2014 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS 2014)
September 14-18, 2014, Chicago, IL, USA

Flop and Roll: Learning Robust Goal-Directed Locomotion for a Tensegrity Robot

Atil Iscen[1], Adrian Agogino[2], Vytas SunSpiral[3] and Kagan Tumer[4]

***Abstract*— Tensegrity robots are composed of compression elements (rods) that are connected via a network of tension elements (cables). Tensegrity robots provide many advantages over standard robots, such as compliance, robustness, and flexibility. Moreover, sphere-shaped tensegrity robots can provide non-traditional modes of locomotion, such as rolling. While they have advantageous physical properties, tensegrity robots are hard to control because of their nonlinear dynamics and oscillatory nature. In this paper, we present a robust, distributed, and directional rolling algorithm, "flop and roll". The algorithm uses coevolution and exploits the distributed nature and symmetry of the tensegrity structure.**

We validate this algorithm using the NASA Tensegrity Robotics Toolkit (NTRT) simulator, as well as the highly accurate model of the physical SUPERBall being developped under the NASA Innovative and Advanced Concepts (NIAC) program. Flop and roll improves upon previous approaches in that it provides rolling to a desired location. It is also robust to both unexpected external forces and partial hardware failures. Additionally, it handles variable terrain (hills up to 33% grade). Finally, results are compatible with the hardware since the algorithm relies on realistic sensing and actuation capabilities of the SUPERBall.

I. Introduction

Tensegrity structures are composed of compression elements that are connected with a network of tension elements (Figure 1). Since there are no bending or shear forces, these structures are lightweight yet still capable of handling large external forces. Using the network of tensions, any external force is internally distributed via multiple load paths, which avoids development of high-stress points. Considering the advantage of being lightweight and robust, tensegrity robotics is both an exciting and recently developed research area. In addition to structural advantages, tensegrity robots are deformable and capable of different modes of locomotion, such as crawling, hopping, and rolling.

The high strength-to-weight ratio and internal force distribution makes tensegrity robots especially exciting for space missions. Considering their flexibility, the same tensegrity robot can act as a landing mechanism, and as an exploration device. As a part of a recent NASA Innovative and Advanced Concepts (NIAC) project [1], a recent study illustrates this fact by analyzing possible behavior of an icosahedron tensegrity robot (superball-bot) on a space mission to Saturn's moon, Titan, as a landing and exploration device [20].

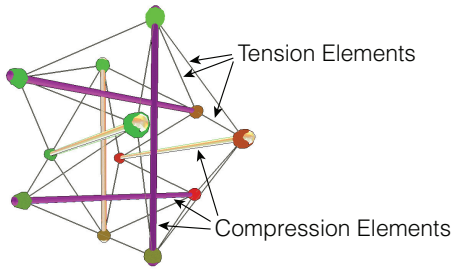

Fig. 1. The tensegrity structure that is composed of 6 rods as compressional elements and 24 muscles as tensional elements.

In this concept, rolling locomotion is a critical research area for tensegrities. Despite the advantages of tensegrities, controlling a tensegrity robot is a fairly difficult problem. Due to the nonlinear interaction of tensional elements, every small change to a tensegrity robot affects the robot's entire structure, which makes it difficult to use established control methods. Finding the right actions for a specific deformation is already a difficult problem; moreover, figuring out the correct deformations that will provide the rolling motion is another challenge.

Previous work shows feasibility of rolling locomotion with open loop signals and evolutionary algorithms, but the approach provides a non-directional rolling behavior that is prone to unexpected external forces or terrain conditions [9]. In this work, we provide a learning based closed loop controls algorithm that uses the contact sensor information of the robot as the feedback. We first divide the problem into simple flops that will optimize rolling behavior. We combine this with policy pooling, a method that we develop to take advantage of the symmetry of the structure. These two methods combined with coevolutionary algorithms provide a learning locomotion that steerable and robust to different environment conditions.

