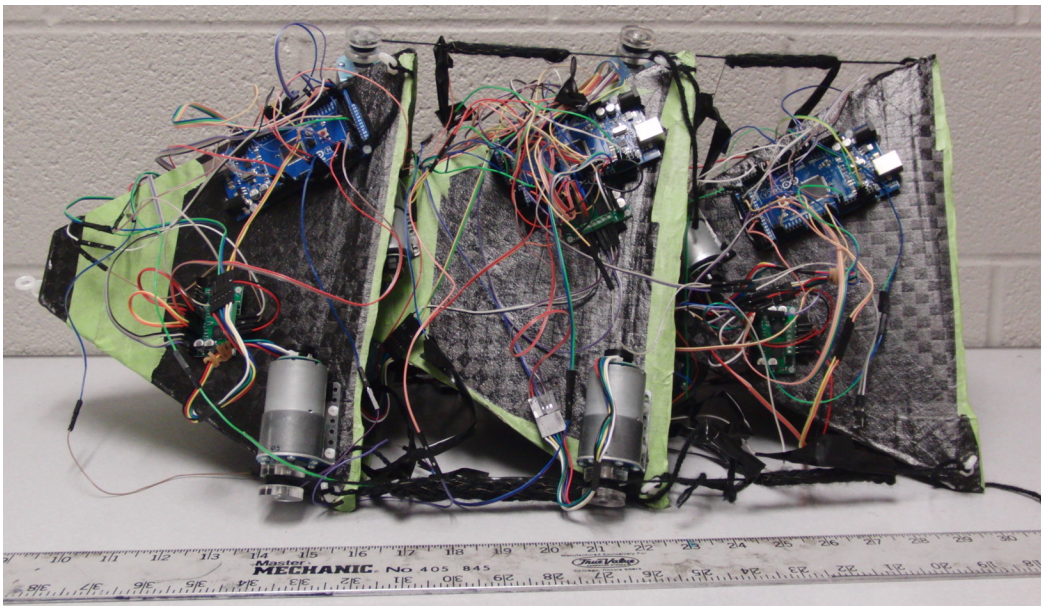

Figure 7.1: The three segment carbon fiber Tetraspine robot, circa April 2013. One Arduino Mega microcontroller controls each segment. Four out of the twelve strings are visible; the units on the ruler are inches.

The version of Tetraspine Ross Carnahan and I constructed in the biorobotics lab provided a proof of concept for the controller I developed in simulation, but was not able to provide sufficiently accurate data for simulator verification. The robot is pictured in Figure 7.1. The carbon fiber segments of the robot measure 28 cm to a 'rod' (along the diagonal from the front to the rear veracities) and were manufactured in the CWRU Biorobotics Additive Manufacturing Lab. Ross constructed a wooden mold of the appropriate dimensions, and then wrapped it with Teflon, carbon fiber, perforated Teflon, and finally a cotton outer layer, before placing everything in a vacuum bag and baking it in the carbon fiber oven for two hours. After the piece cooled and was removed from the mold, we drilled holes in the resulting shell and

mounted the motors and electronics using screws and nuts. I later sealed the edges of these holes with superglue, as carbon fiber is conductive.

The segments are connected with strings that are spooled and unspooled with Pololu 12V 50:1 DC motors with encoders. The inside cables were routed through nylon fisheye screws[1] to bring their line of action closer to the tip of the robot. Each segment of the prototype is controlled by an Arduino Mega R3. The outside segments each have a mass of approximately 0.9 kg without batteries, while the middle segment's mass is 1.6 kg.

7.1.1 Stretch Sensors

Tension in the strings was a major piece of sensory data in the simulator. Many off-the-shelf stretch sensors are too elastic for the forces we needed to apply through the strings, rubber sensors will easily stretch to 50 or 100% of their rest length. More accurate load cells were not within budget for this robot. Thus, we opted to use in-line stretch sensors by knitting conductive thread with yarn, an example is shown in Figure 7.2.

The sensors were constructed using a circular loom. Each end of the conductive thread was attached to a soldered loop of wire with a knot, after the thread was looped around the wire several times. The length of the knit portion was varied based on the application of the stretch sensor.

This creates a resistance that changes non-linearly, for example, a 6 cm sensor changes from 47.6 Ohms to 25.4 Ohms over a 2 cm stretch. The stiffness was also non-linear; the first cm of stretch accumulated 0.7 N of force, while the second cm accumulated an additional 11.3 N. We placed 68 Ohm current limiting resistors between the sensor and the ground, after burning out an Arduino by going over its current limit. The final circuit is shown in 7.3. We ran 5 Volts over these sensors and after

[1]These screws would almost always bend, and occasionally break.

Figure 7.2: A knit stretch sensor at slack length. The conductive yarn is silver, the actuator yarn is black. Units on the ruler are centimeters. From [Tietz et al., 2013].

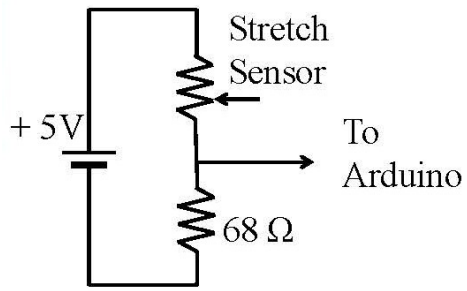

Figure 7.3: A diagram of the sensor and current limiting resistor.

the analog to digital conversion had a resolution of approximately 200 units over the 2 cm length change.

While these sensors were ultimately sufficient for control, they required extensive calibration, which was never directly convertible into units of force. In addition, the knots in the knitting would slip, changing the amount of resistance. Finally, the conductive yarn was fairly thin and prone to breaking, meaning the sensors had to be replaced fairly often.

7.1.2 Carbon Fiber Spine's Performance

With proper calibration, the three segment spine demonstrated forward locomotion at a speed of 7.6 cm per minute. Stills from a one minute video of this motion are shown in Figure 7.4. This result provided a qualitative, visual match of simulator behavior.

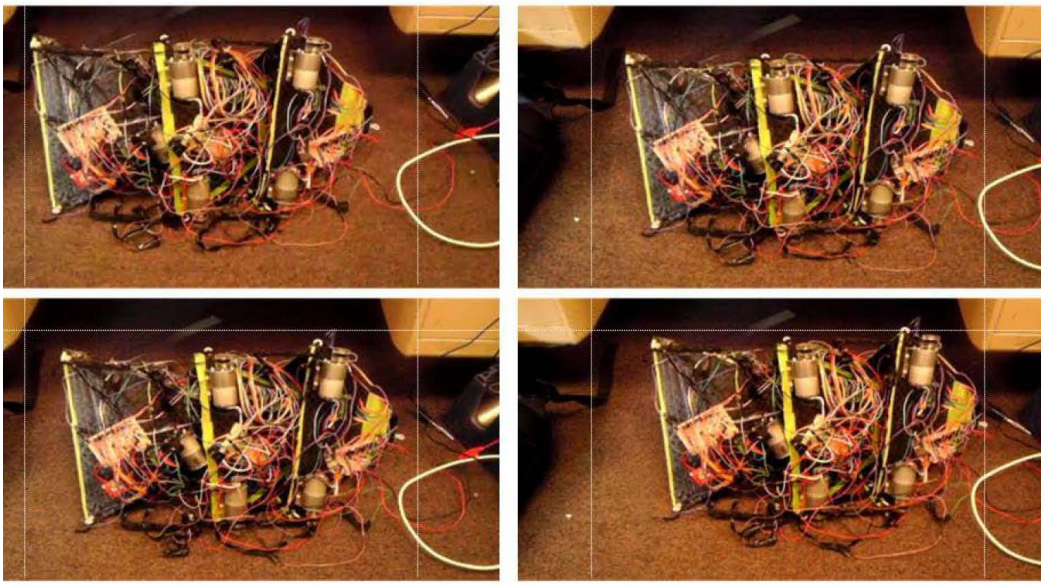

Figure 7.4: Motion of the Carbon Fiber spine on lab carpet. These images represent approximately one minute of motion.

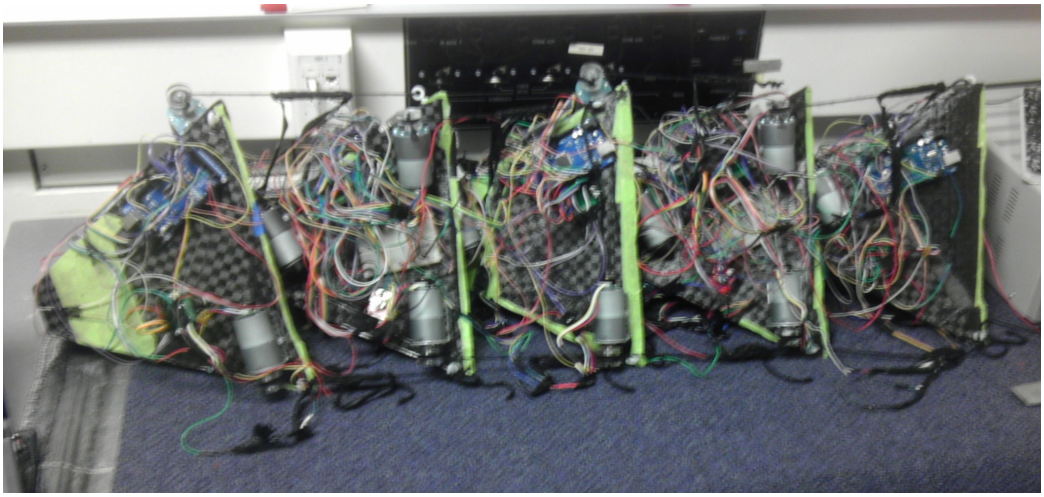

Figure 7.5: The five segment version of the Carbon Fiber spine. Ultimately the forces required for movement of this spine were beyond the strength of the knit stretch sensors.

I attempted to extend this version of Tetraspine to five segments, but this was ultimately unsuccessful due to the limitations in the stretch sensors, especially under the higher forces required by the longer spine, shown in Figure 7.5.

7.2 Tetraspine 3

In order to perform simulator validation tests, Dr. In-Won Park constructed the Tetraspine3 hardware at NASA Ames Research Center. Figure 7.6 shows a snapshot of the Tetraspine3 hardware, which has 12 DC motors, 12 vectran cables, and 12 load cells. The length and diameter of Tetraspine3's rods is to 38.10 cm and 0.64 cm, respectively. One end of each vectran cable is connected to a DC motor (Faulhaber 1524) with a spool for rotation. The other end is attached to a 222 N (50 lb) tension and compression load cell (Load Cell Central LCC-CTD) for measuring cable tension. For 8 out of 12 cables, a 227 N/m spring is placed before the load cell to increase compliance. Since the other four cables (top inside and top outside) required additional stiffness to support the weight of the robot, they did not include springs.

