Learning Tensegrity Locomotion Using Open-Loop Control Signals and Coevolutionary Algorithms

Atil Iscen*,**
Oregon State University

Ken Caluwaerts†
Ghent University

Jonathan Bruce‡
USRA
University of California at Santa Cruz

Adrian Agogino§
University of California at Santa Cruz
NASA Ames Research Center

Vytas SunSpiral§
SGT Inc.
NASA Ames Research Center

Kagan Tumer¶
Oregon State University

Abstract Soft robots offer many advantages over traditional rigid robots. However, soft robots can be difficult to control with standard control methods. Fortunately, evolutionary algorithms can offer an elegant solution to this problem. Instead of creating controls to handle the intricate dynamics of these robots, we can simply evolve the controls using a simulation to provide an evaluation function. In this article, we show how such a control paradigm can be applied to an emerging field within soft robotics: robots based on tensegrity structures. We take the model of the Spherical Underactuated Planetary Exploration Robot ball (SUPERball), an icosahedron tensegrity robot under production at NASA Ames Research Center, develop a rolling locomotion algorithm, and study the learned behavior using an accurate model of the SUPERball simulated in the NASA Tensegrity Robotics Toolkit. We first present the historical-average fitness-shaping algorithm for coevolutionary algorithms to speed up learning while favoring robustness over optimality. Second, we use a distributed control approach by coevolving open-loop control signals for each controller. Being simple and distributed, open-loop controllers can be readily implemented on SUPERball hardware without the need for sensor information or precise coordination. We analyze signals of different complexities and frequencies. Among the learned policies, we take one of the best and use it to analyze different aspects of the rolling gait, such as lengths, tensions, and energy consumption. We also discuss the correlation between the signals controlling different parts of the tensegrity robot.

Keywords
Evolutionary algorithms, tensegrity, locomotion, coevolution, fitness shaping

1 Introduction

Tensegrity robots are part of an exciting emerging field of soft-body robotics that, from a controls perspective, brings many challenges due to the nonlinear interactions between different elements.

* Contact author.
** Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331. E-mail: iscena@onid.oregonstate.edu
† Reservoir Lab, Ghent University, Ghent, 9000, Belgium. E-mail: ken.caluwaerts@nasa.gov
‡ University of California at Santa Cruz, Santa Cruz CA 95064. E-mail: jbruce@soe.ucsc.edu
§ NASA Ames Research Center, MS 269-3, Moffett Field, CA 94035. E-mail: adrian.k.agogino@nasa.gov (A.A.); vytas.sunspiral@nasa.gov (V.S.)
¶ Mechanical, Industrial and Manufacturing Engineering, Oregon State University, Corvallis, OR 97331. E-mail: kagan.tumer@oregonstate.edu

Fortunately, learning-based controls combined with the distributed nature of tensegrity robots can provide a simplified control solution to the fairly complex problem of the locomotion of tensegrity robots.

Tensegrity structures are based on a simple principle: The structure is composed of pure tension and pure compression elements. Disjoint axially loaded compression elements are encompassed within a network of tensional elements; thus, each element experiences either pure compression or pure tension. Based on this simple principle, by increasing the number of members and by changing their stiffness, the structures can be made arbitrarily complex and stiff.

Since the tensegrity structure does not have any bending or shear forces that must be resisted, individual elements can be extremely lightweight. Moreover, the majority of the structure is composed of tension elements that are significantly lighter than the compression elements. A unique property of tensegrity structures is how they can distribute forces internally. As there are no lever arms, forces do not magnify against joints or other common points of failure. Instead, externally applied forces are distributed throughout the structure via multiple load paths, creating system-level robustness and tolerance to forces applied from any direction. Thus, tensegrity structures can be easily reoriented and are ideally suited for operation in dynamic environments, in which contact forces cannot always be predicted.

Tensegrities have a number of beneficial properties. They are:

- *Lightweight:* Forces align axially with components, and shocks distribute through the tensegrity, allowing tensegrities to be made of lightweight tubes or rods and cables or elastic lines.
- *Energy-efficient:* Through the use of elastic tensile components and dynamic gaits, efficient movement is possible.
- *Robust to failures:* Tensegrities are naturally distributed systems and can gracefully degrade performance in the event of actuation or structural failure.
- *Capable of unique modes of locomotion:* Tensegrities can roll, crawl, gallop, swim, or flap wings, depending on construction and need.
- *Impact-tolerant and compliant:* Since forces are distributed upon impact, they can fall or bump into things at moderate speed. In addition, their compliance ensures that they do minimal damage to the objects they contact.
- *Naturally amenable to distributed control:* Characteristics of force propagation in tensegrities allow effective local controllers.

The last property is the most subtle but important. In traditional robots, distributed control becomes messy due to the need to communicate global state information to all the controllers with high precision, and thus it often undermines the very promise of distribution. Fundamentally, this stems from the fact that a rigidly connected structure will magnify forces internally through lever arms, and will accumulate force at joints. Thus, the actions of a local distributed controller can have disproportionate global consequences. These consequences can require a certain amount of global coordination and state management, undermining the value of the local controller. Tensegrity structures are different: Due to the tension network, there are no lever arms in them. Thus, forces *diffuse* through the structure, rather than accumulate at joints. As a result, actions by a local controller likewise diffuse through the structure, integrating with all the other local controllers. While any one local controller will impact the structure globally, that impact is locality-relevant and not magnified via leverage. Thus, the structure enables true distributed control, because local actions stay (predominately) local.

Considering the positive aspects of tensegrity robots discussed above, NASA is currently working on using a tensegrity robot for planetary exploration missions, including using the tensegrity robot as

an actual landing device, as part of the NASA Innovative Advanced Concepts (NIAC) program [1]. For this program, the SUPERball is close to the sphere-shaped, icosahedron tensegrity robot that is designed and currently under production at NASA Ames Research Center. In this work, we use the SUPERball as our tensegrity robot model, and our highly accurate tensegrity simulator, the NASA Tensegrity Robotics Toolkit (NTRT), as the simulation environment.

Despite their desirable properties, tensegrity robots have remained mostly a novelty for many years due to the properties that make them hard to control with traditional control algorithms, such as:

1. *Nonlinear dynamics:* A force generated on one part of the tensegrity propagates in a nonlinear way through the entire tensegrity, causing shape changes that further change force propagations.
2. *Complex oscillatory motions:* Tensegrity robots tend to have oscillatory motions influenced by their interactions with their environment.

Fortunately, the combinatorial optimization capabilities of learning-based controls are a natural match to these problems. Evolutionary algorithms can learn complex control policies that maximize a performance criterion without needing to handle the oscillatory motions and distributed interactions explicitly. Cooperative coevolutionary algorithms (CCEAs) enable different controllers, which are distributed throughout the tensegrity, to learn a cooperative task such as rolling locomotion. The set of learned policies for these controllers form a unified policy that provides locomotion of the whole robot.

In this article we provide a rolling locomotion algorithm for tensegrity robots. From this perspective, the robustness of the learned policy is more important than its optimality. Instead of an optimal rolling behavior that is highly sensitive to actuation noise, we prefer a suboptimal solution that can handle small changes in the policy. Coevolutionary algorithms are already known to favor robustness to non-optimality [37]. With numerous controllers and such a large search space, the time that coevolutionary algorithms take increases exponentially. To overcome this problem, we design a *historical-average* fitness-shaping method that favors robust solutions, while using random sampling to decrease the processing time between generations.

To obtain rolling locomotion for the SUPERball, we study the combination of open-loop signal controllers with coevolutionary algorithms. Being simple and distributed, these controllers can be readily implemented on the SUPERball hardware without requiring sensor information and precise coordination. Open-loop controllers are the first step to rolling locomotion and a good way to show the feasibility of rolling tensegrity robots. For this purpose, we analyze the rolling behavior from different perspectives and make sure that the control commands and the robot's behavior are within the limits of the hardware. We also analyze the similarities of the learned signals for different parts of the robot.

The rest of the article is organized as follows: Section 2 introduces necessary background information and related work. Section 3 defines the control problem and open-loop control schema that we use. In Section 4 we introduce a historical-average fitness-shaping method to promote robustness in coevolutionary algorithms. Section 5 presents the results on learning to roll for different types of signals. Section 6 analyzes the learned rolling behavior from different perspectives, such as movement, energy efficiency, and feasibility. Section 7 analyzes the correlation between the signals that control different parts of the robot while providing the rolling behavior. The article ends with Section 8, where we discuss conclusions and future research directions.

2 Background and Previous Work

Tensegrity structures are a fairly modern concept, having been initially explored in the 1960s by Buckminster Fuller [12] and the artist Kenneth Snelson [31]. The word tensegrity is formed from

the words "tensional integrity." The structure has an internal balance while the tensile elements encounter constant tension. The early research on tensegrities was first concentrated on the design and analysis of static structures [30, 3, 14]. The tensegrity principle was used in big structures, such as towers or bridges, as well as small structures, such as toys or furniture [16].