To test the algorithm, we use the NASA Tensegrity Robotics Toolkit (NTRT), a physics simulator based on a bullet physics engine that has been previously validated using an actual tensegrity robot [4]. The model robot used in the simulations uses the same physical parameters (masses, inertias, spring constants, etc.) from the engineering specifications of the prototype SUPERBall currently under development

[1]Atil Iscen is with Electrical Engineering and Computer Science Department, Oregon State University, Corvallis, OR, 97331, USA iscena@onid.oregonstate.edu

[2]Adrian Agogino is with the UC Santa Cruz and NASA Ames Research Center MS 269-3, Moffett Field, CA 94035, USA adrian.k.agogino@nasa.gov

[3]Vytas SunSpiral is with the SGT Inc. and NASA Ames Research Center MS 269-3, Moffett Field, CA 94035, USA vytas.sunspiral@nasa.gov

[4]Kagan Tumer is with Mechanical Engineering Department, Oregon State University, Corvallis, OR, 97331, USA kagan.tumer@oregonstate.edu

at NASA Ames Research Center [3].

The rest of the paper is organized as follows: section II gives background about tensegrity structures and previous research on tensegrity locomotion. Section III plots the overall idea behind flop and roll. Section IV describes policy pooling as a means to use distributed controllers and coevolution while taking advantage of the connection patterns of the tensegrity robot. Section V explains the coevolutionary algorithms and fitness shaping that are used. Section VI gives simulation results for different tests. Section VII presents hardware under development, and section VIII ends the paper with conclusions and future work.

II. Background

The word 'tensegrity' comes from 'tensional integrity' [19]. The members are either pure tension or pure compression elements. The compression elements (rods or struts) are connected via a network of prestressed tensional elements (cables). Using this property, the structure is lightweight, distributes external forces to its members, and none of the parts encounter bending moment. Tensegrity structures were initially explored by Buckminister Fuller [7] and Kenneth Snelson [19] back in the 1960s. The tensegrity concept was initially used in architechture, but it is also being discovered in biological systems, from individual cells to mamalian physiology [8], [13] . 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 act as the primary load path for the body.

Being a fairly new concept, the studies for tensegrity structures started with design and analysis of static structures [2], [18], [11], then continued with form-finding techniques [21], [14]. On the other hand, active control of tensegrity robots brings significant challenges to the traditional control methods due to the compliance, continuous tension network, and nonlinear dynamics. Paul et al. introduced two tensegrity robots that use crawling as the form of locomotion [15]. Recent studies show different approaches to the control of tensegrities, such as Central Pattern Generators (CPGs) and evolutionary algorithms [6], [22], [10]. For tensegrity locomotion, there are multiple research directions, such as biologically-inspired snake robots [22], using vibration, oscillations, [16] and locomotion by rolling [12]. A recent review shows that there are still many open problems in actively controlling tensegrities [23]. In this work, we study icosahedron tensegrity and rolling locomotion.

Icosahedron tensegrity is a simple and a popular tensegrity model that is composed of 6 struts and 24 muscles (actively controlled cables) (Figure 1). It has a spherical shape suitable for rolling locomotion; it is easily deployable and good at absorbing external forces. Due to these properties, it is proposed in a recent NIAC project, "Superball-bot," to perform both Entry, Descent, and Landing (EDL), and surface exploration. The literature contains few actual deformable icosahedron tensegrity robots with actuated muscles. For example, ReCTeR is a lightweight tensegrity robot with additional actuated muscles that are connected to a passive shell (24 passive; 6 active muscles)[5]. Currently, the Spherical Underactuated Planetary Exploration Robot (SUPERBall) is a modular tensegrity robot that is under development at NASA Ames Research Center.

Previous research done with rolling locomotion of icosahedron tensegrities contains tethered robots as well as simulations. Shibata et al. investigates contact conditions and transitions between them to move an icosahedron robot by deforming its body [17]. Koizumi et al. uses a tethered robot with pneumatic actuators and analyzes different surface patterns required to roll [12]. In this paper, we develop a learning based algorithm and analyze the resulting rolling behavior instead of these transitions between different surfaces. In our previous research, we used evolutionary algorithms and sine waves to show to provide open loop rolling locomotion by body deformation [9]. The presented algorithm was an open-loop controller system combined with coevolutionary algorithms. The resulting behavior was not controllable in a desired direction, was highly sensitive to initial conditions, and unable to handle external forces. Although it was proof of concept of rolling for the SUPERBall robot, it cannot be used as a navigation algorithm due to these problems. Despraz et al. provided the first steerable tensegrity locomotion using CPGs [6]. The algorithm provides locomotion by changing the center of mass of the structure by using 12 additional (36 total) muscles that connect an additional payload located in the center of the robot.