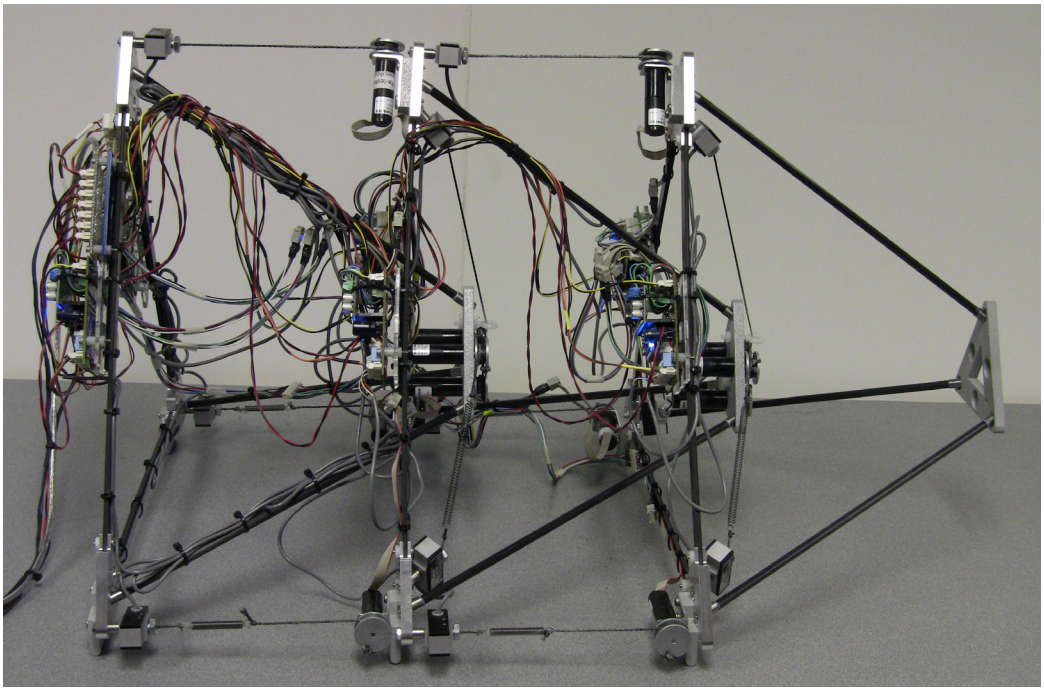

Figure 7.6: Snapshot of TetraSpine3 hardware (the first segment is the farthest right). From [Mirletz et al., 2015a].

The Tetraspine 3 prototype used carbon fiber rods to form the tetrahedrons, which simultaneously reduced the weight compared to the University of Idaho aluminum

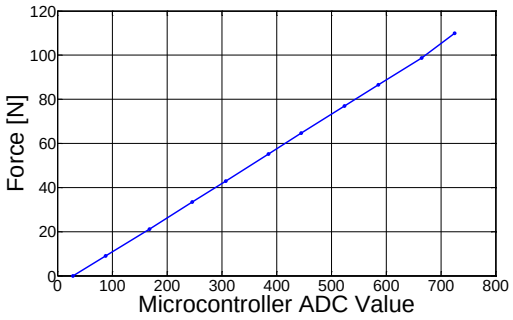

Figure 7.7: The load cell calibration. From [Mirletz et al., 2015a].

spine, and increased strength relative to the carbon fiber shell of the CWRU spine. Table 7.1 compares the mass of all three prototypes. Note that middle segments have six motors, whereas outer segments have three.

Six Arduino Nano slave boards (two per segment) control the motors via a Pololu MC33926 motor driver, a LTC3260 voltage regulator and two instrumentation amplifiers. The instrumentation amplifier increases the voltage range of the load cell before it is connected to the Arduino's 10 bit ADC. Figure 7.7 shows the output of the calibrated load cell, where the maximum measurable tension is 111.2 N (25 lbs). The following linear relationship describes the resolution of the load cell in terms of ADC bits.

$$f = 0.1567(ADC) - 4.9363 \tag{7.1}$$

A PC and the six slave boards communicate using fast-mode I^2C protocol (400 kHz) through a Devantech USB-ISS communication module. Every 10 milliseconds, the PC broadcasts a packet to all six slave boards simultaneously, which contains

Table 7.1: Mass comparison between the three prototypes

Segment	**Carbon Fiber Spine**	**TetraSpine2**	**TetraSpine3**
First	0.9 [kg]	1.45 [kg]	0.73 [kg]
Middle	1.6 [kg]	1.54 [kg]	0.91 [kg]
Last	0.9 [kg]	1.18 [kg]	0.73 [kg]

all 12 desired motor commands. Within 20 milliseconds, the PC then connects to each slave board in order to receive the actual and target tension values, the motor encoder value, and the motor PWM output value. During experiments, the process of broadcasting the packet to all six slave boards and receiving the packet from all six slave boards takes less than 3 milliseconds. Every 200 milliseconds, the PC also stores all feedback values from all six slave boards and displays them in a GUI.

The microcontroller supports both position and impedance control modes, which are selected from the PC. The communication packet contains the desired motor angles in position control, whereas it represents the desired cable velocity during impedance control. Position control is used initially to set the length of each cable. When impedance control is switched on, the current length and tension values are used as the L_0 and T_0 parameters in (6.1). In both control modes, the motor is controlled with a PID controller every millisecond (1000 Hz), where the measured loop times of position control and impedance control are 400 microseconds and 800 microseconds, respectively. Filtering the tension sensor, and calculating the cable length and velocity add the additional time to impedance control. The tension offset, the rest length, the stiffness gain, and the velocity gain in (6.1) can be set differently for each cable from the PC.

7.3 Simulator Validation

With this new robot, we were now able to directly compare the forces experienced by the robot with the simulator. We chose to focus on force for two reasons: first, the impedance controllers' output is force, so this provides a good test of the control system. Second, force is a key property in the design of new structures and selection of actuators. By ensuring the simulator's forces are in range with the hardware, we now have confidence in future simulated designs.

We started by updating the simulated version of Tetraspine to match, as closely as possible, the material properties of the robot such as cable stiffness, inertia, and friction. To verify these properties, we used a quasi-static tests of dragging the robot at a steady velocity and applying a 20 N force to the front segment of the robot. Once we were confident in the match between the simulation and hardware, we tested a trajectory, which was hand tuned on the robot, in the simulator. Finally we ran a simulator learned trajectory for a different gait on the robot. These simulations were run at 1000 Hz. Decreasing the timestep to 250 Hz changes the predicted force on this test by 6% (worst case), increasing the time step to 4000 Hz changes the predicted force by 1.3%. Since the contact cables were not necessary for this simulation, 60 simulation seconds takes between 7 and 14 seconds of real time, depending on the simulation timestep.

7.3.1 Quasi-static

We started by attempting to calibrate the simulation to the hardware and determine properties such as dynamic friction by measuring the force to drag the robot at a steady velocity. We then applied a steady state disturbance of approximately 20 N (a forward force on the front of the robot, and a restoring force on the rear two segments), and verified that the control response was similar in both cases.

Table 7.2: Control parameters used in Fig. 7.8

Value	**Top Cables**	**Other Cables**
K	500 [N/m]	100 [N/m]
B	100 [N/m-s]	100 [N/m-s]

The results of this test are shown in Fig. 7.8. The overall system dynamics are similar, with the added tension from the disturbance being split between the outside cables. When tuning for this test, we found that even without springs in the top

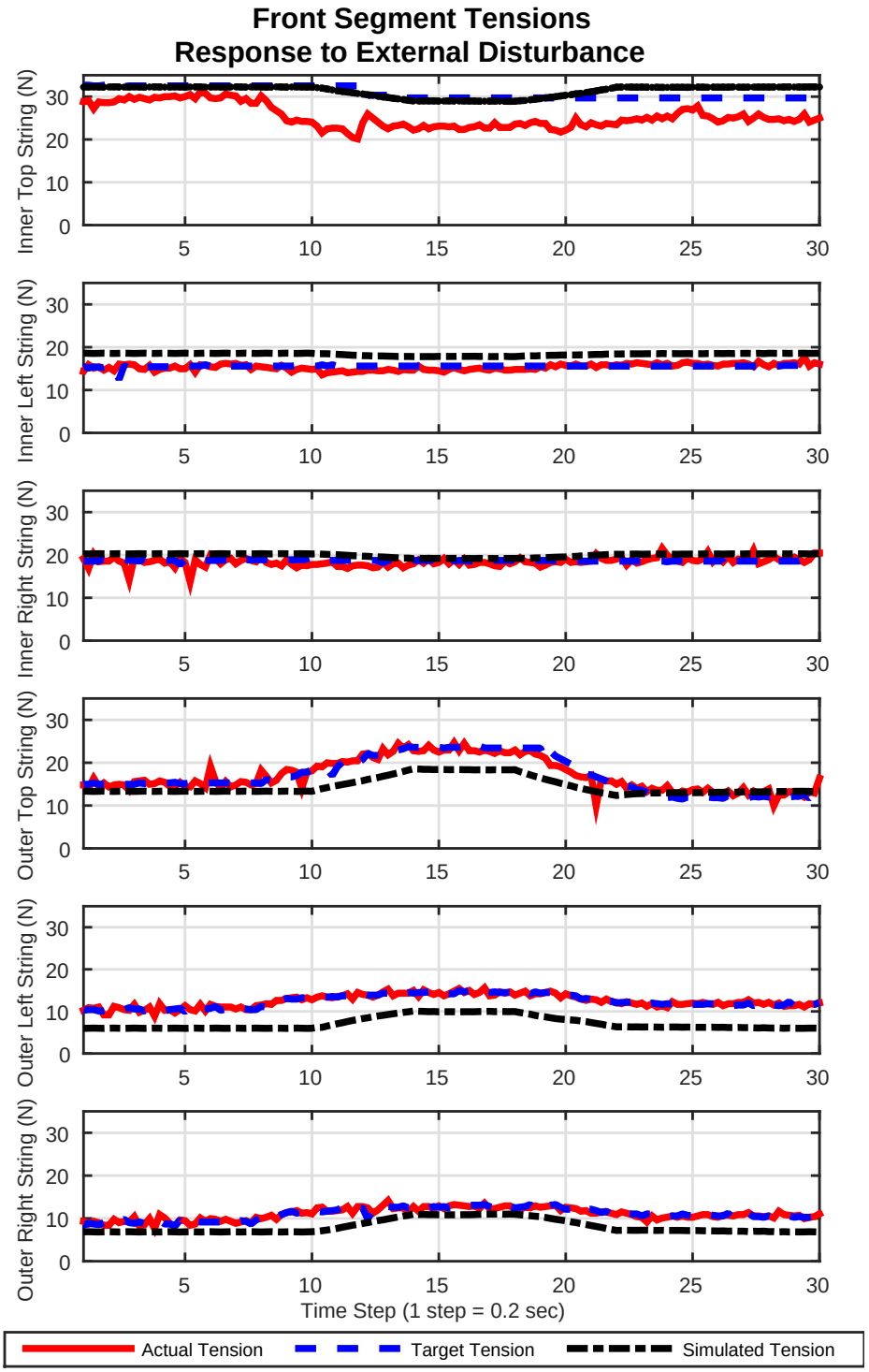

Figure 7.8: We tested the steady state responses of the control by applying a steady load of approximately 20 N (measured using a hand-held scale) to the front segment of the robot while holding the other two in place. The resulting displacement and added tension causes the impedance controller to increase the tension setpoint, mostly in the outside cables. The forces experienced in each cable are plotted, with the hardware's setpoint in blue, its actual tension in red, and the simulator's predictions in black. The simulator's error on the system's maximum tension is within 6.1% (top plot). (from [Mirletz et al., 2015a])

outside and inside cables, it was effective to model them at the same stiffness as the rest of the system (227 N/m), we refer to this as the 'effective stiffness' of these cables.

7.3.2 Hand Tuned Trajectory

Subsequently, we hand tuned a sine wave as the velocity input to the impedance controller on the hardware. The final tuning resulted in the inside cables having the inverse sign of the outside cables, and the rear cables having a phase offset of π from the front cables.

The velocity input in this case required some additional scaling. Amplitudes given to the simulator were 1/10 what they were in the hardware (with identical input, the simulated robot's segments would slide together, causing it to topple). The results of this test (Fig. 7.9) show some differences in the system's pretension levels, but similar sizes of oscillations for each cable. This confirms that some of the actuator properties such as effective stiffness and actuation speed were sufficiently accurate. We suspect the differences are due to the absence of perfect sticking from Bullet's default Coulomb friction model, as the segments tend to slide together under higher pretension. Another difference is sensor noise, which is present in the tension readings (red line), but not in the simulator data (black line). These experiments show the control is robust to this level of noise.