The concept of tensegrity is also being discovered in biological systems, from individual cells to mammalian physiology [13, 18]. Emerging biomechanical theories are shifting focus from bone-centric models to fascia-centric models, where fascia is the connective tissues (muscles, ligaments, tendons, etc.). In the *bio-tensegrity* model, bones are under compression, and a continuous network of fascia acts as the primary load path for the body. Inspired by this, since the cables of the robot are the tensional elements that can change length, we use the term "muscle" in the rest of the article.

Research into control of tensegrity structures was initiated in the mid-1990s, with initial efforts at formalizing the dynamics of tensegrity structures only recently emerging [30, 20, 38]. The very properties that make tensegrities ideal for physical interaction with the environment (compliance, multi-path load distribution, nonlinear dynamics, etc.) also present significant challenges to traditional control approaches.

For the first few decades, the majority of tensegrity related research was concerned with form-finding techniques [39, 19, 33, 26, 40, 22, 25]. The problem of finding the correct configuration to deform a tensegrity robot to a specific shape is already a complex problem due to the nature of these structures. To solve this problem, there are both algebraic studies as well as evolutionary algorithms. There have also been studies to generate different tensegrity structures using evolutionary algorithms.

With regard to locomotion, tensegrity robots are capable of providing different forms of locomotion due to the diversity of the shapes of their structures [11]. Some tensegrity locomotion studies use simple irregular tensegrities with three or four struts [24, 23]. One recent example is a tensegrity snake robot made of nested tetrahedral components that is capable of crawling over a wide range of terrains using a neurologically inspired control network of distributed central pattern generators (CPGs) [34]. Here, we are interested in icosahedron tensegrity robots, as shown in Figures 1 and 2, that are made of six struts with equal lengths and 24 muscles that connect the ends of the struts.

The icosahedron tensegrity provides additional advantages, such as rolling locomotion and deployability. The structure is the simplest tensegrity to provide an overall shape close to a sphere. It can handle external forces and impact with the ground easily, by deforming. As expected, the impact is diffused through the network of tensional elements. Moreover, the structure is easily collapsible to a star pattern by loosening tensional elements. These properties make the icosahedron tensegrity a great option as a lander and exploratory device for space missions. SunSpiral et al. explain the usage of an icosahedron tensegrity robot both as an entry, descent, and landing (EDL) instrument and as a mobile device for a mission to Saturn's moon Titan [32]. This idea

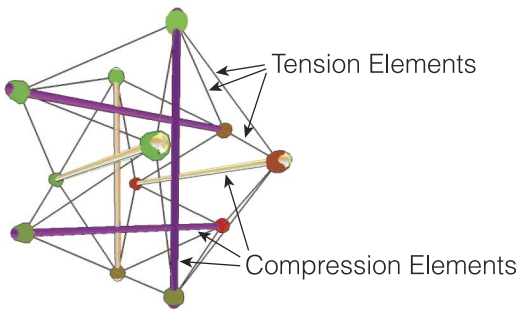


Figure 1. An icosahedron tensegrity structure composed of 6 compression and 24 tension elements.

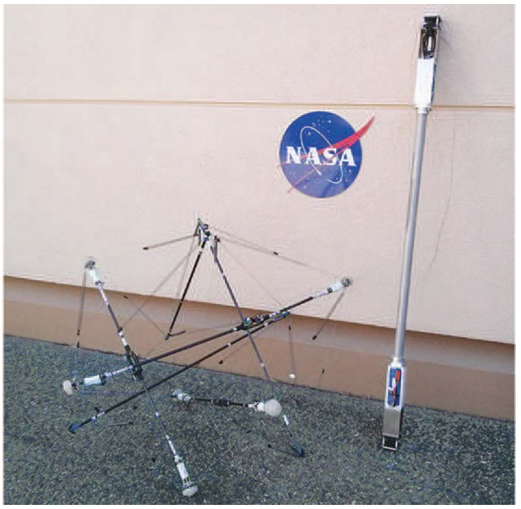


Figure 2. ReCTeR (left), a tensegrity robot, and one strut of the modular SUPERball (right) that is under development at NASA Ames Research Center.

of development and locomotion of an icosahedron tensegrity robot as a space exploration vehicle is part of a project funded by the NASA Innovative Advanced Concepts (NIAC) program [1].

The literature contains multiple studies about active control of icosahedron tensegrities [4, 5]. Shibata et al. analyze different surface patterns for rolling locomotion [29, 28, 17]. Rieffel et al. use vibration frequencies for locomotion without rolling [27]. Not just icosahedron tensegrities, but the entire field of active control of tensegrities is still fairly new. A recent review [35] shows that there are still many open problems in actively controlling tensegrities.

There are few hardware implementations of tensegrity robots. Koizumi et al. use a tethered icosahedron robot with pneumatic actuators to analyze base patterns for low-energy rolling [17]. Rieffel et al. use an icosahedron robot with vibrational motors analyzing forward motion with vibration [27, 26, 15]. Suitable for rolling locomotion, ReCTeR is the closest to the ideal icosahedron robot [9]. It is untethered and has six motors that control the lengths of six muscles. The robot is composed of a passive shell with 24 muscles. These 24 muscles are essential for the icosahedron. In ReCTeR, they are passive, but the structure has six additional active muscles that change their length for deformation and locomotion.

SUPERball is designed in a modular way so that each strut has two motors controlling two active muscles, and so that the struts can be used in tensegrity robots other than the icosahedron. Overall, the next prototype of SUPERball will have an active outer shell with 12 active muscles and 12 passive muscles. It will also be able to handle strong collisions and torque requirements for rolling locomotion on different types of terrains. In this article, we analyze the case for 24 active muscles to use the full capabilities of an icosahedron tensegrity, but reducing the number of the controllers is part of our future work [7].

2.1 SUPERball

The Spherical Underactuated Planetary Exploration Robot (SUPERball) is a tensegrity icosahedron robot currently under development at the NASA Ames Research Center (Figure 3) [6]. The main design goal of SUPERball is to be a more capable robot than a prior prototype called ReCTeR, to provide more reliable sensors, and to handle rougher environments.

ReCTeR is underactuated and can roll by actuating six tensile elements running through the center of the robot. A passive tensegrity icosahedron (24 tensile elements) forms ReCTeR's outer

shell; the six actuated members connect opposite sides of the shell. Hence, only 20% of ReCTeR's tensile members are actuated.

For SUPERball, the goal is to have the possibility to actuate all tensile elements. The current prototype has 12 out of 24 actuated tensile elements, but considering our future goal, this article studies the fully actuated robot. The main rationale for full actuation is that we aim to explore various control strategies (e.g., the one presented in this work) without hardware limitations and develop precise manipulation of capabilities (e.g., for end effector or payload positioning).

With a total mass of 1.1 kg (batteries included), ReCTeR cannot carry any scientific payload without a significant effect on the robot's dynamics. This is another reason to develop a new platform. SUPERball's mass with 50% actuated tensile members is around 20 kg. This larger mass will allow us to explore the behavior of tensegrity robots with a small payload suspended by tensile elements at the center of the robot [32]. Another important improvement is SUPERball's high power-to-weight ratio. ReCTeR's small brushed direct-current (dc) motors are often maxed out (25 W/kg), while SUPERball has significant headroom (>60 W/kg, final design > 100 W/kg). This allows for dynamic motion even in energetically suboptimal regimes.

Typical spring tensions are 5 to 20 N for ReCTeR, while the average tension in SUPERball's current configuration is 50 N. The SUPERball hardware is designed to handle tensile forces up to 500 N. One interesting aspect of the tensegrity icosahedron configuration is that the design can conveniently be scaled up or down. Forces scale approximately linearly as a function of the robot's mass. For practical purposes, SUPERball is deployed in its minimum-diameter configuration (struts approximately 1.7 m in length).

Increasing the diameter of the robot does not significantly increase its total mass, because lightweight hollow aluminum tubes are used to connect the end caps. More precisely, we are using 35-mm-diameter tubes with a 1.25-mm wall thickness. These tubes currently account for 15% of the robot's mass. Tripling the robot's diameter (4.5-m struts) with the same end caps would only increase this fraction to 41% (not considering buckling). This is an advantage over an airbag design, for which the surface area and, thus, the mass scale quadratically as a function of the diameter. It can easily be seen that the payload volume scales cubically as a function of the strut length. This trivially defeats the square-cube law, because the density of the structure drops as one increases the strut length.

To simplify our designs and to allow us to explore various morphologies, SUPERball is a highly modular platform. Its basic element is the end cap, and each end cap is independent, possessing actuation, battery power, various sensors, and wireless communication. In the current design, each SUPERball end cap houses a single 100-W brushless dc motor and approximately 70 W h of battery power. Thus, we can currently control twelve tensile members in the icosahedron configuration (50%). The sensory equipment of each end cap consists of two tensile force transducers, a torque sensor on the motor, and an inertial measurement unit with nine degrees of freedom.