In this work, we use the feedback from sensors as states combined with learning. This approach provides the first steerable rolling locomotion towards a desired direction using the 24 muscles of the icosahedron tensegrity robot. The algorithm is distributed and can handle different terrains or external forces as shown in section VI.

To be able to work on control of tensegrities, NASA is developing the NASA Tensegrity Robotics Toolkit (NTRT). NTRT is a simulator based on the Bullet Physics Engine and provides an enhanced and realistic model for muscles. NTRT provides different models of tensegrities and controllers to work with. As part of the development, the simulator was validated with the actual ReCTeR robot and precise motion capture. The results showed that the simulator can correctly simulate tensegrity structures within a small margin of error [4]. In our work, we are interested in the rolling locomotion of spherical tensegrities, concentrated on the icosahedron tensegrity SUPERball. We use NTRT and a tensegrity model compliant with the physical specifications of the SUPERball (length, weight, spring constants, etc.) [3].

III. Flop And Roll

Rolling as a means of locomotion has many advantages. First, it uses gravity to its own advantage as the main driving force. In addition, this locomotion does not have balancing issues. Considering tensegrity structures with a near spherical shape, it is the most natural way to move.

On the other hand, the locomotion algorithm to roll for a tensegrity robot is not trivial. The structure has to deform itself by changing the lengths of the actuated muscles to create any motion. Finding the right configuration to create motion in a direction is already a hard problem, it consists of 24 interconnected elements that all affect each other, resulting in a nonlinear system. Compared to deformation, figuring out continuous and smooth rolling adds inertia as a component to the problem, which creates another level of complexity.

The first step that we take is to divide the problem of rolling into consecutive flops. Considering that the structure is stable on one of its surfaces, we define 'a flop' as deforming the structure and falling to one side only to end up lying on another surface. Doing one flop towards the target will move the robot towards the flop direction and change its orientation. Following the same routine over and over will end up moving the tensegrity robot in the desired direction. Learning to do a single flop in a desired direction is a simpler problem than learning smooth rolling. Unlike the previous approach of learning to roll, what the robot optimizes is a simple 'flop' behavior with the help of feedback from its sensors.

On the other hand, learning a single flop and repeating it does not necessarily provide a smooth rolling locomotion. As an analogy, the difference is similar to the difference between repeating the routine 'one step forward and stop' and smooth walking. During smooth walking, steps taken are optimized for consecutive steps and they differ from taking one single step forward. To avoid such a difference, we chose our fitness function to evaluate overall rolling. During the evolution of policies, the policy that we evaluate makes a single flop, but these single flops will evolve according to their success when they are executed consecutively over 60 seconds. With the same analogy of walking, we evolve the robot to learn how to make a step, but the policy to perform each step is evolved so that it will maximize the walking behavior when executed over and over. The details of the learning algorithm and the state and the fitness function will be explained in section V

The first advantage of the approach that we take is making the control policy simpler (single flop), while learning a more complex behavior (smooth rolling). Additionally, the algorithm can handle external and unexpected forces during rolling motion. This robustness is mainly provided by the fact that the algorithm is composed of smaller pieces to make each flop as opposed to previous research that provides the whole rolling sequence on a given surface [9]. Let's imagine a tensegrity robot that encounters a large external force while rolling using "flop and roll" towards a target point. The external force applied will break the sequence of flops for the robot, and the robot will end up in a random orientation. Since the robot has a spherical and symmetrical nature, it will land on one of its faces. The robot then executes the algorithm in the new orientation, and will deform itself to undertake the first flop that will be followed by a rolling behavior towards the desired target point.