7.3.3 Simulator Generated Trajectory

In order to ensure that we have a useful model, as opposed to a fit of existing data, we chose to make a prediction and test it on the hardware. For our prediction, we used our tuning methods in simulation to generate a trajectory through an open loop CPG capable of sideways motion, similar to the side-winding gait for the twelve segment version [Mirletz et al., 2014]. We exported these values to a text file and then used them as the velocity input on the hardware, without additional scaling. The resulting

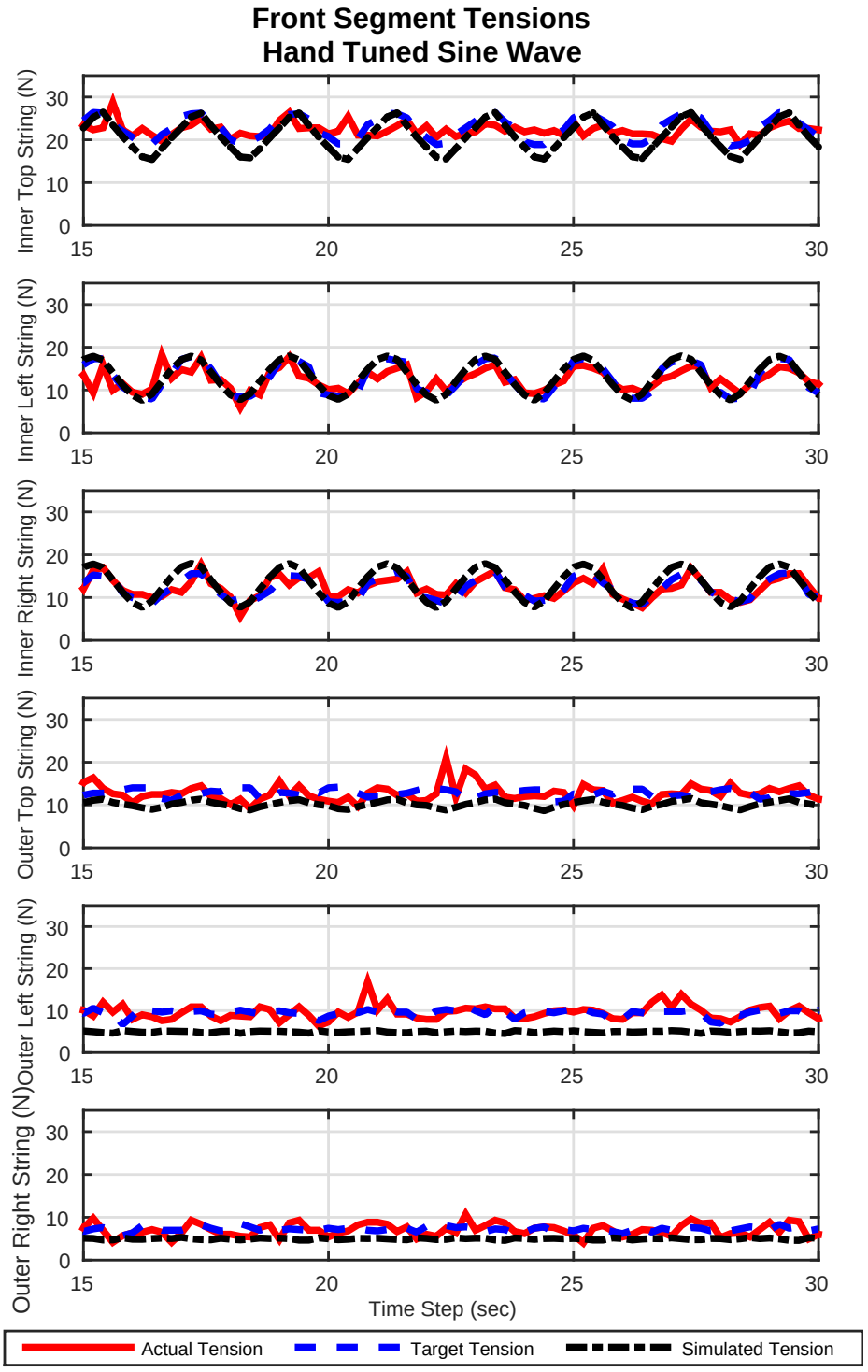

Figure 7.9: A comparison of the tensions experienced by the hardware and simulation under a hand tuned, sine wave input on the velocity term of the impedance controller. This figure shows the full trial, with the trajectory applied to the simulation the entire time, but on the hardware it is only active between timesteps 70 and 170 (14 and 34 s). The simulated maximum system tension in this case is within 7.9% of the hardware. (from [Mirletz et al., 2015a])

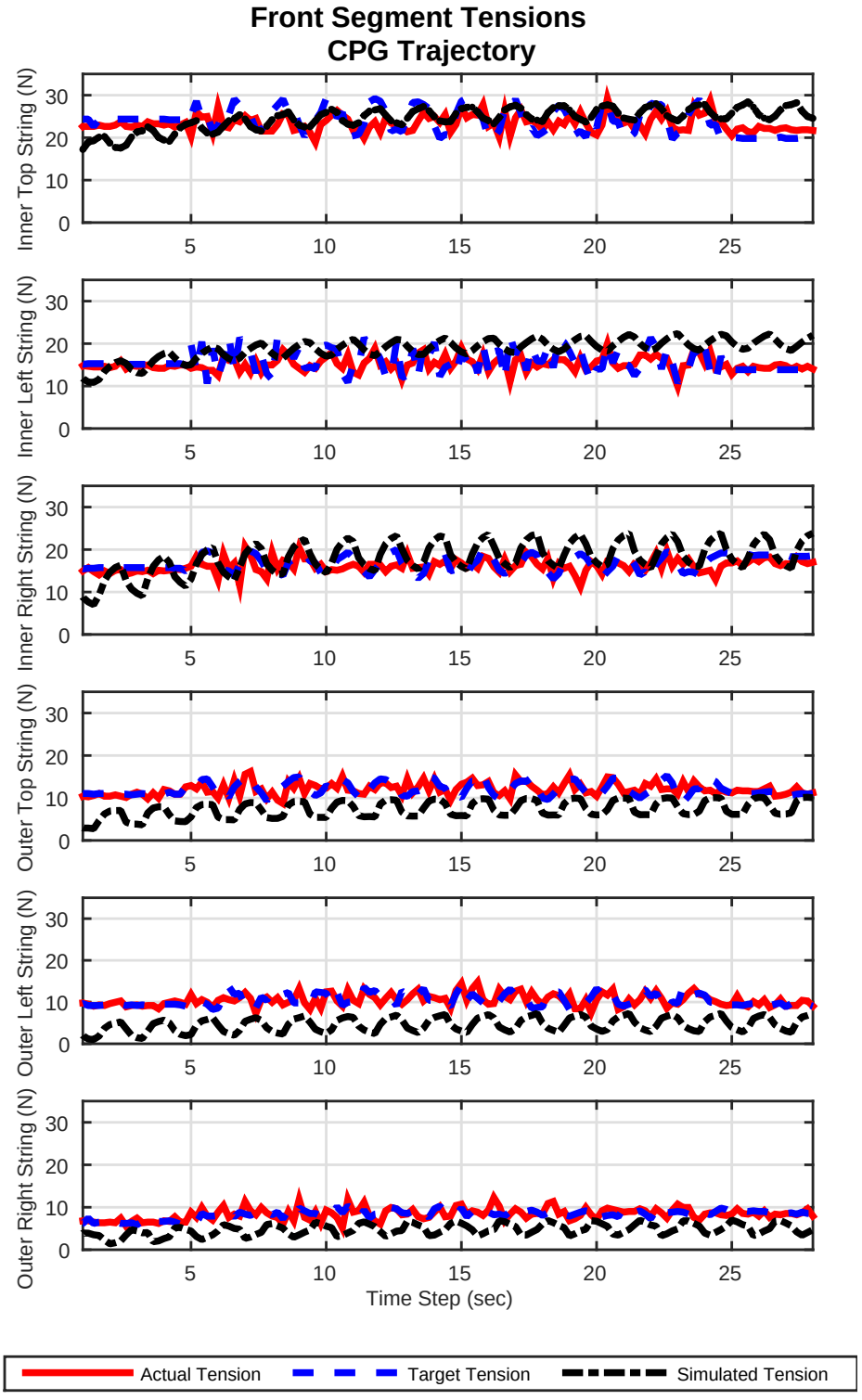

Figure 7.10: A comparison of forces experienced under a simulator-tuned CPG based gait. Like the sine wave, this shows the full trial, with the hardware's trajectory activated later than the simulator's. The maximum tension experienced by the robot is within 1.6% of what was predicted by the simulator. (from [Mirletz et al., 2015a])

forces are shown in Fig. 7.10.

The gait produced successful locomotion on the hardware implementation[2]. Note how the amplitude of the CPG's velocity trajectory forces the inside cables to a more saturated triangle-wave like oscillation in both cables (sharper peaks), while the outside cables are more sinusoidal, verifying the output of the impedance controllers to this input. Again, the pretension is lower in the outside cables, and the speed of locomotion in hardware is slower due to additional static friction and unmodeled cable friction, but the overall direction of locomotion is similar.

We found that the forces predicted by the NASA Tensegrity Robotics Toolkit were within expected errors for robotic design, and that the accuracy extended from a single motion caused by a single actuator to the continuous coordinated motion of twelve actuators. Due to the nature of our cable implementation (applying impulses directly) changing Bullet constraint parameters such as the error reduction parameter (by a factor of 4) changes our results by less than 0.3%. We expect improving the friction models, both in actuators and within collisions will be the most effective way to improving the accuracy of this work. The accuracy of the work presented here could also be improved with more formal, automated system identification methods, especially if motion capture is used along with the force results. The errors as discussed would allow for designs with a safety factor of 1.15, whereas the actual safety factor we used in Tetraspine3's design was close to 2. These results confirm that we can continue to use NTRT for the design of new robots, with reasonable safety factors on parameters.

[2]Video available at: .`https://www.youtube.com/watch?v=VRdKwPsjmcI`

Chapter 8

Conclusions

8.1 Contributions

The contributions of this dissertation to the field of tensegrity robotics are algorithms that control adaptive tensegrity spines with many degrees of freedom, and simulation tools such as cable models with realistic forces and contact dynamics that are implemented in the open source NASA Tensegrity Robotics Toolkit. Use of CPGs coupled with impedance controllers was proven in two hardware implementations, one of which validated the forces predicted by the NASA Tensegrity Robotics Toolkit Simulator, and demonstrated locomotion of a machine learned controller. The final algorithms produced locomotion on a 72 degree of freedom spine with 88 actuators, and directed a six segment spine to multiple goal locations over three types of terrain. In doing so, this work also contributed to the field of biologically inspired robotic control, demonstrating the first CPG based robotic controller capable of goal directed locomotion on rough terrain.

8.2 Future Work: Methods and Details

8.2.1 Machine Learning

Though machine learning has enabled the tuning of controllers far faster and more efficiently than hand tuning, there is still room for improvement here. A logical extension of our current approach would be to conduct 'meta-learning,' where the algorithm automatically chooses when and how to make the problem more difficult. These algorithms exist in artificial intelligence applications [Vilalta and Drissi, 2002], and may benefit robotics.

Another approach would be to find online methods for tuning CPGs. This has already been attempted for a small, modular system using Powell's method [Sproewitz et al., 2008]. In order to scale up to the types of systems presented here, a more stochastic approach may be necessary. Work on using chaos to explore locomotion behaviors looks promising for online tuning [Shim and Husbands, 2012]. If properly designed, these methods could also tune parameters on hardware.

Finally, if tuning the controller is sped up or automated sufficiently, applying machine learning techniques could optimize the structures. Since most existing tensegrities have been designed as static structures, this could lead to the first tensegrity structures specifically designed for their dynamic motion. Additional components to optimize would be the placement of cables, and adding cables that span multiple segments.