To make the robot compliant, SUPERball has compression springs inside the tube connecting the end caps. The tensile elements are cable-spring assemblies. Each cable runs between two end caps, but on different struts. The cable runs from the opposing end cap to the spring end cap, passes through a cable housing assembly inside the spring end cap, and then connects to a compression spring located within the spring end cap's tube. A cable runs between two end caps, goes into a tube through an end cap, and then connects to a compression spring. In the case of an actuated tensile member, the opposing end cap of the cable connects to a motor or is simply fixed to the opposing

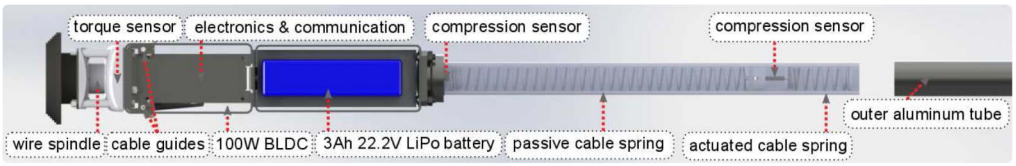


Figure 3. Cross section of the end of the rod for the current design of the SUPERball.

end cap for passive elements. To change the rest length of the tensile member, the motor assembly is a wire spindle that winds or unwinds the cable.

Passive and actuated cable-spring assemblies behave as linear springs with a given (possibly varying) rest length. This can be implemented in hardware in various ways. Active tensile elements are actuated by changing their rest length and (we will assume in this article) constant-velocity motors. While not entirely accurate from a hardware perspective, we will show in this work that the required motor power for our controllers is significantly below the motor's power capacity, which validates this simplification.

2.2 NASA Tensegrity Robotics Toolkit (NTRT)

Simulation of tensegrity robots is critical for application of intelligent controls. Simulation provides multiple advantages, such as being able to train controllers faster than training them on actual hardware, or using the simulator to explore design options and hardware requirements. In our work, we use the open source NTRT simulator that has been developed on top of the open source Bullet Physics Engine. The Bullet Physics Engine is a discrete-time-step, iterative physics simulator that handles collisions and interactions between different types of objects.[1]

The NTRT simulator relies on the Bullet Physics Engine to handle movement and collisions of rigid-body objects. On the other hand, to simulate the tensional elements, we needed precise elastic components whose lengths we could change to simulate active controls of the robot. For this purpose, we implemented our own tensional elements (muscles) that apply tension according to their stretch.

A muscle is attached to two different rigid bodies from specific points. We assume that muscles are abstract elements that apply force to these two rigid bodies according to their tensions. They have basic properties, such as rest length (length without stretch) and elasticity coefficient. The tension of a muscle is computed by looking at the current distance between two attachment points and the rest length of the muscle. For simulation purposes, we assume that the muscles have negligible weight compared to the rest of the structure. Moreover, we also ignore the interactions of the muscles with the rest of the environment. These two assumptions are plausible because the tensional elements stay inside the structure and do not interact with other objects except for extreme deformations. Also, the weight of the physical cables versus the weight of the rest of the robot is extremely low.

The active controls of the muscles are implemented by changing the rest length of the muscle at a constant speed (motor speed). This is modeled after motors that pull elastic cables to increase the tension of a specific elastic component.

In our previous work, we validated the NTRT with a physical tensegrity robot (ReCTeR) and precise motion capture [8]. The results showed that the simulator can correctly simulate tensegrity structures with a small margin of error for a static structure with active controls and a semi-static structure that moves step by step.

3 Controls Problem

Controlling a tensegrity robot brings multiple challenges, such as distributed controls, nonlinear interactions between components, and difficult-to-model dynamics, such as oscillations. For this reason, traditional centralized controllers and centralized designs are not a good match for a tensegrity robot. In contrast, we present a decidedly distributed approach for controlling a tensegrity robot. On the hardware side, the core of our design is an independently controllable rod containing two independent end-cap controllers on each side of the rod. This model naturally matches the distributed yet holistic nature of a tensegrity. The controllers act independently of each other but interact through the system, in that changing the length of one of the muscles affects the whole structure in a

1 Bullet Physics Engine, http://www.bulletphysics.org.

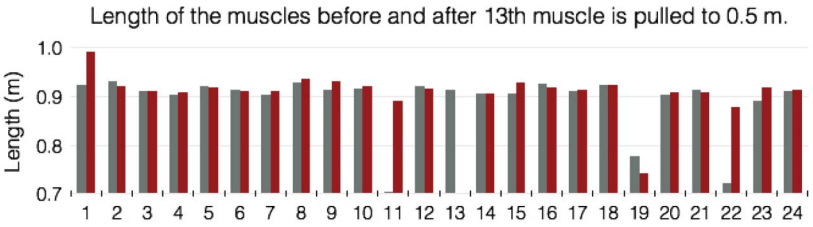


Figure 4. Change in length of the muscles when one of them (the 13th) is pulled to 0.5 m while the others keep the same rest length as before. Gray bars show the original length, and red show the final length. While the robot is at the exact same orientation, the actual lengths of the muscles change in a nonlinear way. Some of the muscles shorten due to the tension introduced by muscle 13, and some of them relax.

nonlinear way. This behavior can be seen in Figure 4. When we pull only one of the muscles (muscle 13), the lengths of all the muscles change; some of them increase while others decrease.

To facilitate distributed assembly, the controllers communicate via a Wi-Fi wireless network. This design allows for simplified construction and reduces cabling problems that could arise when the tensegrity robot needs to roll through adverse conditions. This design is not only distributed, but modular. For a simple six-bar tensegrity, we simply assemble six identical rods to form the robot. In addition, for more complex designs, additional rods can be used without changing the design of the rods themselves.

Our next challenge is to control a set of assembled rods into a high-performance tensegrity robot. To do this, we need controls that are able to work in a distributed control environment, and also work when wireless communication may not be high-bandwidth or reliable. To overcome this problem, we use distributed controls and distributed learning, where each controller learns its policy, but the overall behavior requires coordination of these controllers to make the tensegrity robot move. The setup described is a coordination learning problem where we have independent learners working towards a shared goal. The details of the learning distributed controls for this setup are described in Section 5.

An additional control challenge is how to handle the physical hardware limitations of each actuation system. Ideally, we would like our controller to be able to simply dictate the actual length of each muscle it is responsible for. However, due to the overall tension caused by the rest of the structure, the controllers can only provide the rest lengths of the muscles. Since the muscles are flexible, the controller changes the actual lengths.

In addition, hardware limitations also play an important role when tensions get higher. Since all the motors in the robot pull against each other, it is possible to reach tensions that the motors cannot handle. Moreover, since the rest of the structure can potentially overpower any one motor, there is a chance that a motor is back-driven and forced to feed out some of the cable stored on the spool. To address these limitations, if a muscle is experiencing tensions above the motor limit, and the cable is pulled to its maximum length of 1.1 m, the simulation is stopped, and the policy is considered infeasible.

To stay within the bounds of the physical hardware, we simulate the motors with high-level controllers that have a constant speed (0.2 m/s) while the tensions stay within reasonable limits. Indeed, the physical motors on the SUPERball can pull at a rate of 0.5 m/s within the tension range that we are dealing with, but we selected 0.2 m/s in order to leave hardware headroom, and also to lower the power consumption. While the motors move with constant velocity, the controllers dictate preferred positions for the motors. Dictating a preferred position is exactly the same as dictating a preferred rest length if the cables do not slip. At each time step, the motors pull or release their cables with a constant speed to get closer to their goal. While staying within reasonable tensions, this setup is feasible on the real robot. The assumption is that there is an intermediate controller layer that regulates the voltage versus torque in order to provide a constant rotation speed.

The overall goal of the controller is to have the tensegrity robot roll smoothly within the limitations of the actuation and communications hardware. To accomplish this, we use a periodic

open-loop controller with parameters that are set by an evolutionary algorithm (we have also performed research on closed-loop systems, but due to sensor feedback difficulties and overall increased complexity, we are focusing on open-loop controllers in this article). During rolling locomotion, the robot (and the controllers) will repeat the same motion (one full revolution) over and over. Because the rolling locomotion is a repetitive behavior, signals produced by the controllers will be periodic. The key to making this system work is determining the shape of this periodic signal.

Let's assume that the periodicity of the signal is t and we represent the signal as $F(x)$, x being time within the interval $[0, t]$. There are many possible ways to represent this control function. For instance, a natural choice would be a sine wave, or a series of overlapping waves to form more complex control policies. To reduce complexity, in this article we use an even simpler control model: We break down each control interval into subintervals and assign different preferred rest lengths for each subinterval. Subject to the limited velocity of the motors, the motor will slowly move to reach these selected points during the subintervals. The control model is essentially a set of overlapping square waves. As an example, we can divide one period into two subintervals, where the motor will have a preferred length of y_1 for the first half of the signal, and y_2 for the second half. With the motor moving towards y_1 and y_2, the resulting signal will be similar to that in Figure 5.