Finally, we test the robustness of the algorithm with an environment where executing a flop is harder due to different terrain properties or external forces. As learned, the robot will try to execute the single flop until it lands on a different surface. The algorithm considers itself at the same state and works on completing the flop until it succeeds. This property allows the algorithm to work on uneven terrains, hills, small obstacles, and unexpected external forces as shown in the results in section VI.

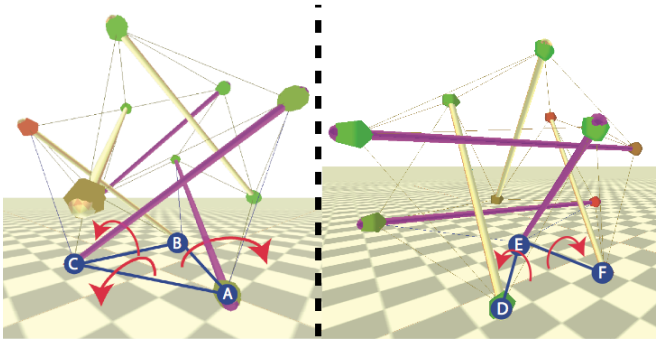

Fig. 2. Possible bases and flops for an icosahedron tensegrity robot. There are two types of possible base triangles when the robot is balanced: an equilateral triangle (left) or an isoceles triangle (right). For the first configuration (left), there are three possible flops (over AB, BC, or CA). For the second configuration, there are two possible flops (over DE and EF). We do not consider the flop over DF since it requires lot more deformation.

IV. Distributed Controls via Pooling

We first discuss how to control the robot using distributed controls while still taking advantage of the symmetrical nature of the structure. The ideal control algorithm for the robot provides rolling motion that is steerable in a desired direction and robust to external forces. This is the main significance of the rolling algorithm that we present in our paper. In terms of input, the algorithm takes sensor information from the robot. Traditionally, robots are equipped with many different sensors, such as cameras and infrared sensors. We will use the minimal information, such as pressure or contact sensors, at the end of the rods and the desired direction.

The robot has 24 muscles that are controlled by 24 independent controllers. Each controller is responsible for selecting the desired length for the muscle that they control. One important point here is that the algorithm does not directly control the lengths of the muscles at a given time. Since the algorithm is based on simple flops for the robot, the lengths provided are the desired lengths for the muscles, so that when reached to the configuration, the structure will flop and start rolling. This particular point makes the algorithm time-independent.

The actual lengths of the muscles at a particular time depends on the speed of the motors and previous configuration, but the algorithm still works with slower motors since reaching the desired configuration will make the tensegrity flop. This fact is also supported with experimental results in section VI. Another advantage of time-independency is that the controllers do not need to have precise synchronization. The robot can tolerate latency in coordination and will still

perform the flop.

At each moment, the robot has a state (contact points and goal direction), the controllers (24 total) and actions to choose (preferred lengths). The goal of the algorithm is to provide rolling behavior that is composed of single flops as explained in section III. The set of policies are defined as

$$\pi_x : (\phi, \upsilon) \rightarrow (l_x) | x \in \{1, 2, .., 24\}$$

where ϕ is the contact points and υ is the desired direction, and l_x is the desired length for the muscle x. Now we will reduce the complexity of the policies to learn by what we call 'pooling', which takes advantage of the symmetrical nature-repetitive pattern of the tensegrity structures.

Let's first analyze the stable configurations for the robot in its default configuration (equal lengths for all the muscles). When the structure will be stable on a surface, it will have 3 points of contact forming the base triangle. Here, we use this advantage for discretization of the state space. Instead of having the orientation of the structure, we use the base triangle that the structure is lying on. For the desired direction, we use the sides of the base triangle, which gives us 3 possible values. We define the new state variables as:

$$s : (\overline{XYZ}, d),$$

where X, Y, and Z represent the edges of the base triangle and d takes values of 0, 1 or 2 (XY, YZ, or ZX), representing the side of the triangle that encapsulates the desired direction to roll.