8.2.2 Cable Models

As of this writing, the cables in NTRT do not interact with each other. Adding a collision resolution algorithm between cables would allow for more accurate simulation of nested tensegrity structures, meshes, and soft terrain features. This may require adding three dimensional internal dynamics, and mass, to the ca-

ble. One possible model, Corde, has already been partially implemented in NTRT [Spillmann and Teschner, 2007], but has not yet been used outside of simple test cases. The collision dynamics as currently implemented for this model (based on Spillman's dissertation) need improvement. The collision resolution algorithm that is being used for the massless cables (Section 3.2) would be a suitable replacement. Implementing the Corde model would also improve motor models, since this would better approximate friction between the cable and the spool.

8.2.3 Sensors

Tension and length sensors are certainly not the only sensors available to robots. Future work could consider how to incorporate additional sensors, such as ground contact sensors into the CPG for better performance on rough terrain. The goal directed motion on rough terrain could be improved by adding exteroceptive sensors such as vision or a laser range finder for planning or other high level control. Humans and animals are good at determining the path of least resistance, and adjusting the path given the terrain. Ideally, this sort of planning will be part of an autonomous controller. Incorporating sensor noise would also be a good way to improve the robustness of the control algorithms.

8.3 Future Work: High Level Research Directions

8.3.1 Tensegrity Quadruped

As may be obvious from the Chapters 1 and 2, the next high level work for tensegrity spines may be integrating them with a legged robot. There are two possible directions for this: either placing the spine as a component with traditional legs, or constructing an entire tensegrity quadruped. The latter is currently underway at UC Santa Cruz, pictured in Figure 8.1. A quadrupedal design should enable a tensegrity system to

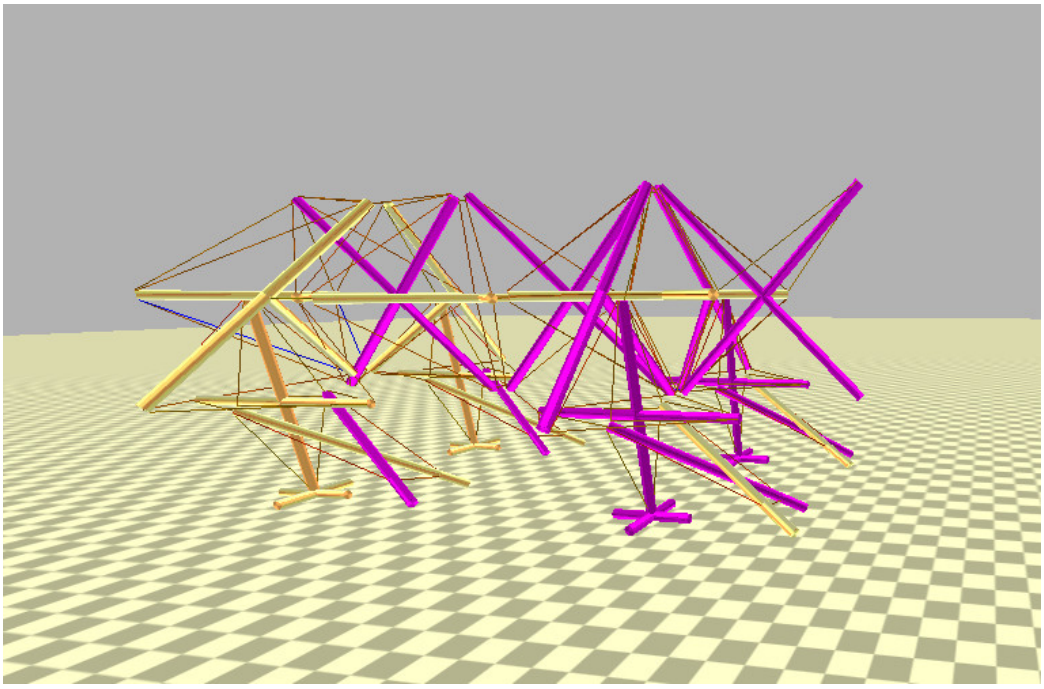

Figure 8.1: A prototype tensegrity quadruped 'big puppy' implemented in NTRT, inspired by a passive design by Tom Flemons. Printed with permission of Dawn Hustig-Schultz.

handle larger perturbations in the terrain more easily than the spine models.

In addition to the features of the NTRT simulator, the automatically generated CPG morphologies will be particularly valuable to designing controllers for the quadruped. Additionally, the concept of tension offsets and shape control as a steering mechanism from the goal direction work could inform whole body behaviors for this type of robot.

Cable routing and good design of tensegrity 'joints' will be important features in this work. In the spine work, the velocity of the robot never exceeded 150% of the actuator's maximum velocity. This is because the structures in this work never featured cables that spanned multiple segments. By creating properly constrained joints with actuators working in parallel, a quadruped robot should be able to exceed this speed limitation and maintain whole body compliance.

8.3.2 Tensegrity Biomechanics

It is worth asking whether a bio-inspired robotics effort can contribute to knowledge to biology, or in this case, biomechanics. While qualitative theories of tensegrity biomechanics are already being discussed, particularly by bodyworkers such as massage therapists and chiropractors (referred to by Myers as spatial medicine [Myers, 2014]), more work is necessary to quantify the tensegrity model relative to biomechanical data. The insight of tension holding bones apart, as opposed to traditional ball and socket joints, seems sufficiently valuable to merit further investigation, especially given recent advances in computing power. It is not clear whether tensegrities as simulated in this dissertation would have distinct advantages over tensegrity inspired finite element models in this respect, as the finite element models would be able to include more detailed material properties in their meshes.

8.4 Conclusions

This work will hopefully inform and encourage future work in compliant, adaptive robotic systems with many degrees of freedom, particularly robots with flexible spines and torsos. Future robots with flexible spines and torsos will be better able to perform in situations such as search and rescue, planetary exploration, and environments dangerous to humans.

Appendix A

Parameters Used In Simulation

The goal of this appendix is to capture a full set of parameters in one place so they do not have to be pieced together from disparate sections of code. This appendix will focus on hard coded or configuration based parameters, learned parameters can be found on Github (see Appendix B).

A.1 Learning Parameters

The following are from the final learning step that produced the goal directed controller. These are associated with the 'node' values of the CPG (radius, frequency, feedback gains).

```
''learning" : true,
"startingControllers" : 25,
"monteCarlo" : false,
"numberOfStates" : 0,
"numberOfOutputs" : 5,
"numberOfInstances" : 1,
"populationSize" : 50,
```

```
"useAverage" : true,
"numberToMutate" : 15,
"numberOfChildren" : 10,
"mutationChance" : 0.5,
"mutationDev" : 0.01,
"paramMax" : 1.0,
"paramMin": 0.0,
"childMutationChance" : 0.5
```

The previous trials had 25 elite controllers, so it made sense to start with the controllers that had already been tested. This could be increased to 50 if the trial needed to be restarted in exactly the same place (depending on the random seed). 50 trials were run per generation, meaning that controllers were run together in groups (no co-evolution). Number of states would indicate the number of inputs to a neural network. numberOfChildren indicates the output of crossover. mutationDev was often run at 0.03, this was finer tuning. Neural network weights were ± 1, note that scaling for individual parameters typically occurs in the C++ main application file, where controllers are declared.

A.2 Structure and actuator parameters

Note that this simulation was run at cm scale. Units of force are centi-Newtons $(cm - kg/s^2)$

Rods:

```
const double density = 4.2/300.0 kg/cm^3
const double radius  = 0.5 cm
const double friction = 0.5;
```

```
const double rollFriction = 0.0;
const double restitution = 0.0;

const double elasticity = 1000.0 kg/s^2
const double damping = 10.0 kg/s
const double pretension = 0.0 centi-Newtons
const bool   history = false;
const double maxTens = 7000.0 centi-Newtons
const double maxSpeed = 12.0 cm/s

const double mRad = 1.0 cm
const double motorFriction = 10.0 cm^2 kg / s
const double motorInertia = 1.0 cm^2 kg
const bool backDrivable = false;
```

Appendix B

NTRT Implementation Details

This appendix highlights how the controllers and algorithms discussed in this dissertation were implemented in the NASA Tensegrity Robotics Toolkit. The main project page for NTRT is: `http://ti.arc.nasa.gov/tech/asr/intelligent-robotics/tensegrity/ntrt/`. Current documentation detailing how to use the code can be found at `http://ntrt.perryb.ca/doxygen`. Each section of this appendix provides the relevant classes to look up within NTRT, with links that will help readers track the current code. If you wish to see the exact code that was used for the data presented here, look at commit c274e296f86ee640e6506155850df2b7bc7dccf9. This requires replacing the text in the link with the string of that commit number. For example:

`https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/core/tgBulletSpringCable.cpp` points to the most recent version of tgBulletSpringCable in the master (main) branch, while `https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/c274e296f86ee640e6506155850df2b7bc7dccf9/src/core/tgBulletSpringCable.cpp` points to the specific version written about here. Note that as of this writing that particular file has not changed since December 12, 2014, so those two links

should be identical.

B.1 Reference dates

Below lists the dates when data for this dissertation was taken. If use of features is not explicitly stated, this will provide a reference relative to implementation dates.

Jan. 2013 Data for [Tietz et al., 2013]

May 2013 Videos of Carbon Fiber Tetraspine

April 2014 Multiple spines for [Mirletz et al., 2014]

June 2014 NTRT's open source release on Github

September 2014 Data taken from Tetraspine 3 for simulator validation

December 2014 Contact cables and brushless motor model finalized

January 2015 NTRT version 1.1 released, data for CPG feedback [Mirletz et al., 2015b]

February 2015 Data for simulator validation updated to V 1.1 [Mirletz et al., 2015a]

April 2015 Controllers transitioned to JSON interface

June 2015 Goal directed motion

B.2 Chapter 3: NTRT Core and Cable Models

This section primarily deals with code located in src/core. Classes that are expected to be used by multiple applications are prefixed with 'tg'.

B.2.1 Bullet Simulation Core

The Bullet Physics Engine is primarily designed for collision detection and then solving the resulting linear complimentary problem for contact dynamics [Coumans, 2014]. Bullet includes several solvers for doing this, we found the best results with the Projected Gauss Seidel solver. Our interface for this (including collision detection algorithms) are included in tgWorldBulletPhysicsImpl: `https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/core/tgWorldBulletPhysicsImpl.cpp` This class also includes some enabling components for the cables with contact dynamics. This was last updated December, 12, 2014.

B.2.2 Cable Internal Dynamics

Due to the assumption of a cable being in line with a spring for each actuator, the base class for cables and actuators is a tgSpringCable. This defines the interface, the details of equation 3.1 are Bullet specific, and are therefore implemented in tgBulletSpringCable `https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/core/tgBulletSpringCable.cpp` Note the use of the applyImpulse function. I found that the friction behavior was more accurate with this function than with applyForce, since applyImpulse allows the actuator forces to be accurately considered when friction is applied. The cables interface with Bullet's rigid bodies via tgBulletSpringCableAnchor `https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/core/tgBulletSpringCableAnchor.h` The type of contact specified by the anchor (fixed or sliding for the contact cables) is specified in its configuration.