To generate a signal, the only parameters needed are the number of subintervals and the rest length values for each subinterval. For the specific example given in Figure 5, the number of subintervals is two, and y_1 and y_2 are the values of the preferred rest lengths for those intervals. The example given in Figure 5 is a simple signal, and because the number of subintervals is low, the complexity of signals that can be generated is limited. The complexity of possible signals increases with the number of subintervals. Due to this, we will, from now on, refer to this parameter as the complexity degree (n) of the signal. Depending on the complexity degree and the values of $y_1, y_2, \ldots, y_n$, the shape of the signal can change between a typical trapezoid, zigzags, stairs, and combination of those.

To summarize, the complexity of the signal depends on n, and each controller has n inputs, depending on the complexity selected. The rest lengths of the muscles follow this signal, and the actual lengths of the muscles change according to the activities of other muscles and the interaction of the robot with the environment. Each controller has a separate signal, and it controls only one of the motors. Twenty-four motors control twenty-four muscles independently, but all affect each other to achieve the common goal of rolling locomotion.

4 Historical-Average Fitness Shaping

Evolutionary algorithms are a family of biologically inspired learning methods, where new candidate solutions for a problem are generated, evaluated, and eliminated repeatedly [2]. Evolutionary algorithms consist of the cycle of forming new members, assigning fitness, and selecting the fittest

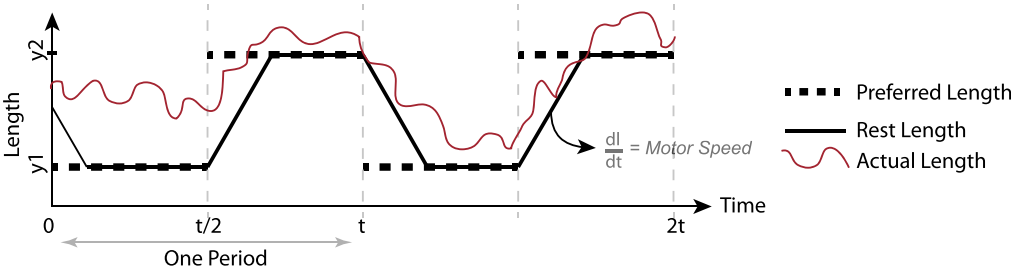


Figure 5. An example signal with two sub-intervals with preferred lengths y_1 and y_2 and periodicity t.

members. Each iteration of this cycle is called one generation, and it is repeated until the desired behavior is obtained.

The problem of tensegrity locomotion is distributed by its very nature. The controllers are independent, and no centralized control mechanism exists. To work on such problems, *cooperative coevolutionary algorithms* (CCEAs) are a class of algorithms that extend evolutionary algorithms. Instead of one centralized solution for a problem, there are multiple agents that collaborate to optimize an outcome [37]. Each agent has a separate population that evolves independently except in the evaluation phase. Since the task requires cooperation of agents to optimize the outcome, evaluating an agent requires placing it in a team of collaborators.

During each evaluation phase, each member is evaluated multiple times using different combinations of possible teammates. To move on to the elimination phase, each member has to be assigned a single fitness number, using the performances of the teams that it has been included in. This problem is called the *credit assignment problem*. The literature contains multiple approaches to credit assignment in CCEAs, such as assigning the maximum score (leniency), taking the average, or testing each candidate with the approximate best team so far (hall of fame) [36, 21, 10].

Leniency is also called *optimistic* credit assignment [21]. The idea behind lenient agents is to forgive poor performances caused by bad teammate combinations. Each candidate is tested in multiple teams, and the best score of each agent is taken as its shaped fitness. All the candidates in populations for different agents (since agents are coevolving, they have separate gene pools) receive the maximum score that they can reach using current potential teammates.

Optimizing a robotics locomotion algorithm, we have motivations beyond finding the best solution. In robotics, the physical environment and the hardware itself is full of noises that the locomotion algorithm has to handle. The physical robots experience both sensing and actuation noise that might cause problems for algorithms that prefer the best possible performance over robustness. Moreover, the motivation behind the study of tensegrity locomotion is not particularly finding the best possible locomotion. Instead, we work on showing that rolling locomotion is feasible for icosahedron tensegrity robots, which makes applicability and robustness have priority over getting the best result.

For problems where robustness is important, the fitness-shaping method should promote compatibility with different teammates as well as the maximum potential. We prefer a solution that would give similar performance with slight changes of initial conditions, instead of a great solution that might not work with a small amount of noise. One strategy to promote compatible solutions is to take the average score (instead of the maximum) of a candidate while testing it with randomly selected teams (instead of the best team). This is actually the default setup for CCEAs.

The problem with the average fitness rises with an increasing number of agents and an increasing population size. If the agent is tested with all possible teammates within that generation, the number of simulations per generation increases as the size of the population to the power of the number of agents. For a reasonable multiagent problem with 10 agents (a CCEA with the population size of 10), the number of experiments per generation becomes as much as 10^{10}.

To address this problem, random sampling is a commonly used method. Random sampling decreases the number of evaluations between two generations. Especially when using average performance as the fitness, it is possible to get an approximation with a much smaller number of evaluations. A good sampling size depends on the problem and the generation. A small number of evaluations can result in a higher error margin. Higher sampling values will result in a closer approximation of the average fitness with all possible teammates, but it increases the number of evaluations per generation and therefore the computational complexity of the algorithm.

In evolutionary algorithms, the fittest candidates survive to the next generation. During future generations, each team member's gene pool contains different candidates. The old candidates that survived previous generations are now evaluated with new team members. If these surviving agents are not modified with a mutation, they are actually evaluated with more teammates over the course of multiple generations. The historical-average method proposes to use the history of evaluations that these surviving agents collect.

Algorithm 1 (One generation of CCEA with historical average):

```
Data: Population of n elements for each agent
 1 for i = 1..k do
 2 |  randomTeam ← ∅;
 3 |  forall the Populations do
 4 |  |  randomTeam ← randomAgent;
 5 |  end
 6 |  score = evaluate(randomTeam);
 7 |  forall the agents ∈ randomTeam do
 8 |  |  agent.history ← score;
 9 |  end
10 end
11 forall the Populations do
12 |  forall the agents do
13 |  |  agent.fitness = average(agent.history);
14 |  end
15 |  order population according to the fitness;
16 |  eliminate the last z members;
17 |  copy the first z to the last z;
18 |  mutate the last z;
19 |  clear history for the last z;
20 end
```

Algorithm 1 shows the general flow of one generation of CCEA with historical average. Lines 1–10 show the random sampling for selecting teams. For each teammate of the selected teams, the team score is added to each team member's history (line 8). For assigning the fitness, each agent's fitness is selected as the average of all the scores in the history (line 13). Lines 15–19 are the elimination and breeding phase of coevolution.

As an example, consider a system with three teammates and a pool size of 10 (30 candidates total). Instead of all possible team combinations (10^3), we choose to use random sampling with n evaluations. After n evaluations, let's say that the fittest 5 of each pool survives for the next generation (15 total). The bottom 5 of each pool are replaced with new candidates, and these 30 candidates are evaluated with another n evaluations using randomly selected teams.

For each random team, the chance of a specific candidate being selected is $\frac{1}{5}$. After n random samplings, each candidate will have been evaluated approximately $\frac{n}{5}$ times on average. With historical averaging, the previous top 5 that survived for this generation will use their previous evaluations together with the new ones to take the average up to approximately $\frac{2n}{5}$. For a candidate that survives k generations, the size of its history becomes $\frac{kn}{5}$. The fitness of that candidate is the average of those $\frac{kn}{5}$ evaluations during the last k generations.

The purpose of taking the average of these evaluations is to approximate the average compatibility of that candidate with other possible teammates. For the best candidates that survive multiple generations, our approximation gets better with an increasing history size. Although we use a small number of evaluations per generation, the longer these agents survive, the better their fitness approximation gets.

One concern with the historical average can be the fact that a good and long-surviving candidate is still judged with the teammates that coevolutionary algorithms had during the early generations. One can argue that CCEA produces better teammates that will provide better results. On the other hand, since the size of the history increases by $\frac{kn}{5}$ at each generation, the effect of the evaluations using past (and possibly worse) teammates decreases over time. In addition, at every generation, these surviving candidates will have offspring that can possibly perform better with the new-generation teammates. The higher evaluation scores that these offspring receive will eliminate their ancestors that survived many generations.

As an overview, on using the historical average in CCEAs:

- The average score allows more robust solutions.
- The number of evaluations per generation is decreased by using random sampling of possible teams.
- The surviving candidates keep their history of evaluations. They grow a larger history to take the average of; therefore they have a better approximation of their fitness.
- Each candidate receives a shaped fitness according to both current possible teammates and their past.