There are twenty possible surfaces for the icosahedron robot. Since tensegrity robots have repetitive patterns, there are only two types of base triangles: an equilateral triangle where all 3 nodes are connected, or an isosceles triangle where only two of the sides are connected. Figure 2 shows the only two types of possible base configurations. Out of twenty triangle surfaces, eight of them are equilateral (Figure 2 - left) and twelve of them are iscoceles triangles (Figure 2 - right). At any one of these stable situations, the goal of the controllers is to make the robot flop on one of the sides of the triangle. The possibilities are sides AB, BC, and CA on the left of Figure 2 and DE and EF on the right of Figure 2. We do not consider flopping over DF, because not only is DF not connected, it is impossible to perform that flop with a small deformation, since the projection of the center of mass of the structure is much closer to point E, as opposed to the edge DF.

Let's assume that all the controllers have the policy to flop in a given state of $\overline{ABC}, AB$. It can be seen that the structure is symmetrical for all 3 sides (Rotating 120 degrees around a gravitational axis will give the exact same structure). Moreover, if the structure is lying on any other equilateral triangles, we will see the same pattern of connections. This resemblance lets us reuse the knowledge of the policies for state $\overline{ABC}, AB$ for every equilateral base, and $\overline{DEF}, DE$ for every isosceles triangle. The idea is similar transfer learning. We do not directly copy the knowledge, but the policies that perform these flops are reused for flops in all possible orientations.

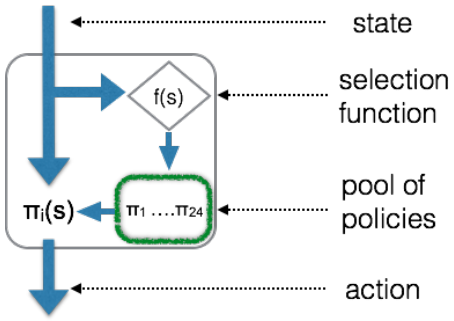

Fig. 3. The decision flow for each agent in Flop and Roll. The pool of policies is same for all the agents, and the decision of policies is done according to the state.

To reuse the knowledge gathered, we use a virtual pool of policies, assuming that all the learned policies $(\pi_1, .., \pi_{24})$ for those particular states ($\overline{ABC}, AB$ and $\overline{DEF}, DE$) are available to all of the controllers. In a new state s' , we only need a function F that returns the policy that each controller select from the pool so that the desired flop will happen.

$$F : (s, i) \rightarrow (j_\pi),$$

where s is the state, i is the unique ID of the controller, and j_π is the ID of the policy that the the controller i should pick from the pool. Figure 3 illustrates the agents on the left side. These agents are used for each of the muscles to perform rolling, and the pool of all the agents is updated at the end of rolling. Please note that the given method is not specific to our robot, F can easily be designed or discovered using the repetitive nature of tensegrity structures. Using pooling, we now reduced the complexity of the problem to learn to 24 policies with only two different states. 24 policies combined with the pooling function F will provide the capability to flop in every possible orientation.

An alternative explanation to this pooling mechanism uses 'roles.' Depending on the new state, each controller selects one of the 24 roles. For example, the controllers of the base muscles in current condition will select the roles of the muscles AB, BC, and CD. The policies in the pool actually represent what to do for that specific role to make a flop. This selection function F , is hand-coded according to the structure, but the policies for the roles are learned using coeovolutionary algorithms, as we explain in section V.

Sharing a pool of policies gives the impression of excessive communication. Yet, there is no active communication during rolling. The policies to use are decided before each episode, and they are not updated during each trial of rolling. Since we use evolutionary algorithms (section V), and delayed fitness assignment, the policies are only updated before each episode. The only time that controllers have to communicate is before starting an experiment, to make sure that their pool is synchronized. Once it is synchronized, the robot can start the experiment and roll without the need of communication for policies. A small amount of communication is used to make sure all the agents figure out and learn the current state. The state is basically a binary value of

a pressure sensor for each node (twelve bits total), and this communication is not sensitive to timing as discussed earlier. It is also possible to derive the state without communication (i.e., using tension sensors, proximity sensors, etc.), but we leave that for future research.

V. Learning to Roll via Flops

Pooling (section IV) reduced the problem to learn to flop with 24 agents in two possible states. On the other hand, the higher-level function to learn is to roll. We will learn the pool of policies, where each policy controls one muscle of the robot for a specific orientation. We use coevolutionary algorithms since they are more suited to the distributed nature of the problem.