B.2.3 Cable Contact Dynamics

Contact dynamics are handled by tgBulletContactSpringCable, which is a child class of tgBulletSpringCable `https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/core/tgBulletContactSpringCable.cpp` The high level interface is identical to other spring cables, but the low level maintains a Bullet collision object: mghostObject. This class maintains vectors of tgBulletSpringCableAnchors, which are then related to Bullet's btPersistentManifold for collision detection. This pointer is somewhat problematic, as Bullet can update it without notifying NTRT. Thus tgBulletContactSpringCable::updateManifolds tries to rectify this. Problems with the anchors and the contact dynamics are being discussed here: `https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/issues/119`

B.2.4 Motor models

NTRT contains two models for actuating the cables. The base of both of these classes is tgSpringCableActuator. The basic behavior for changing the rest length of the system exists in tgBasicActuator: `https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/core/tgBasicActuator.h`. As long as you dont exceed the force or speed constraints set by the config (from tgSpringCable) then it will make your desired changes to the cables rest length.

tgKinematicActuator is the implementation of equation 3.3. This adds four additional parameters to the configuration, the motor's radius, the rotor's inertia, a friction value, and a Boolean indicating whether the motor is backdrivable. This model also implements a linear torque speed curve, and accounts for the external force when actuation is desired. PID control adjusts the desired applied tension. Numerical integration for the motor's acceleration and speed is handled internally with semi-implicit

Euler integration. https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/core/tgKinematicActuator.h

B.3 Chapter 4: Machine Learning

Many of the details of this chapter are available in src/learning. An updated Python implementation is available in scripts (this will change shortly to scripts/src/learning). https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/tree/master/scripts. The Python scripts include code to translate parameters from the C++ specification (.nnw comma delimited text files) to JSON.

Rather than implementing each method in a separate file, we implemented the methods in the same class and switched between them using configuration files. A summary of the options is available here: https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/doc/source/learning-library-walkthrough.rst

B.4 Chapter 5: Structures

Each of the spines discussed in Chapter 5 is implemented in src/examples/learningSpines. The design is specified as nodes in cartesian coordinates, pairs of nodes, and tags according to what should be created for each pair. This code utilizes the builder library in src/tgcreator in order to translate this structure specification into objects in the Bullet physics world. The procedure is as follows:

1. Declare a tgStructure - the container for a structural specification
2. Add nodes to this structure, as a point in three dimensions
3. Pair these points as rigid bodies and cables by node numbers, and tag them accordingly using words

4. Duplicate these structures, move them around, and place them in a super structure as you desire (for spines this is done inside of a for loop, since they are a linear chain)

5. Repeat steps 2 - 4 as desired

6. Declare the physical properties of your structure in a tgBuildSpec. This relates tags to 'info' classes which contain construction details, such as tgBasicActuatorInfo

7. Create a structure info (tgStructureInfo) and add the build spec and the tgStructure to that

8. Specify the model and the world that is being built into (typically a pointer to the class that is currently being setup)

9. Notify any controllers that setup is finished

The spines are implemented in the following classes:

Tetraspine `https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/examples/learningSpines/TetraSpine/TetraSpineLearningModel.cpp`

Tetrahedral Complex `https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/examples/learningSpines/TetrahedralComplex/FlemonsSpineModelLearning.cpp`

Octaheadral Complex `https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/examples/learningSpines/OctahedralComplex/FlemonsSpineModelLearningCL.cpp`

Ribbed Spine https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/examples/learningSpines/ribDemo/RibModel.cpp

B.5 Chapter 6: Control

B.5.1 Impedance Control

Common control classes are located under src/controllers. tgImpedanceController can be used with either motor model: https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/controllers/tgImpedanceController.cpp

B.5.2 CPG Implementation

The initial CPG implementation is located in src/util. The numerical integration is handled with the CPGEquations: https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/util/CPGEquations.h The value of any node can be pulled from this using the [ ] operator, as used by tgBaseCPGNode. Equations for each node are determined by the CPGNode class https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/util/CPGNode.cpp. Implementations for these equations can be found in BaseSpineCPGControl https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/examples/learningSpines/BaseSpineCPGControl.h and JSONCPGControl https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/dev/btietz/JSONTests/JSONCPGControl.h

B.5.3 CPG Morphology Determination

The high level behavior here is dictated by the controller, the most basic of which is BaseSpineCPGControl's setupCPGs method. This requires the structure, and the parameters from learning or a configuration file. The method assigns nodes to each of the structure's cables, which associates that cable's controller with a pointer to the CPG class (both of which are owned by BaseSpineCPGControl). Then setupCPGs calls each cable's setConnectivity method, and passes it a vector of all the cables in the structure. This method searches through all of the cables, and checks if it shares a rigid body. If it does, it gets the correct parameter from a four dimensional vector, based on whether the segment is in front of, behind, or the same as this cable, the cable's number within the segment (the remainder after dividing by the number of cables per segment), the other cable's number, and the parameter list. `https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/examples/learningSpines/tgCPGActuatorControl.cpp`

The data from [Mirletz et al., 2014] uses this control code and the applications associated with the spines as indicated in the previous section. Updates since that data was taken include adding the kinematic cables, the contact cables, and changes that occured during the validation work. An integration test ensures that performance never degrades without a good reason (and, has in fact upgraded since that paper was published): `https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/test_integration/SpineTests/WorldConf_Spines_test.cpp`

B.5.4 Feedback Code

The feedback code requires a separate set of CPG equations. These are implemented in CPGEquationsFB, CPGNodeFB, which are located in: `https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/tree/master/src/dev/CPG_feedback` This folder also includes structure

level and cable level controllers for this class. It is used in the multiTerrain application, which generated the data for [Mirletz et al., 2015b]: https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/tree/master/src/dev/btietz/multiTerrain That application can adjust terrain, robot, and controller features from the command line. This required an implementation of the tetrahedral complex spine with kinematic contact cables: https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/dev/btietz/TCContact/FlemonsSpineModelContact.cpp This behavior has been reproduced with a JSON based controller: https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/dev/btietz/JSONTests/JSONFeedbackControl.cpp

B.5.5 Goal Direction

In addition to the controller, goal directed spines needed to add the goal box to the simulation. This is done through BaseSpineModelGoal: https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/src/dev/btietz/TC_goal/BaseSpineModelGoal.h The final goal directed controller is called JSONGoalTensionNNW: https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/TC_Tension_length/src/dev/btietz/TC_nn_Tension/JSONGoalTensionNNW.cpp

B.6 Chapter 7: Hardware

Each plot in Chapter 7 used a different controller, so for simplicity they were each implemented in a different application. They all use common hardware, implemented in: https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/TC_Tension_length/src/dev/btietz/TetraSpineStatic/TetraSpineStaticModel_

hf.cpp Note that in addition to the standard build process, this places several spheres at locations around the structure to more accurately model heavy components. The static test (Figure 7.8) uses the following controller: https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/TC_Tension_length/src/dev/btietz/TetraSpineStatic/SerializedSpineControl.cpp Note that that controller directly applies forces to the rigid body via abstract markers. This is not recommended behavior, and we are working on alternative solutions. An integration test has been implemented to preserve this behavior: https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/master/test_integration/ICRA2015Tests/ICRA2015_test.cpp

Figure 7.9 utilizes: https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/TC_Tension_length/src/dev/btietz/handTunedSineWaves/htSpineSine.cpp and Figure 7.10 uses: https://github.com/NASA-Tensegrity-Robotics-Toolkit/NTRTsim/blob/TC_Tension_length/src/dev/btietz/hardwareLearning/LearningSpineJSON.cpp Raw robot data will be posted online and linked to NTRT after publication of the paper.

B.6.1 Arduino Code for Carbon Fiber Spine

Below is the full Arduino code for one segment of the carbon fiber Tetraspine:

```
#include <Encoder.h>

#include <PID_v1.h>

#include "DualMC33926MotorShield.h"

int k =0;
const float pi = 3.141592654;
long time;
long prevTime;
long tempTime;

//IT, IR, IL, OT, OR, OL
const int analogInPins[6] = {
```

```
  A2, A3, A4, A5, A6, A7};
const int motorSensorPins[6] = {
  A8, A9, A10, A11, A12, A13};
double sensorValue[6];
double prevSensorValue[6];
double sensorMin[6] = {
  220, 180, 160, 100, 280, 280};
double sensorMax[6] = {
  290, 215, 250, 130, 350, 350};

double motorOutputs[6];
double setPoints[6] = {
  730, 730, 730, 780, 730, 670};
double offSets[6] = {
  430, 335, 370, 250, 470, 470};
//double setPoints[6] = {200, 340, 466, 356, 158, 251};
double finalOutputs[6];

long enc_value[6];
long prev_enc[6];

DualMC33926MotorShield md_OL_OR(22, 4, A13, 24, 5, A12, 12, 53);
DualMC33926MotorShield md_OT_IR(26, 6, A11, 30, 8, A9, 12, 49);
DualMC33926MotorShield md_IL_IT(28, 7, A10, 32, 9, A8, 12, 43);
int otherInterrupts[3] = {
  45, 47, 41};
//M1DIR, M1PWM, M1FB, M2DIR, M2PWM, M2FB, nD2, nSF)

PID stringControl_IT(&sensorValue[0], &motorOutputs[0],
 &setPoints[0], 1, 0, 0, DIRECT);
PID stringControl_IR(&sensorValue[1], &motorOutputs[1],
 &setPoints[1], 1, 0, 0, DIRECT);
PID stringControl_IL(&sensorValue[2], &motorOutputs[2],
 &setPoints[2], 1, 0, 0, DIRECT);
PID stringControl_OT(&sensorValue[3], &motorOutputs[3],
 &setPoints[3], 1, 0, 0, DIRECT);
PID stringControl_OR(&sensorValue[4], &motorOutputs[4],
 &setPoints[4], 1, 0, 0, DIRECT);
PID stringControl_OL(&sensorValue[5], &motorOutputs[5],
 &setPoints[5], 1, 0, 0, DIRECT);

Encoder enc_OL(3,23);
Encoder enc_OR(18,25);
Encoder enc_OT(2,27);
Encoder enc_IL(19,29);
```

```
Encoder enc_IR(20,31);
Encoder enc_IT(21,33);

boolean faultValue;

int encDiff;
float sensorDiff;

int led = 13;

static inline int sgn(float val) {
  if (val < 0) return -1;
  if (val==0) return 0;
  return 1;
}

void stopIfFault()
{
  faultValue = false;
  for(int j = 0; j< 3; j++){
    if(!digitalRead(otherInterrupts[j])){
      Serial.println(j);
      faultValue=true;
    }
  }
  for(int j=0; j<6; j++){
    if(time>10&&(sensorValue[j]<25||sensorValue[j]>1020)){
      Serial.println(j);
      Serial.println(time);
      Serial.println(sensorValue[j]);
      faultValue=true;
    }
    encDiff = enc_value[j]-prev_enc[j];
    sensorDiff = sensorValue[j]-prevSensorValue[j];
    if((sgn(sensorDiff)!=-sgn(encDiff))
&&(sensorDiff*encDiff!=0)&&(abs(sensorDiff)>=15)
&&abs(encDiff)>20&&(abs(finalOutputs[j])>150))
    {
     Serial.println(j);
     Serial.print("Vel: ");
     Serial.print(sensorDiff);
     Serial.print(" Command ");
     Serial.println(finalOutputs[j]);
     //faultValue=true;
    }
```

```
  }

  if (md_OL_OR.getFault()||md_OT_IR.getFault()
||md_IL_IT.getFault()||faultValue)
  {
    if(md_OL_OR.getFault()){
      Serial.println("Pin 53");
    }
    if(md_OT_IR.getFault()){
      Serial.println("Pin 49");
    }
    if(md_IL_IT.getFault()){
      Serial.println("Pin 43");
    }
    Serial.println("fault");
    md_OL_OR.setSpeeds(0, 0);
    md_OT_IR.setSpeeds(0, 0);
    md_IL_IT.setSpeeds(0, 0);

    while(1);
  }
}

void setup()
{
  analogReference(EXTERNAL);
  Serial.begin(115200);
  Serial2.begin(9600);
  Serial3.begin(9600);
  Serial.println("Dual MC33926 Motor Shield");

  for(int i =0; i<3; i++){
    pinMode(otherInterrupts[i], INPUT);
  }

  md_OL_OR.init();
  md_OT_IR.init();
  md_IL_IT.init();
  md_OL_OR.setSpeeds(0, 0);
  md_OT_IR.setSpeeds(0, 0);
  md_IL_IT.setSpeeds(0, 0);
  stringControl_OL.SetMode(AUTOMATIC);
  stringControl_OR.SetMode(AUTOMATIC);
  stringControl_OT.SetMode(AUTOMATIC);
  stringControl_IL.SetMode(AUTOMATIC);
```

```
  stringControl_IR.SetMode(AUTOMATIC);
  stringControl_IT.SetMode(AUTOMATIC);

  for(int i=0; i<6; i++){
    prev_enc[i]=0;
    prevSensorValue[i]=analogRead(analogInPins[i]);
  }
  pinMode(led, OUTPUT);
  time= millis();
  prevTime = millis();
}

void loop()
{
  time= millis();
  //stopIfFault();
  enc_value[5] = enc_OL.read();
  enc_value[4] = enc_OR.read();
  enc_value[3] = enc_OT.read();
  enc_value[2] = enc_IL.read();
  enc_value[1] = enc_IR.read();
  enc_value[0] = enc_IT.read();
  //Serial.println(time-prevTime);

  for(int i=0; i<6; i++){
    //stopIfFault();

    sensorValue[i] = analogRead(analogInPins[i]);
    //sensorValue[i] = stretchValue[i];
    if(i>3){ //Set 2 outside bottom
      setPoints[i] = offSets[i] + enc_value[i]/75.0
+(enc_value[i]-prev_enc[i])/(time-prevTime)/1000.0
+100.0*sin(pi/1250*time+pi/2);
    }
    else if(i>2){ //Set 2 outside top
      setPoints[i] = offSets[i] + enc_value[i]/75.0
+(enc_value[i]-prev_enc[i])/(time-prevTime)/1000.0
+100.0*sin(pi/1250*time+pi/2);
    }
    else if (i>0){ //set 1 inside bottom
      setPoints[i] = offSets[i] + enc_value[i]/10.0
+(enc_value[i]-prev_enc[i])/(time-prevTime)/1000.0
-100.0*sin(pi/1250*time);
    }
    else{ //Set 1 inside top
```

```
      setPoints[i] = offSets[i] + enc_value[i]/10.0
+(enc_value[i]-prev_enc[i])/(time-prevTime)/1000.0
-100.0*sin(pi/1250*time);
    }
  }
  stopIfFault();

  //Serial.print("Encoder: ");
  //Serial.println(newLong/250.0);

  //Serial.print("PID In: ");
  //Serial.println(sensorValue);

  stringControl_IT.Compute();
  stringControl_IR.Compute();
  stringControl_IL.Compute();
  stringControl_OT.Compute();
  stringControl_OR.Compute();
  stringControl_OL.Compute();

  for(int i=0; i<6; i++){
    //stopIfFault();
    //motorOutputs[i] = (setPoints[i]-sensorValue[i]);
    //finalOutputs[i] = 3*motorOutputs[i];
    if(i>2){
      finalOutputs[i] = 3*(motorOutputs[i]-128);
    }
    else{
      finalOutputs[i] = 3*(motorOutputs[i]-128);
    }

    if(((sensorValue[i]<sensorMin[i])&&(finalOutputs[i]<0))
||((sensorValue[i]>sensorMax[i])&&(finalOutputs[i]>0))){
      finalOutputs[i]=0;
    }

    if(k>500&&Serial){
      Serial.print("Time: ");
      Serial.print(time);
      Serial.print(" Out: ");
      Serial.print(i);
      Serial.print(" enc: ");
      Serial.print(enc_value[i]/125.0);
      Serial.print(" sensor: ");
      Serial.print(sensorValue[i]);
```

```
      Serial.print(" PID ");
      Serial.print(finalOutputs[i]);
      Serial.print(" desired: ");
      Serial.print(setPoints[i]);
      Serial.print(" Curr ");
      Serial.println(analogRead(motorSensorPins[i]));

    }
    }

    if(k>500){
        //Serial.println(analogRead(A0));
      k=0;
    }
    //Get Base Voltage


    md_OL_OR.setSpeeds(0*finalOutputs[5], 0*finalOutputs[4]);
    md_OT_IR.setSpeeds(0*finalOutputs[3], 0*finalOutputs[1]);
    md_IL_IT.setSpeeds(0*finalOutputs[2], 0*finalOutputs[0]);
    k++;

  if(millis()%1000>900){
    digitalWrite(led, HIGH);
  }
  else{
    digitalWrite(led, LOW);
  }

  for(int i=0; i<6; i++){
    prev_enc[i]=enc_value[i];
    prevSensorValue[i] = sensorValue[i];
  }
  prevTime = time;
}
```