From a biological inspiration perspective, if we analyze the resulting behavior, surviving candidates use their past experience to better approximate their fitness. On the other hand, past experience relies on past teammates that are possibly worse than current teammates. If there is a big difference between current and past teammates, new offspring of these experienced candidates obtain a better average, since they are only experienced with the new teammates. As a result, new candidates eliminate old experienced candidates, and the cycle continues as expected. The whole process brings to mind the interactions between different generations of human beings. The good candidates survive longer, as usual. Surviving candidates use their experience to shape their fitness. Since their teammates evolve and change, if their teammates get better, they cannot compete with new offspring that are more compatible with new-generation teammates.

There are open-ended questions about the historical-average method, such as the effects of the amount of random sampling, the average history size before and after convergence, the variation within gene pools, and the robustness of the produced results. Through the next sections, we use the historical average to solve the tensegrity locomotion problem, and we investigate its performance and these open-ended questions.

5 Learning to Roll

The control parameters to generate the signal are explained in Section 7. While the signals are simple, the interaction between these signals to reach a rolling behavior is highly complex. As explained before, the nonlinear and oscillatory nature of the problem makes the tensegrity hard to control with classical control methods. The consequences of specific signal combinations can be simulated, but finding the correct signal parameters for a specific behavior is not possible. In this section we explore how we can use the simulation combined with a historical average to evolve a set of control parameters that will lead to the desired behavior.

The problem setup is episodic; the agents have 60 s to test their policies. At the end of each episode these candidates are evaluated according to their performance. In the rolling tensegrity problem, we measure performance as the distance covered in 60 s. Formally, the evaluation is defined as

$$f = d(y_{0,0}, y_{0,1}, \ldots, y_{0,n}, y_{1,0}, \ldots, y_{24,n}), \tag{1}$$

where $y_{i,j}$ is the rest length for the ith controller and jth subinterval. Depending on the complexity of the signals (n) selected, there are $24n$ parameters to learn. Note that the decomposition of the distance function d is not readily obtainable in closed form. Instead, it must be computed from observing simulations or measured from a physical implementation. Also, note that our evaluation does not explicitly take any behavior into account besides the distance moved (final position − initial position). Tensegrities can exhibit many different gaits, ranging from hopping to rolling; and many different paths, ranging from spirals to straight lines. However, tensegrities that maximize final versus initial position tend to roll in one direction. Deviations from this pattern tend to hurt performance.

First, we compare historical average, lenient learners, and a regular coevolutionary setup. As we discussed in Section 4, we expect to see that the historical average produces more robust solutions. One way to validate this conclusion is to look at the average score per generation. At every generation, agents are tested with random teammates multiple times. Each population contains slightly modified versions of the best candidates. As a result, the average score of that generation shows the candidates' ability to work with slightly modified teammates. On the other hand, the maximum score that we obtain at each generation is the score of the best team that we have obtained so far.

Figure 6 shows both the best and the average score per generation for three algorithms. We test standard CCEA, lenient learners, and historical average. All three algorithms use random sampling of 50 teams per generation. When we look at the maximum scores reached by the three algorithms, we find that they all converge to the same policy. Leniency converges faster, since it favors best solution over robustness. On the other hand, if we look at the average scores per generation, we find that leniency converges to a lower point than average fitnesses. This means that the solutions that the historical average produces can handle variations of teammates in a better way. Moreover, the average score of the historical-average method keeps improving even after the convergence of the best team.

Second, we show an example learning session using signals with a complexity (n) of 5 and a period (t) of 4 s. Figure 7 illustrates the distance rolled by the robots over the course of learning. Starting with 0 m, the robots converge, rolling over 32 m in 60 s. This result shows that successful learning of rolling locomotion using CCEA is possible. In Section 3, we discussed when a policy is labeled as unfeasible during learning. The second line in the same figure (Figure 7) shows the rate of unfeasible policies that are tried while learning to roll. On converging to rolling locomotion, unfeasible policies drop to zero. This shows that the learned policy lies within reasonable lengths and tensions; moreover, it is also far from the limits, since small mutations tried during the evolution are also feasible.

Considering that the robot has a shape that is similar to a sphere with a diameter of 1.5 m, rolling 32 m means approximately seven revolutions in a minute. This means 8 s per revolution. Considering that we selected 4 s as the periodicity of the signal, the learned signals provide a half revolution, and applying the same signals again results in the other half of the complete revolution. This

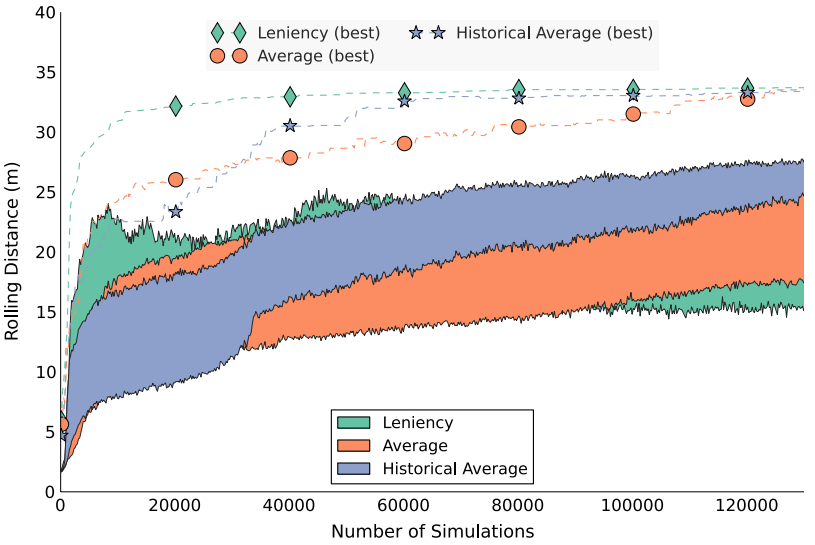


Figure 6. Comparison of three methods of fitness assignment when used with random sampling. Both the score of the best team and the average score per generation are shown. Although leniency reaches the best team faster, the average score per generation is lower. The best score of the historical-average method reaches the same value, and the average score per generation is much higher than for leniency. Solutions found by the historical-average method not only reach the same best score; they are also more compatible with variations of its teammates.

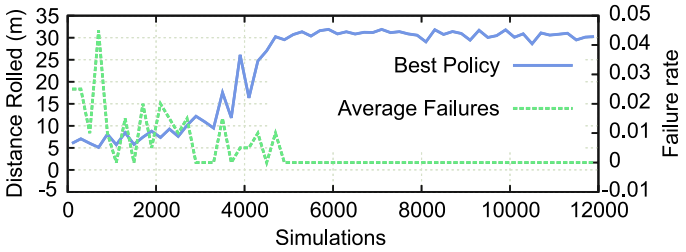


Figure 7. The performance of the robots during the learning session for signals of a complexity of 5 and a period of 4 s. As a side result, the percentage of the policies that failed to stay within reasonable limits is shown in the second line.

supports the reasoning behind selecting periodic signals to obtain rolling locomotion as a periodic movement of the robot.

The first set of experiments illustrated when n and t are selected as 4 and 5 (4 s with a complexity of 5). Next, we investigate the results of learning using signals with different complexity and periodicity. Figure 8 shows the converged behaviors when we fix n at 5 and learn using variable t. Note that the signal used for different values of t is not the same. The signals are learned from scratch for each value of t.

Clearly, the peak is when the signals have periods of 4 s (a frequency of 0.25 Hz). When we shorten the period below 4 s, the robot cannot learn to roll. One might think that providing the same signal with a higher frequency could provide the same rolling behavior, but when the tensegrity deforms to start rolling with a higher speed, the contact forces from the ground and the reaction of the structure change completely. When we increase the period beyond 4 s, the frequency drops and the performance gradually decreases as expected. Moreover, the rolled distance is linearly proportional to the frequency. For periods of 4 to 8 s, the performance divided by the frequency gives the same value $\left(\frac{33}{1/4} \simeq \frac{27}{1/5} \simeq \frac{22.5}{1/6} \simeq \frac{19}{1/7} \simeq 132\right)$.

Next, we investigate the effects of a different signal complexity level on the learning. The period is fixed at 4 s, because it gave the best score combined with the complexity of 5 in the previous set of experiments (Figure 8). We started with a complexity of 2, where the signal is as simple as possible. The signal alters between one high value for the first 2 s and one low value for the last 2 s. We increase the complexity to 9. The result is illustrated in Figure 9. The first conclusion is that signals with a complexity of 2 cannot succeed in learning to roll, but the performance improves with higher complexity. Clearly the controllers need more complex signals to provide rolling locomotion. The peak performance is reached at a complexity of 5, where the preferred length alters between five different points during five intervals of 0.8 s each.