Evolutionary algorithms consist of the cycle of forming new members, assigning fitness, and selecting the most fit members. This cycle is called generation and it is repeated until desired behavior is obtained. In coevolution particularly, each one of the 24 policies has its own population. Each of these populations evolve separately, but the policies have to form a team for fitness assignment, because the task needs cooperation of policies to maximize the rolling behavior. At each experiment, one policy from each population is chosen randomly to form a team. This set of policies forms the pool for that experiment.

The problem is episodic, the agents have 60 seconds to test the policies in the pool. At the start of the episode all the agents copy the pool to minimize communication during rolling. At each state s, each agent calls the function $F(s, i)$ to select the policy π_j from the pool. Then agents use the policy until the state changes to a different state, meaning that the structure performed one flop and ended up on another surface.

At the end of each episode, all of the policies forming that team are evaluated according to the performance of the whole robot after 60 seconds. The global fitness function is the distance covered towards the desired goal point. Once again, fitness is not related to the flops, it is only related to the overall rolling. Although it is possible to evaluate all these policies with the same fitness function (distance traveled), a better fitness assignment can evaluate each policy according to their contribution to the overall performance. The method of providing different fitness assignments for each agent is called fitness shaping. In this work, we use historical average fitness-shaping that is previously used with tensegrity robots with success [9]. In the historical average, each policy in the pool shapes the global fitness according to its previous experience by taking the average of all the teams that it has been previously tested with.

For each generation, we test the individuals using their performances with different teams. To form teams we use random sampling. 50 random teams are formed and the members are assigned fitness according to these experiments. After 50 experiments, each population eliminates half of its members to keep the most fit ones. These selected members form new members by mutation.

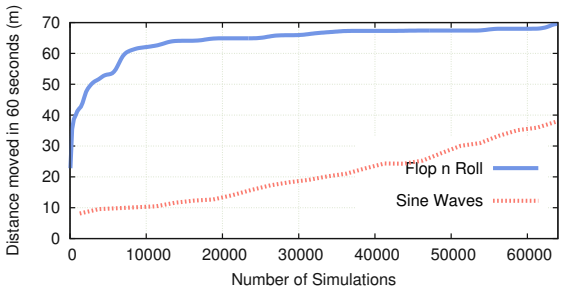

Fig. 4. The performance of the best policies over time during the learning process. Flop and roll is compared to the non-directional sine wave approach. Flop and roll learns directional rolling behavior in a short amount of time.

VI. Results

In this section, we present the results of the experiments that we conduct using NTRT. As stated in section II, NTRT has been previously validated with a physical tensegrity robot and has been shown to produce less than 1% error for a passive tensegrity and for semi-static controls of the active tensegrity robot used. In our experiments, each strut is 1.5m in length, 3 kg in weight, and the same as in the specifications of the SUPERball design. We used 24 active muscles controlled by 24 controllers, where the muscles have a rate of change in length of 0.3 m/s, and the elasticity coefficient for the muscles is 3kN/m.

The first experiment is to learn to roll using the algorithm 'flop and roll,' as described in previous sections. We compare it to the sine wave rolling algorithm that was previously presented in [9] and discussed in section II. First, flop and roll is a method for learning to roll towards a goal; on the other hand, the sine wave approach that we compare it to is a less-capable algorithm that provides uncontrollable rolling motion in one direction. To be fair on the sine wave approach, we compare the distance traveled in any direction by the sine wave approach vs. the distance traveled towards the goal using flop and roll.

Figure 4 shows that the flop and roll algorithm actually takes less time to learn and perform better. The flop and roll algorithm takes advantage of the reuse of knowledge and has a learning curve with a jumpstart. Flop and roll learns a basic rolling behavior in a few generations, and it reaches rolling with a speed of 60 meters per minute towards a desired direction.