Bibliography

[Agogino et al., 2013] Agogino, A., SunSpiral, V., and Atkinson, D. (2013). Super ball bot: structures for planetary landing and exploration. *NASA Innovative Advanced Concepts (NIAC) Program, Final Report*.

[Ahnert and Mulansky, 2011] Ahnert, K. and Mulansky, M. (2011). Odeint - Solving ordinary differential equations in C++. In *AIP Conf. Proc. 1389*, pages 1586–1589.

[Aldrich et al., 2003] Aldrich, J., Skelton, R., and Kreutz-Delgado, K. (2003). Control synthesis for a class of light and agile robotic tensegrity structures. In *American Control Conference, 2003. Proceedings of the 2003*, volume 6, pages 5245–5251. IEEE.

[Alexander, 1988] Alexander, R. M. (1988). Why mammals gallop. *American zoologist*, 28(1):237–245.

[Allen et al., 2003] Allen, T. J., Quinn, R. D., Bachmann, R. J., and Ritzmann, R. E. (2003). Abstracted biological principles applied with reduced actuation improve mobility of legged vehicles. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1370–1375. IEEE.

[Bhounsule et al., 2014] Bhounsule, P. A., Cortell, J., Grewal, A., Hendriksen, B., Karssen, J. D., Paul, C., and Ruina, A. (2014). Low-bandwidth reflex-based control for lower power walking: 65 km on a single battery charge. *The International Journal of Robotics Research*, 33(10):1305–1321.

[Bicanski et al., 2013] Bicanski, A., Ryczko, D., Knuesel, J., Harischandra, N., Charrier, V., Ekeberg, Ö., Cabelguen, J., and Ijspeert, A. J. (2013). Decoding the mechanisms of gait generation in salamanders by combining neurobiology, modeling and robotics. *Biological cybernetics*, 107(5):545–564.

[Bliss, 2011] Bliss, T. (2011). Central pattern generator control of a tensegrity swimmer. *Ph.D. thesis, Dept. Mech. Aerosp. Eng., Univ. Virginia,*.

[Bliss et al., 2013] Bliss, T., Werly, J., Iwasaki, T., and Bart-Smith, H. (2013). Experimental validation of robust resonance entrainment for cpg-controlled tensegrity structures. *Control Systems Technology, IEEE Transactions on*, 21(3):666–678.

[Bliss et al., 2008] Bliss, T. K., Iwasaki, T., and Bart-Smith, H. (2008). CPG Control of a Tensegrity Morphing Structure for Biomimetic Applications. *Advances in Science and Technology*, 58:137–142.

[Boeing and Bräunl, 2007] Boeing, A. and Bräunl, T. (2007). Evaluation of real-time physics simulation systems. In *Proceedings of the 5th int. conf. on Computer graphics and interactive techniques in Australia and Southeast Asia*, pages 281–288. ACM.

[Brown, 1911] Brown, T. G. (1911). The intrinsic factors in the act of progression in the mammal. *Proceedings of the Royal Society of London. Series B, Containing Papers of a Biological Character*, 84(572):308–319.

[Bruce et al., 2014] Bruce, J., Caluwaerts, K., Iscen, A., Sabelhaus, A. P., and SunSpiral, V. (2014). Design and evolution of a modular tensegrity robot platform. In *IEEE Int. Conf. on Robotics and Automation (ICRA-2014)*.

[Buchli et al., 2006] Buchli, J., Righetti, L., and Ijspeert, A. J. (2006). Engineering entrainment and adaptation in limit cycle systems. *Biological Cybernetics*, 95(6):645–664.

[Burt, 2013] Burt, S. J. (2013). Kinematics algorithms for tensegrity structures.

[Büschges et al., 2008] Büschges, A., Akay, T., Gabriel, J. P., and Schmidt, J. (2008). Organizing network action for locomotion: insights from studying insect walking. *Brain research reviews*, 57(1):162–171.

[Cabelguen et al., 2003] Cabelguen, J.-M., Bourcier-Lucas, C., and Dubuc, R. (2003). Bimodal locomotion elicited by electrical stimulation of the midbrain in the salamander notophthalmus viridescens. *The Journal of Neuroscience*, 23(6):2434–2439.

[Caluwaerts et al., 2014] Caluwaerts, K., Despraz, J., Işçen, A., Sabelhaus, A. P., Bruce, J., Schrauwen, B., and SunSpiral, V. (2014). Design and control of compliant tensegrity robots through simulation and hardware validation. *Journal of The Royal Society Interface*, 11(98):20140520.

[Caluwaerts et al., 2013] Caluwaerts, K., D'Haene, M., Verstraeten, D., and Schrauwen, B. (2013). Locomotion without a brain: Physical reservoir computing in tensegrity structures. *Artificial life*, 19(1):35–66.

[Cholewicki and McGill, 1996] Cholewicki, J. and McGill, S. M. (1996). Mechanical stability of the in vivo lumbar spine: implications for injury and chronic low back pain. *Clinical Biomechanics*, 11(1):1–15.

[Collins et al., 2005] Collins, S., Ruina, A., Tedrake, R., and Wisse, M. (2005). Efficient bipedal robots based on passive-dynamic walkers. *Science*, 307(5712):1082–1085.

[Connelly and Back, 1998] Connelly, R. and Back, A. (1998). Mathematics and tensegrity: Group and representation theory make it possible to form a complete catalogue of" strut-cable" constructions with prescribed symmetries. *American Scientist*, pages 142–151.

[Coumans, 2012] Coumans, E. (2012). Bullet physics library. `http://bulletphysics.org/wordpress/`.

[Coumans, 2014] Coumans, E. (2014). Exploring mlcp solvers and featherstone. In *Game Developers Conference*.

[Crespi and Ijspeert, 2006] Crespi, A. and Ijspeert, A. J. (2006). Amphibot ii: An amphibious snake robot that crawls and swims using a central pattern generator. In *Proceedings of the 9th int. conf. on climbing and walking robots (CLAWAR 2006)*, volume 11, pages 19–27. Citeseer.

[Crespi et al., 2007] Crespi, A., Lachat, D., Pasquier, A., and Ijspeert, A. J. (2007). Controlling swimming and crawling in a fish robot using a central pattern generator. *Autonomous Robots*, 25(1-2):3–13.

[Daltorio et al., 2008] Daltorio, K., Witushynsky, T. C., Wile, G. D., Palmer, L. R., Malek, A. A., Ahmad, M. R., Southard, L., Gorb, S. N., Ritzmann, R. E., Quinn, R. D., et al. (2008). A body joint improves vertical to horizontal transitions of a wall-climbing robot. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3046–3051. IEEE.

[Daun-Gruhn and Tóth, 2011] Daun-Gruhn, S. and Tóth, T. I. (2011). An intersegmental network model and its use in elucidating gait-switches in the stick insect. *Journal of computational neuroscience*, 31(1):43–60.

[Delcomyn, 1980] Delcomyn, F. (1980). Neural basis of rhythmic behavior in animals. *Science*, 210(4469):492–498.

[Djouadi et al., 1998] Djouadi, S., Motro, R., Pons, J., and Crosnier, B. (1998). Active control of tensegrity systems. *Journal of Aerospace Engineering*, 11(2):37–44.