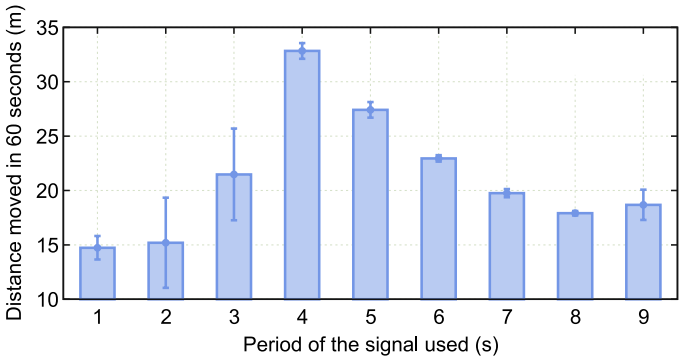


Figure 8. The performance of the converged policies after learning for signals with periods of different lengths, while the complexity is fixed at 5. The best performance is reached with signals that are repeated every 4 s. Signals with longer periods have decreasing performance inversely proportional to the periodicity.

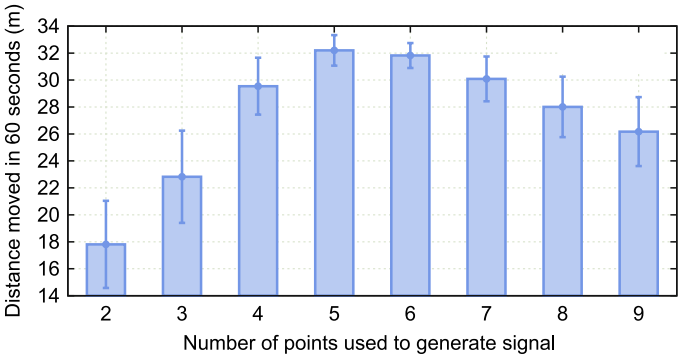


Figure 9. The performance of the converged policies after learning for signals with different complexity levels, while the periodicity is fixed to 4 s. The best performance is reached with signals that use five points. Less complex signals cannot generate rolling locomotion, and more complex signals are hard to learn.

The second conclusion of this experiment is seen when the complexity is increased even further. The learned behavior gradually decreases, and error bars show that the statistical significance goes down. The reason behind this behavior is that the parameters to learn increase linearly, and the problem to learn becomes linearly harder for each controller. Since all the controllers learn simultaneously while interacting with each other, the overall difficulty of the problem is increased even further. The error bars show that in more complex problems, some statistical tests achieve good results while some of them fail completely due to the difficulty of the problem.

6 Analyzing the Rolling Behavior

In the previous section, we showed that learning to roll for a tensegrity robot succeeds with signals of correct periodicity and complexity. In this section, we look at the learned behavior and analyze the feasibility of the behavior, lengths and tensions during rolling, converged signals, and robustness of the behavior.

As a sample learning behavior, we select one of the learned behaviors with a period of 4 s and a complexity of 5. The learning process for this behavior is illustrated in Section 5 and Figure 7. For each simulation, the robot tests different policies for 60 s, and the distance moved is marked as the score. The policies that are tested are updated according to the cooperative coevolutionary algorithm with a historical average (Algorithm 1). For this particular experiment, the robots reached the performance of rolling 32 m/min in approximately 5,000 simulations. As a side result, the percentage of failed policies (due to high tension) reached zero.

First, we take this learned behavior and look at the learned policy. Figure 10 shows the intervals that each muscle's length lies within. The muscles' lengths vary from 0.7 to 1.1 m. While some of the muscles, such as numbers 1 and 21, have minimal change in length, some of them, such as numbers 13 and 23, have larger changes and larger intervals. Another remark is that the mean of the signal (noted by the red horizontal line) is not necessarily at the center of the interval that the signal lies within. This is one difference that more complex signals provide. For example, the length of muscle 1 is mostly around 0.82 m, but it reaches as low as 0.7 m once in a while.

One way to analyze the rolling behavior is by looking at the average lengths and tensions of the muscles in addition to the potential and kinetic energy of the structure. First, we look at how the actual lengths of the muscles change compared with the signals provided. Figure 11a shows the average rest length of the muscles (signal provided) and the average actual length of the muscles. The separation between the two lines shows the stretch of the muscles due to the tension. The most interesting fact about this graph is the difference between frequencies for the two lines. Although the signals that are provided to the muscles repeat themselves every 4 s, the actual lengths repeat

themselves every 8 s. This supports our previous conclusion about using signals that have a periodicity of 4 s that can result in revolutions that take 8 s. The first and second halves of the roll use the same signal, but ground interactions make the actual lengths differ.

The average and maximum tensions of the muscles during rolling are illustrated in Figure 11b. The average tension is low and stays around 60 N. The second line shows the tension of the muscle with the longest stretch at each particular time of the simulation. Its value goes up to 200 N, staying within values that our hardware design can handle. The maximum tension graph also repeats itself every eight seconds as expected.

When we observed the gait learned using our simulator, we see that the rolling locomotion does not have a constant speed. Instead, it slows down and speeds up periodically during each revolution. To illustrate this behavior, we looked at the total kinetic energy of all the rods over time as seen in Figure 11c. If we take the interval between two peak points (when $t = 27$ and $t = 35$), the kinetic energy stays at zero for one second, which is around $t = 30$ s. Moreover, the repetitive acceleration and deceleration can clearly be seen. This behavior creates an inefficiency in terms of energy for the gait. There are two main reasons for this behavior: First, the learning algorithm only optimizes the distance rolled, not the energy spent during motion. Optimizing more than one criterion is actually part of our future work. Second, in this work, we are testing open-loop controllers. By using some feedback from the robot (such as lengths, tensions, or orientation), a smoother rolling experience may be possible. This is also another aspect of future work that we explain in Section 8.

Next, we observe the potential energy stored in the muscles. Figure 11d shows the pattern, which repeats itself every 8 s as expected. The first and the second four seconds are similar, but they differ slightly due to different reactions with the environment during the second half of a complete roll. The pattern shows an overall behavior of increasing the potential energy slowly over time, and releasing it. This matches the kinetic energy behavior that we observed, as seen in Figure 11c. The kinetic energy of the structure increases during the few seconds following the moment at which potential energy is released (i.e., $t = 25$ s).

The last set of experiments analyzes the approximate power usage by the motors during rolling locomotion. In simulation, the power consumption is approximated using the current tension of the element and the constant speed with which the motors shorten the muscles. As we explained earlier, the learned behavior is not optimized to be power-efficient for this study. On the other hand, we want to make sure that the required power is within the limits of the motors and batteries that will be used in the hardware. Figure 11e illustrates the average power consumption of the muscles, which varies between 2 and 6 W per muscle. Considering that the motors always pull against the

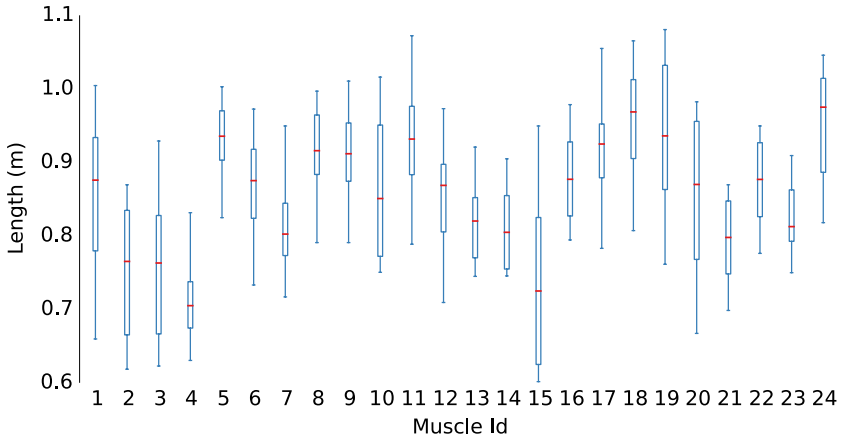
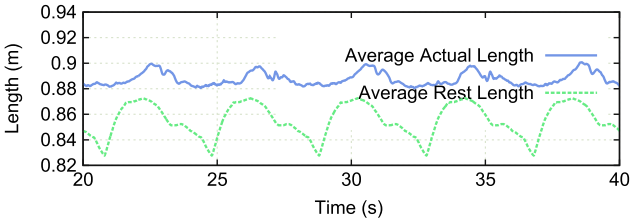
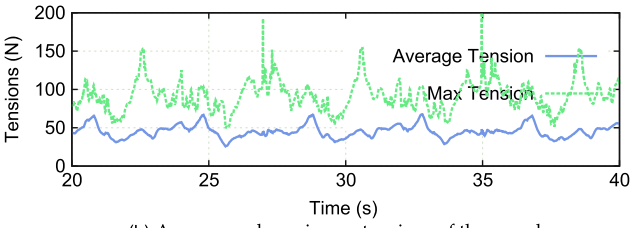


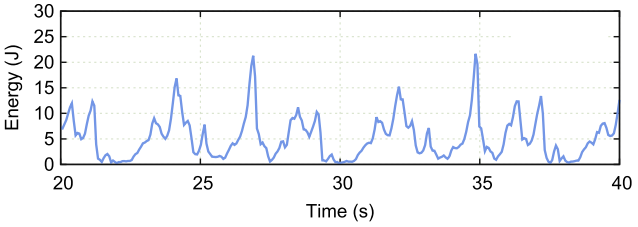
Figure 10. A sample learned policy for 24 motors is illustrated. For each signal, the red line at the center shows the mean of the signal, and the box and dashed lines show the interval that the signal lies within.



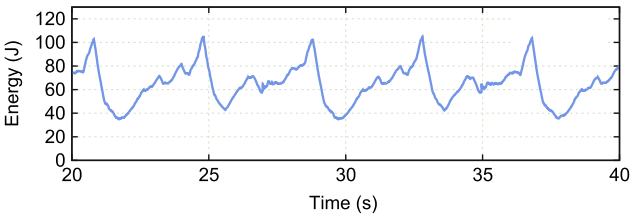
(a) Average rest lengths and actual lengths of the muscles



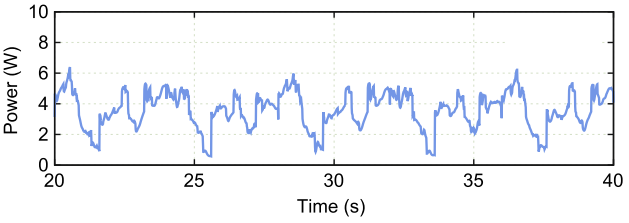
(b) Average and maximum tensions of the muscles



(c) Kinetic energy of the tensegrity robot



(d) Total potential energy stored in muscles



(e) Power used by the motors to roll

Figure 11. Illustration of different aspects of the tensegrity robot over time, during rolling locomotion. The signals used for the muscles repeat themselves every 4 s. The tensegrity robot completes one revolution in 8 s. Tensions, lengths, and power usage of the robot stay within our defined hardware limits.

tension (and all the muscles are tight all the time), this value is rather low. Moreover, it will be possible to lower it further by using a feedback controller in future work.

7 Analyzing the Roles of Different Muscles

Next, we analyze the signals further by looking at their shapes and the correlation between them. The top half of Figure 12 shows all 24 signals when they are normalized between zero and one. The purpose of this experiment is to show the similarities of the signals and different types of signals learned at the end of coevolution. First, we look at the correlation between signals. For each signal, the red-yellow-red area highlights the interval that maximizes the correlation with other signals. In this ordering, it is hard to see the similarities between signals. As shown in the bottom half of Figure 12, we shift the signals so that their selected intervals match; then we use hierarchical clustering to group the signals according to the similarity metric.

After reordering the signals (Figure 12, bottom half), the results show that half of the signals are relatively simple, with one peak high and one peak low, whereas the other half have two peak points per cycle. This result leads to several conclusions. First, the learning algorithm makes use of the complexity provided. Although a subset of the signals is simple, another subset has more complex

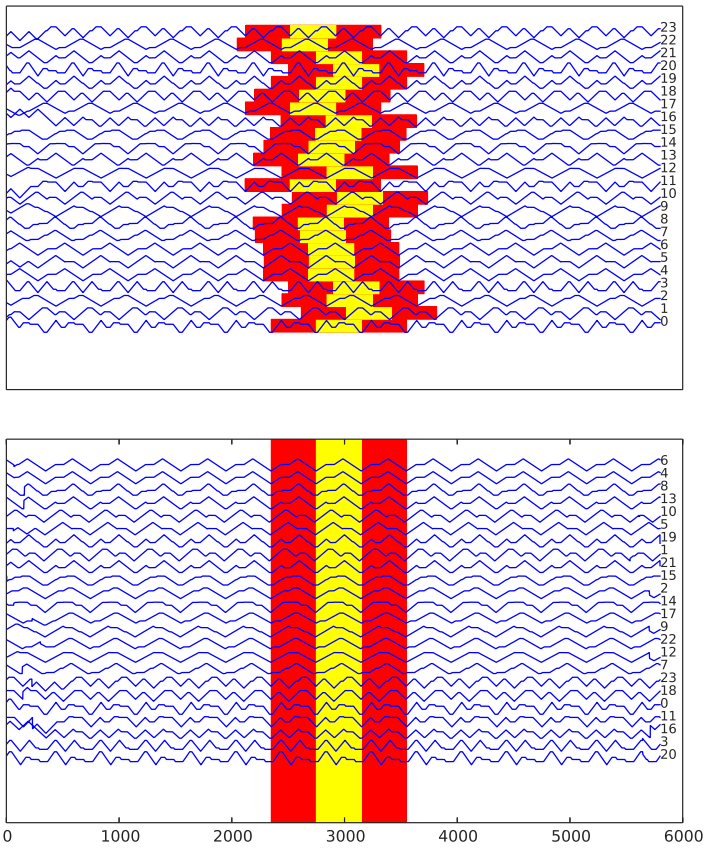


Figure 12. The upper graph shows the 24 signals used by the 24 controllers. These signals are normalized in $[i, i + 1]$ for muscle i. All the signals have the same frequency, and highly correlated intervals are marked with red-yellow-red colors. The lower graph shows the same signals after they are shifted so that highlighted areas synchronize, and the signals are reordered according to hierarchical clustering of the correlation matrix. As a result, the similarity between signals shows that the muscles use out-of-phase versions of similar signals.

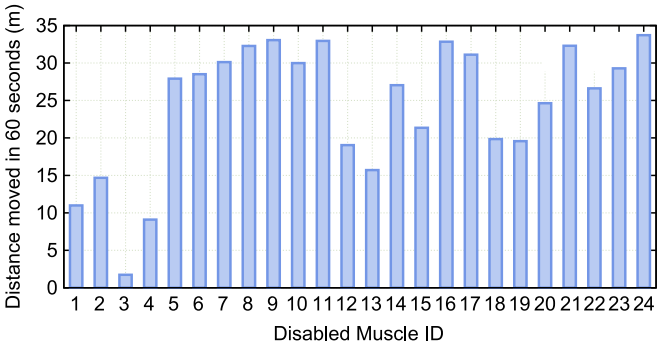


Figure 13. The performance of the learned policy when one of the muscles is disabled. Learned policy is partially robust to failures of some muscles.

signals with multiple peak points that can only be generated with complexity coefficients that are higher than 3. The subsets of the signals that are similar can be regenerated using different parameters, but with the same formula. Let's consider a normalized signal $f(x)$. The formula $g(x) = A + B \cdot f(x + C)$ can produce similar signals for different values of A, B, and C. Using this idea, all 24 signals can be reproduced using three or four base functions and different parameters. This gives us a hint about why many articles in the literature propose to use central pattern generators (CPGs) to control tensegrity robots.

The last set of results shows how critical each muscle is for a given rolling locomotion. Taking the tensegrity robot with the learned policy, we disable one of the muscles and observe the effect of such a failure on its overall behavior. Figure 13 shows that the performance depends on which muscle fails. For a significant number of muscles, using the same algorithm still provides rolling behavior with similar performance. Notably, though, some muscles' roles are critical to the algorithm's success. Since the structure is symmetric, one can expect to have a correlation between the effects of disabling similar muscles. On the other hand, disabling muscle 3 breaks the rolling behavior. During the rolling locomotion, the expectation is confirmed. On the other hand, since the robot is stationary at start, some muscles (such as muscle 3) play more important roles in triggering the first movement. Otherwise, disabling the muscles can decrease the performance, but rolling locomotion can still be performed with an underactuated version of the robot.

8 Conclusions and Future Work

Tensegrity robots have great potential with their flexibility, deployability, robustness, and different forms of locomotion. Due to these advantages, tensegrity robots are considered good candidates for space exploration and mobility missions. On the other hand, controlling tensegrity robots brings multiple challenges due to their distributed nature and the nonlinear interactions between their components. Controlling a tensegrity robot to achieve rolling locomotion with classical control methods is difficult to figure out.

In this work, we used coevolutionary algorithms combined with open-loop signals in order for an icosahedron tensegrity robot to learn distributed rolling locomotion. We used the model of the robot that is currently under production at NASA Ames Research Center. As a simulator, we used the NASA Tensegrity Robotics Toolkit (NTRT), a simulator for tensegrity robots that closely matches reality. For learning, we analyzed the results of using signals with different complexity levels and frequencies. Results show that by selecting the right type of signals, a tensegrity robot's learned behavior can reach a decent rolling speed of 32 m/min.

To study the learned behavior, we used one of the learned policies and analyzed the signals used versus the actual lengths of the muscles and the tensions and energy stored in the structure. Results

show that one complete revolution takes 8 s. Muscles use the same signal for the first and second halves of the revolution. Moreover, the rolling behavior is not smooth, and the robot repeatedly accelerates and decelerates during rolling. This causes inefficiency, but that is understandable, since the learning algorithm does not optimize energy consumption.