[Duperret et al., 2014] Duperret, J., Kenneally, G., Pusey, J., and Koditschek, D. (2014). Towards a comparative measure of legged agility. In *International Symposium on Experimental Robotics*.

[Eckert et al., 2015] Eckert, P., Spröwitz, A., Witte, H., and Ijspeert, A. J. (2015). Comparing the effect of different spine and leg designs for a small bounding quadruped robot. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3128–3133. IEEE.

[Ekeberg, 1993] Ekeberg, Ö. (1993). A combined neuronal and mechanical model of fish swimming. *Biological cybernetics*, 69(5-6):363–374.

[Erez et al., 2015] Erez, T., Tassa, Y., and Todorov, E. (2015). Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4397–4404. IEEE.

[Fest et al., 2004] Fest, E., Shea, K., and Smith, I. F. (2004). Active tensegrity structure. *Journal of Structural Engineering*, 130(10):1454–1465.

[Flemons, 2007] Flemons, T. (2007). The geometry of anatomy. `http://www.intensiondesigns.com/geometry_of_anatomy.html`.

[Flemons, 2012] Flemons, T. (2012). The bones of tensegrity. *http://www.intensiondesigns.com/bones_of_tensegrity*.

[Franklin et al., 2010] Franklin, K., Ozkan, E., and Powell, D. (2010). Design of the kurilpa pedestrian bridge for dynamic effects due to pedestrian and wind loads. In *5th Civil Engineering Conference in the Asian Region and Australasian Structural Engineering Conference 2010, The*, page 885. Engineers Australia.

[Friesen et al., 2014] Friesen, J., Pogue, A., Bewley, T., de Oliveira, M., Skelton, R., and SunSpiral, V. (2014). Ductt: a tensegrity robot for exploring duct systems. In *IEEE Int. Conf. on Robotics and Automation (ICRA-2014)*. IEEE.

[Fuller, 1962] Fuller, R. (1962). Tensile-integrity structures. US Patent 3,063,521.

[Fuller, 1965] Fuller, R. B. (1965). Laminar geodesic dome. US Patent 3,203,144.

[Gay et al., 2013] Gay, S., Santos-Victor, J., and Ijspeert, A. (2013). Learning robot gait stability using neural networks as sensory feedback function for central pattern generators. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 194–201. IEEE.

[Gracovetsky, 1985] Gracovetsky, S. (1985). An hypothesis for the role of the spine in human locomotion: a challenge to current thinking. *Journal of biomedical engineering*, 7(3):205–216.

[Gray, 1918] Gray, H. (1918). *Anatomy of the human body*. Lea & Febiger.

[Grillner, 1985a] Grillner, S. (1985a). Neural control of vertebrate locomotion-central mechanisms and reflex interaction with special reference to the cat. In *Feedback and motor control in invertebrates and vertebrates*, pages 35–56. Springer.

[Grillner, 1985b] Grillner, S. (1985b). Neurobiological bases of rhythmic motor acts in vertebrates. *Science*, 228(4696):143–149.

[Grillner, 2006] Grillner, S. (2006). Biological pattern generation: the cellular and computational logic of networks in motion. *Neuron*, 52(5):751–766.

[Grillner et al., 2007] Grillner, S., Kozlov, A., Dario, P., Stefanini, C., Menciassi, A., Lansner, A., and Hellgren Kotaleski, J. (2007). Modeling a vertebrate motor system: pattern generation, steering and control of body orientation. *Progress in brain research*, 165:221–234.

[Grillner et al., 1981] Grillner, S., McClellan, A., and Perret, C. (1981). Entrainment of the spinal pattern generators for swimming by mechano-sensitive elements in the lamprey spinal cord in vitro. *Brain research*, 217(2):380–386.

[Guizzo and Ackerman, 2015] Guizzo, E. and Ackerman, E. (2015). Darpa robotics challenge: A compilation of robots falling down [online]. `http://spectrum.ieee.org/automaton/robotics/humanoids/darpa-robotics-challenge-robots-falling`. Accessed: 2015-06-18.

[Hauser et al., 2011] Hauser, H., Ijspeert, A. J., Füchslin, R. M., Pfeifer, R., and Maass, W. (2011). Towards a theoretical foundation for morphological computation with compliant bodies. *Biological cybernetics*, 105(5-6):355–370.

[Hauser et al., 2012] Hauser, H., Ijspeert, A. J., Füchslin, R. M., Pfeifer, R., and Maass, W. (2012). The role of feedback in morphological computation with compliant bodies. *Biological cybernetics*, 106(10):595–613.

[Heartney, 2009] Heartney, E. (2009). *Kenneth Snelson: forces made visible*. Hard press editions Lenox.

[Hellgren et al., 1992] Hellgren, J., Grillner, S., and Lansner, A. (1992). Computer simulation of the segmental neural network generating locomotion in lamprey by using populations of network interneurons. *Biological cybernetics*, 68(1):1–13.

[Hirai and Imuta, 2012] Hirai, S. and Imuta, R. (2012). Dynamic simulation of six-strut tensegrity robot rolling. In *Robotics and Biomimetics (ROBIO), 2012 IEEE Int. Conf. on*, pages 198–204. IEEE.

[Hogan, 1985] Hogan, N. (1985). Impedance control: An approach to manipulation: Part I - Theory. *Transactions of the ASME*, 107(March 1985):1–7.

[Ijspeert, 2008] Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: a review. *Neural networks: the official journal of the Int. Neural Network Society*, 21(4):642–53.

[Ijspeert et al., 2007] Ijspeert, A. J., Crespi, A., Ryczko, D., and Cabelguen, J. M. (2007). From swimming to walking with a salamander robot driven by a spinal cord model. *Science (New York, N.Y.)*, 315(5817):1416–1420.

[Ingber, 2003] Ingber, D. E. (2003). Tensegrity ii. how structural networks influence cellular information processing networks. *Journal of cell science*, 116(8):1397–1408.

[Ingber et al., 2014] Ingber, D. E., Wang, N., and Stamenović, D. (2014). Tensegrity, cellular biophysics, and the mechanics of living systems. *Reports on Progress in Physics*, 77(4):046603.

[Iscen, 2014] Iscen, A. (2014). *Multiagent learning for locomotion and coordination in tensegrity robotics*. PhD thesis, Oregon State University.

[Iscen et al., 2013a] Iscen, A., Agogino, A., SunSpiral, V., and Tumer, K. (2013a). Controlling tensegrity robots through evolution. In *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference*, GECCO '13, pages 1293–1300, New York, NY, USA. ACM.

[Iscen et al., 2013b] Iscen, A., Agogino, A., SunSpiral, V., and Tumer, K. (2013b). Robust distributed control of rolling tensegrity robot. In *The Autonomous Robots and Multirobot Systems (ARMS) workshop at AAMAS 2013*.

[Iscen et al., 2014] Iscen, A., Agogino, A., SunSpiral, V., and Tumer, K. (2014). Flop and roll: Learning robust goal-directed locomotion for a tensegrity robot. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2236–2243. IEEE.

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

[Kern, 2012] Kern, N. I. (2012). *Cable-Driven Flexible Spines for Human Orthoses and Mobile Robots*. PhD thesis, Case Western Reserve University.

[Khazanov et al., 2013] Khazanov, M., Humphreys, B., Keat, W., and Rieffel, J. (2013). Exploiting dynamical complexity in a physical tensegrity robot to achieve locomotion. In *Advances in Artificial Life, ECAL*, volume 12, pages 965–972.

[Kim et al., 2014] Kim, K., Agogino, A. K., Moon, D., Taneja, L., Toghyan, A., Dehghani, B., SunSpiral, V., and Agogino, A. M. (2014). Rapid prototyping design and control of tensegrity soft robot for locomotion. In *International Conference on Robotics and Biometrics*.

[Koizumi et al., 2012] Koizumi, Y., Shibata, M., and Hirai, S. (2012). Rolling tensegrity driven by pneumatic soft actuators. In *2012 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1988–1993.

[Korkmaz et al., 2011] Korkmaz, S., Ali, N. B. H., and Smith, I. F. (2011). Determining control strategies for damage tolerance of an active tensegrity structure. *Engineering Structures*, 33(6):1930–1939.

[Leeser, 1996] Leeser, K. F. (1996). Locomotion experiments on a planar quadruped robot with articulated spine. Master's thesis, Massachusetts Institute of Technology.

[Levin, 1997] Levin, S. M. (1997). Putting the shoulder to the wheel: a new biomechanical model for the shoulder girdle. *Biomedical sciences instrumentation*, 33:412–417.

[Levin, 2002] Levin, S. M. (2002). The tensegrity-truss as a model for spine mechanics: biotensegrity. *Journal of Mechanics in Medicine and Biology*, 2:375–388.

[Levy, 1991] Levy, M. (1991). Floating fabric over georgia dome. *Civil EngineeringASCE*, 61(11):34–37.

[Lipson, 2014] Lipson, H. (2014). Challenges and opportunities for design, simulation, and fabrication of soft robots. *Soft Robotics*, 1(1):21–27.

[Liu et al., 2009] Liu, S.-Y., Cheng, T.-H., Yan, J.-L., and Huang, H.-P. (2009). Development of a dexterous cable driven spine mechanism for humanoid robots. In *Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference on*, pages 1588–1593. IEEE.

[Masic and Skelton, 2004] Masic, M. and Skelton, R. E. (2004). Open-loop control of class-2 tensegrity towers. *Proc. of the 11th Smart Structures and Materials Conference*, 5383:298–308.

[Masic et al., 2005] Masic, M., Skelton, R. E., and Gill, P. E. (2005). Algebraic tensegrity form-finding. *International Journal of Solids and Structures*, 42(16):4833–4858.

[Matsuoka, 1985] Matsuoka, K. (1985). Sustained oscillations generated by mutually inhibiting neurons with adaptation. *Biological cybernetics*, 52(6):367–376.

[McGarry and Prendergast, 2004] McGarry, J. G. and Prendergast, P. J. (2004). A three-dimensional finite element model of an adherent eukaryotic cell. *Eur Cell Mater*, 7:27–33.

[Mirats-Tur and Camps, 2011] Mirats-Tur, J. M. and Camps, J. (2011). A three-dof actuated robot. *Robotics & Automation Magazine, IEEE*, 18(3):96–103.

[Mirletz et al., 2014] Mirletz, B. T., Park, I. W., Flemons, T. E., Agogino, A. K., Quinn, R. D., and SunSpiral, V. (2014). Design and control of modular spine-like tensegrity structures. In *The 6th World Conf. of the Int. Association for Structural Control and Monitoring (6WCSCM)*.

[Mirletz et al., 2015a] Mirletz, B. T., Park, I. W., Quinn, R. D., and SunSpiral, V. (2015a). Towards bridging the reality gap between tensegrity simulation and robotic hardware. In *To appear at 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[Mirletz et al., 2015b] Mirletz, B. T., Quinn, R. D., and SunSpiral, V. (2015b). Cpgs for adaptive control of spine-like tensegrity structures. In *2015 International Conference on Robotics and Automation (ICRA2015) Workshop on Central Pattern Generators for Locomotion Control: Pros, Cons & Alternatives*.

[Mizuuchi et al., 2005] Mizuuchi, I., Yoshikai, T., Nakanishi, Y., Sodeyama, Y., Yamamoto, T., Miyadera, A., Niemelä, T., Hayashi, M., Urata, J., and Inaba, M. (2005). Development of muscle-driven flexible-spine humanoids. In *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*, pages 339–344. IEEE.

[Moored et al., 2011] Moored, III, K. W., Taylor, S. A., and Bart-Smith, H. (2011). Optimization of a Tensegrity Wing for Biomimetic Applications. *Proceedings of SPIE*, 6173:617313.

[Mori and Hirose, 2002] Mori, M. and Hirose, S. (2002). Three-dimensional serpentine motion and lateral rolling by active cord mechanism acm-r3. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 829–834 vol.1.