Finally, we analyzed the signals of different muscles and concluded that while some muscles can function with simple signals, others require complex activation patterns. Additionally, there is significant similarity and overlap among set signals that implies interchangeability of the muscles. This observation leads us to conclude, with regard to robustness, that while some muscles play a critical role in rolling, the algorithm we used can handle failures in some muscle groups.

In conclusion, we show that coevolution combined with open-loop signals can provide rolling locomotion. Moreover, learned behavior is feasible for the designed hardware that is under production.

Currently, there are still unanswered research questions related to tensegrity rolling locomotion. First, using multi-objective optimization techniques, we are currently investigating ways of minimizing energy consumption and obtaining a smoother rolling behavior. Second, using feedback for the controllers can increase the overall ability of locomotion. The algorithm can use the information about the interaction of the structure with the environment. This future research direction can address problems such as rolling on uneven terrains, rolling uphill, and rolling when there are external forces that disturb or interrupt locomotion.

Acknowledgments

This research was supported by the NASA Innovative Advanced Concepts (NIAC) program. Ken Caluwaerts was supported by a Ph.D. fellowship of the Research Foundation–Flanders (FWO). Support also came from NSF Graduate Research Fellowship DGE1106400, and from NASA Prime Contract NAS2-03144 awarded to the University of California, Santa Cruz, University Affiliated Research Center.

The authors would like to thank Andrew P. Sabelhaus, Alice Agogino, Brian Tietz, Terry Fong, Greg Orzech, and the NASA Ames Intelligent Robotics Group.

References

1. Agogino, A., SunSpiral, V., & Atkinson, D. (2013). Super Ball Bot—structures for planetary landing and exploration. *NASA Innovative Advanced Concepts (NIAC) Program, Final Report*.
2. Back, T., Fogel, D. B., & Michalewicz, Z. (1997). *Handbook of evolutionary computation*. Bristol, UK: IOP Publishing Ltd.
3. Bel Hadj Ali, N., Rhode-Barbarigos, L., Pascual Albi, A., & Smith, I. (2010). Design optimization and dynamic analysis of a tensegrity-based footbridge. *Engineering Structures*, *32*(11), 3650–3659.
4. Böhm, V., Jentzsch, A., Kaufhold, T., Schneider, F., & Zimmermann, K. (2011). An approach to compliant locomotion systems based on tensegrity structures. In *Proceedings of the 56th IWK, TU Ilmenau*.
5. Böhm, V., Zeidis, I., & Zimmermann, K. (2015). An approach to the dynamics and control of a planar tensegrity structure with application in locomotion systems. *International Journal of Dynamics and Control*, *3*(1), 41–49.
6. Bruce, J., Caluwaerts, K., Iscen, A., Sabelhaus, A., & SunSpiral, V. (2014). Design and evolution of a modular tensegrity robot platform. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*.
7. Bruce, J., Sabelhaus, A., Chen, Y., Lu, D., Morse, K., Milam, S., Caluwaerts, K., Agogino, A., & SunSpiral, V. (2014). Superball: Exploring tensegrities for planetary probes. In *12th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*.
8. Caluwaerts, K., Despraz, J., Iscen, A., Sabelhaus, A. P., Bruce, J., Schrauwen, B., & SunSpiral, V. (2014). Design and control of compliant tensegrity robots through simulation and hardware validation. *Journal of the Royal Society Interface*, *11*(98), 20140520.
9. Caluwaerts, K., D'Haene, M., Verstraeten, D., & Schrauwen, B. (2013). Locomotion without a brain: Physical reservoir computing in tensegrity structures. *Artificial Life*, *19*(1), 35–66.

10. Colby, M., & Tumer, K. (2012). Shaping fitness functions for coevolving cooperative multiagent systems. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems—Vol. 1* (pp. 425–432). International Foundation for Autonomous Agents and Multiagent Systems.

11. Fujiia, M., Yoshiia, S., & Kakazub, Y. (2006). Movement control of tensegrity robot. *Intelligent Autonomous Systems 9: IAS-9*, *9*, 290–297.

12. Fuller, B. (1961). Tensegrity. *Portfolio & Art News Annual*, *4*, 112–127.

13. Ingber, D. E. (1993). Cellular tensegrity: Defining new rules of biologic design that govern cytoskeleton. *Journal of Cell Science*, *104*, 613–627.

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

15. Khazanov, M., Jocque, J., & Rieffel, J. (2014). Evolution of locomotion on a physical tensegrity robot. In *ALIFE 14: The Fourteenth Conference on the Synthesis and Simulation of Living Systems* (pp. 232–238).

16. Klimke, H., & Stephan, S. (2004). The making of a tensegrity tower. In *Proceedings of the IASS Symposium 2004*, Montpellier.

17. Koizumi, Y., Shibata, M., & Hirai, S. (2012). Rolling tensegrity driven by pneumatic soft actuators. In *2012 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 1988–1993). IEEE.

18. Levin, S. (2002). The tensegrity-truss as a model for spine mechanics. *Journal of Mechanics in Medicine & Biology*, *2*, 375–388.

19. Masic, M., Skelton, R. E., & Gill, P. E. (2005). Algebraic tensegrity form-finding. *International Journal of Solids and Structures*, *42*, 4833–4858.

20. Motro, R. (2003). *Tensegrity: Structural systems for the future*. London: Butterworth-Heinemann.

21. Panait, L., Tuyls, K., & Luke, S. (2008). Theoretical advantages of lenient learners: An evolutionary game theoretic perspective. *The Journal of Machine Learning Research*, *9*, 423–457.

22. Paul, C., Lipson, H., & Cuevas, F. J. V. (2005). Evolutionary form-finding of tensegrity structures. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, GECCO '05* (pp. 3–10). ACM: New York.

23. Paul, C., Roberts, J., Lipson, H., & Cuevas, F. (2005). Gait production in a tensegrity based robot. In *Proceedings, 12th International Conference on Advanced Robotics, 2005. ICAR'05* (pp. 216–222). IEEE.

24. Paul, C., Valero-Cuevas, F., & Lipson, H. (2006). Design and control of tensegrity robots for locomotion. *IEEE Transactions on Robotics*, *22*(5), 944–957.

25. Pugh, A. (1976). *An introduction to tensegrity*. Berkeley: University of California Press.

26. Rieffel, J., Valero-Cuevas, F., & Lipson, H. (2009). Automated discovery and optimization of large irregular tensegrity structures. *Computers & Structures*, *87*(5–6), 368–379.

27. Rieffel, J. A., Valero-Cuevas, F. J., & Lipson, H. (2010). Morphological communication: Exploiting coupled dynamics in a complex mechanical structure to achieve locomotion. *Journal of the Royal Society Interface*, *7*, 613–621.

28. Shibata, M., & Hirai, S. (2010). Moving strategy of tensegrity robots with semiregular polyhedral body. In *Proceedings of the 13th International Conference on Climbing and Walking Robots (CLAWAR 2010), Nagoya* (pp. 359–366).

29. Shibata, M., Saijyo, F., & Hirai, S. (2009). Crawling by body deformation of tensegrity structure robots. In *IEEE International Conference on Robotics and Automation, 2009. ICRA'09* (pp. 4375–4380). IEEE.

30. Skelton, R. E., & De Oliveira, M. C. (2009). *Tensegrity systems* (2009 ed.). Heidelberg: Springer.

31. Snelson, K. (1965). Continuous tension, discontinuous compression structures. United States Patent 3169611.

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

33. Tibert, A. G., & Pellegrino, S. (2003). Review of form-finding methods for tensegrity structures. *International Journal of Space Structures*, *18*, 209–223.

34. Tietz, B. R., Carnahan, R. W., Bachmann, R. J., Quinn, R. D., & SunSpiral, V. (2013). Tetraspine: Robust terrain handling on a tensegrity robot using central pattern generators. In *2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)* (pp. 261–267).

35. Tur, J. M. M., & Juan, S. H. (2009). Tensegrity frameworks: Dynamic analysis review and open problems. *Mechanism and Machine Theory*, *44*, 1–18.

36. Wiegand, R. P. (2004). *An analysis of cooperative coevolutionary algorithms*. Ph.D. dissertation, George Mason University, Fairfax, VA.

37. Wiegand, R. P., Liles, W. C., & De Jong, K. A. (2001). An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Vol. 2611 (pp. 1235–1245).

38. Wroldsen, A., De Oliveira, M., & Skelton, R. (2006). A discussion on control of tensegrity systems. In *2006 45th IEEE Conference on Decision and Control* (pp. 2307–2313). IEEE.

39. Zhang, J. Y., & Ohsaki, M. (2006). Adaptive force density method for form-finding problem of tensegrity structures. *International Journal of Solids and Structures*, *43*, 5658–5673.

40. Zhang, L., Maurin, B., & Motro, R. (2006). Form-finding of nonregular tensegrity systems. *Journal of Structural Engineering*, *132*, 1435–1440.