[Myers, 2014] Myers, T. (2014). Spatial medicine–a call to arms. *Journal of bodywork and movement therapies*, 18(1):94–98.

[Myers et al., 2001] Myers, T. W., Chaitow, L., and Juhan, D. (2001). *Anatomy Trains: Myofascial Meridians for Manual and Movement Therapists, 1e.* Churchill Livingstone, 1 edition.

[Nagase and Skelton, 2015] Nagase, K. and Skelton, R. (2015). Double-helix tensegrity structures. *AIAA Journal*, 53(4):847–862.

[Orki, 2012] Orki, O. (2012). *A Model Of Caterpillar Locomotion Based On Assur Tensegrity Structures*. PhD thesis, Tel Aviv University.

[Orki et al., 2012] Orki, O., Ayali, A., Shai, O., and Ben-Hanan, U. (2012). Modeling of caterpillar crawl using novel tensegrity structures. *Bioinspiration & Biomimetics*, 7(4):046006.

[Park et al., 2015] Park, H.-W., Park, S., and Kim, S. (2015). Variable-speed quadrupedal bounding using impulse planning: Untethered high-speed 3d running of mit cheetah 2. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 5163–5170. IEEE.

[Paul, 2006] Paul, C. (2006). Morphological computation: A basis for the analysis of morphology and control requirements. *Robotics and Autonomous Systems*, 54(8):619–630.

[Paul et al., 2005a] Paul, C., Lipson, H., and Cuevas, F. J. V. (2005a). Evolutionary form-finding of tensegrity structures. *Proceedings of the 2005 Genetic and Evolutionary Computation Conference (GECCO)*, pages 3–10.

[Paul et al., 2005b] Paul, C., Roberts, J. W., Lipson, H., and Cuevas, F. J. V. (2005b). Gait production in a tensegrity based robot. In *Advanced Robotics, 2005. ICAR '05. Proceedings., 12th Int. Conf. on*.

[Paul et al., 2006] Paul, C., Valero-Cuevas, F. J., and Lipson, H. (2006). Design and control of tensegrity robots for locomotion. *IEEE Transactions on Robotics*, 22(5).

[Pfeifer and Gómez, 2009] Pfeifer, R. and Gómez, G. (2009). Morphological computation–connecting brain, body, and environment. In *Creating Brain-Like Intelligence*, pages 66–83. Springer.

[Pfeifer et al., 2013] Pfeifer, R., Marques, H. G., and Iida, F. (2013). Soft robotics: the next generation of intelligent machines. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 5–11. AAAI Press.

[Raibert et al., 2008] Raibert, M., Blankespoor, K., Nelson, G., Playter, R., et al. (2008). Bigdog, the rough-terrain quadruped robot. In *17th World Congr. The Int. Federation of Automatic Control*, pages 10823–10825.

[Rieffel et al., 2008] Rieffel, J., Trimmer, B., and Lipson, H. (2008). Mechanism as Mind : What Tensegrities and Caterpillars Can Teach Us about Soft Robotics The Manduca Sexta Caterpillar : Morphological Communication in Tensegrity Robots. *Artificial Life*, pages 506–512.

[Rieffel et al., 2009] Rieffel, J., Valero-Cuevas, F., and Lipson, H. (2009). Automated discovery and optimization of large irregular tensegrity structures. *Computers & Structures*, 87(5):368–379.

[Rieffel et al., 2010] Rieffel, J. A., Valero-Cuevas, F. J., and Lipson, H. (2010). Morphological communication: exploiting coupled dynamics in a complex mechanical structure to achieve locomotion. *Journal of the Royal Society, Interface / the Royal Society*, 7(45):613–21.

[Righetti et al., 2006] Righetti, L., Buchli, J., and Ijspeert, A. J. (2006). Dynamic hebbian learning in adaptive frequency oscillators. *Physica D: Nonlinear Phenomena*, 216(2):269–281.

[Righetti et al., 2009] Righetti, L., Buchli, J., and Ijspeert, A. J. (2009). Adaptive frequency oscillators and applications. *Open Cybernetics & Systemics Journal*, 3:64–69.

[Robertson and Moulins, 1981] Robertson, R. M. and Moulins, M. (1981). Oscillatory command input to the motor pattern generators of the crustacean stomatogastric ganglion. *Journal of comparative physiology*, 143(4):453–463.

[Roos et al., 2006] Roos, L., Guenter, F., Guignard, A., and Billard, A. G. (2006). Design of a biomimetic spine for the humanoid robot robota. In *Biomedical Robotics and Biomechatronics, 2006. BioRob 2006. The First IEEE/RAS-EMBS International Conference on*, pages 329–334. IEEE.

[Rovira and Mirats Tur, 2009] Rovira, A. G. and Mirats Tur, J. M. (2009). Control and simulation of a tensegrity-based mobile robot. *Robotics and Autonomous Systems*, 57(5):526–535.

[Ruina, 2011] Ruina, A. (2011). Cornell ranger 2011, 4-legged bipedal robot. [online]. `http://ruina.tam.cornell.edu/research/topics/locomotion_and_robotics/ranger/Ranger2011/`. Accessed: 2015-02-02.

[Sabelhaus et al., 2015] Sabelhaus, A. P., Ji, H., Hylton, P., Madaan, Y., Yang, C., Agogino, A. M., Friesen, J., and SunSpiral, V. (2015). Mechanism design and simulation of the ultra spine, a tensegrity robot. In *To appear in the Proceedings of The ASME 2015 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/ CIE 2015).*

[Scarr, 2014] Scarr, G. (2014). *Biotensegrity: The structural basis of life*. Handspring Publishing, first edition.

[Schilling and Carrier, 2010] Schilling, N. and Carrier, D. R. (2010). Function of the epaxial muscles in walking, trotting and galloping dogs: implications for the evolution of epaxial muscle function in tetrapods. *The Journal of experimental biology*, 213(9):1490–1502.

[Schilling and Hackert, 2006] Schilling, N. and Hackert, R. (2006). Sagittal spine movements of small therian mammals during asymmetrical gaits. *Journal of Experimental Biology*, 209(19):3925–3939.

[Secor et al., 1992] Secor, S. M., Jayne, B. C., and Bennett, A. F. (1992). Locomotor performance and energetic cost of sidewinding by the snake crotalus cerastes. *Journal of experimental biology*, 163(1):1–14.

[Seok et al., 2013] Seok, S., Wang, A., Chuah, M. Y., Otten, D., Lang, J., and Kim, S. (2013). Design principles for highly efficient quadrupeds and implementation on the mit cheetah robot. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3307–3312. IEEE.

[Servin and Lacoursière, 2007] Servin, M. and Lacoursière, C. (2007). Massless cable for real-time simulation. In *Computer Graphics Forum*, volume 26, pages 172–184. Wiley Online Library.

[Servin et al., 2011] Servin, M., Lacoursiere, C., Nordfelth, F., and Bodin, K. (2011). Hybrid, multiresolution wires with massless frictional contacts. *Visualization and Computer Graphics, IEEE Transactions on*, 17(7):970–982.

[Shim and Husbands, 2012] Shim, Y. and Husbands, P. (2012). Chaotic exploration and learning of locomotion behaviors. *Neural computation*, 24(8):2185–2222.

[Skelton et al., 2001] Skelton, R. E., Adhikari, R., Pinaud, J.-P., Chan, W., and Helton, J. (2001). 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.

[Skelton and De Oliveira, 2009] Skelton, R. E. and De Oliveira, M. C. (2009). *Tensegrity Systems*. Springer, 2009 edition.

[Snelson, 1965] Snelson, K. (February 1965). Continuous tension, discontinuous compression structures. United States patent 3169611.

[Spillmann and Teschner, 2007] Spillmann, J. and Teschner, M. (2007). C o r d e: Cosserat rod elements for the dynamic simulation of one-dimensional elastic objects. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 63–72. Eurographics Association.

[Sproewitz et al., 2008] Sproewitz, A., Moeckel, R., Maye, J., and Ijspeert, A. J. (2008). Learning to move in modular robots using central pattern generators and online optimization. *The International Journal of Robotics Research*, 27(3-4):423–443.

[Sterk, 2006] Sterk, T. d. (2006). Shape control in responsive architectural structures-current reasons & challenges. In *Proceedings of the 4th World Conference on Structural Control and Monitoring, San Diego, CA, USA*.

[Stevenson and Kutsch, 1987] Stevenson, P. A. and Kutsch, W. (1987). A reconsideration of the central pattern generator concept for locust flight. *Journal of Comparative Physiology A*, 161(1):115–129.

[SunSpiral et al., 2013] SunSpiral, V., Gorospe, G., Bruce, J., Iscen, A., Korbel, G., Milam, S., Agogino, A., and Atkinson, D. (2013). Tensegrity based probes for planetary exploration: Entry, descent and landing (edl) and surface mobility analysis. In *10th International Planetary Probe Workshop*.

[Takuma et al., 2010] Takuma, T., Ikeda, M., and Masuda, T. (2010). Facilitating multi-modal locomotion in a quadruped robot utilizing passive oscillation of the spine structure. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 4940–4945. IEEE.

[Tietz et al., 2013] Tietz, B. R., Carnahan, R. W., Bachmann, R. J., Quinn, R. D., and Sunspiral, V. (2013). Tetraspine: Robust Terrain Handling on a Tensegrity Robot Using Central Pattern Generators. In *IEEE/ASME Advanced Intelligent Mechatronics*, pages 261–267.

[Transeth et al., 2009] Transeth, A. A., Pettersen, K. Y., and Liljebäck, P. (2009). A survey on snake robot modeling and locomotion. *Robotica*, 27(07):999.

[Tucker, 1970] Tucker, V. A. (1970). Energetic cost of locomotion in animals. *Comparative Biochemistry and Physiology*, 34(4):841–846.

[Turvey and Fonseca, 2014] Turvey, M. T. and Fonseca, S. T. (2014). The medium of haptic perception: A tensegrity hypothesis. *Journal of motor behavior*, 46(3):143–187.

[Vilalta and Drissi, 2002] Vilalta, R. and Drissi, Y. (2002). A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95.

[Wang et al., 2002] Wang, N., Tolić-Nørrelykke, I. M., Chen, J., Mijailovich, S. M., Butler, J. P., Fredberg, J. J., and Stamenović, D. (2002). Cell prestress. i. stiffness and prestress are closely associated in adherent contractile cells. *American Journal of Physiology-Cell Physiology*, 282(3):C606–C616.

[Webster et al., 2013] Webster, V., Lonsberry, A. J., Horchler, A. D., Shaw, K. M., Chiel, H. J., Quinn, R. D., et al. (2013). A segmental mobile robot with active tensegrity bending and noise-driven oscillators. In *Advanced Intelligent Mechatronics (AIM), 2013 IEEE/ASME International Conference on*, pages 1373–1380. IEEE.

[Wilson, 1961] Wilson, D. M. (1961). The central nervous control of flight in a locust. *J. exp. Biol*, 38(47):l–490.

[Wright et al., 2012] Wright, C., Buchan, A., Brown, B., Geist, J., Schwerin, M., Rollinson, D., Tesch, M., and Choset, H. (2012). Design and architecture of the unified modular snake robot. *2012 IEEE Int. Conf. on Robotics and Automation*, pages 4347–4354.

[Wright et al., 2007] Wright, C., Johnson, A., Peck, A., McCord, Z., Naaktgeboren, A., Gianfortoni, P., Gonzalez-Rivero, M., Hatton, R., and Choset, H. (2007). Design of a modular snake robot. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2609–2614. IEEE.

[Zhao et al., 2013] Zhao, Q., Ellenberger, B., Sumioka, H., Sandy, T., and Pfeifer, R. (2013). The effect of spine actuation and stiffness on a pneumatically-driven quadruped robot for cheetah-like locomotion. In *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on*, pages 1807–1812. IEEE.