In this figure, the result of the previous sine wave approach is slightly different than the original result that was published in 2013. Since the publication of the sine wave algorithm, the realism of the simulation environment is increased using improvements that are made on NTRT. Moreover, in this work, the specifications of the tensegrity model is closer to the prototype that is currently under development [3]. In this work, we are testing the two algorithms in a more realistic environment with a robot that is harder to control.

Second, we analyze the learned rolling behavior. We take the best policy and test it with 3 consecutive targets. Figure 5 shows the targets and the path of the center of the robot.

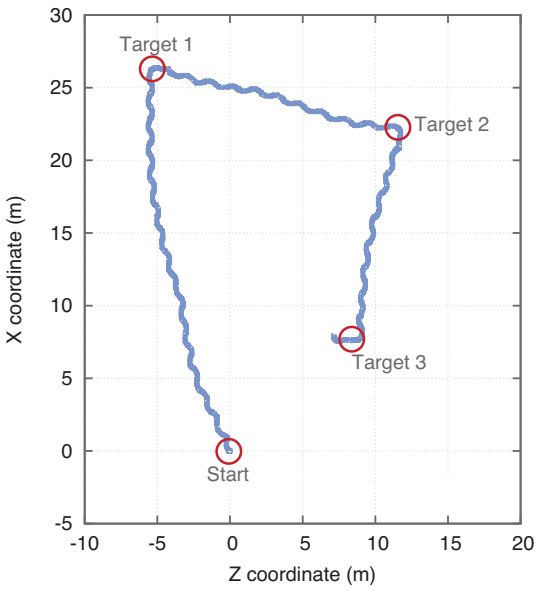

Fig. 5. The 2D path followed by the robot with the learned policy tested with 3 consecutive targets. Flop and roll provides successful rolling and changing directions.

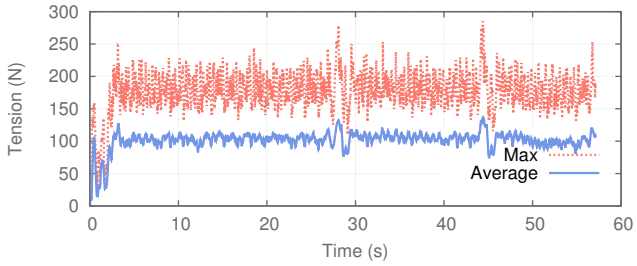

Fig. 6. The average and maximum tensions of the muscles during the experiment with 3 targets. The tensions stay within reasonable limits and the peaks are reached during change of direction while switching to different targets.

The robot successfully navigates to all of the targets. The zig-zag behavior that is observed is due to the fact that the robot does not have a perfect spherical shape, it falls on two sides in order to roll.

As part of the same experiment, we analyze the lengths and tensions of the muscles during this navigation. These results are a keypoint for the applicability of the algorithm, because we have to make sure that resulting behavior does not require an unrealistic amount of tensions or deformations on the muscles. Figure 6 shows that the stationary robot (first 2 seconds) has 50N of average tension and 100 N maximum tension. During the rolling motion (after 2 seconds), the maximum tension is around 180N, with the average around 100 N. The graphs shows two peaks around 30 and 45 seconds. These peaks are due to righ-turning when the robot reaches target 1 and 2. Even in those cases, the tensions stay below 300N. These numbers gives an idea about the applicability of the algorithm to the physical robot.

If we look at the lengths of the muscles, Figure 7 shows

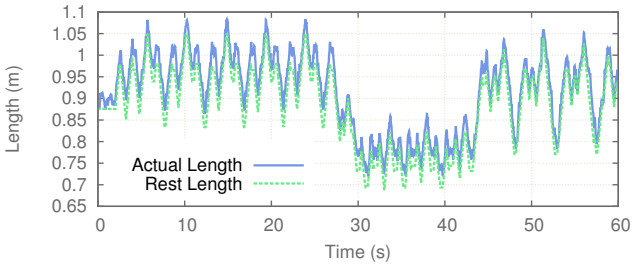

Fig. 7. The rest length and real length of one of the muscles during the experiment with 3 targets. The patterns emerge between the 0-30's, the 30-45's, and the 45-60's due to the rolling to different directions for 3 different targets. Rolling to different directions give different roles to this particular muscle.

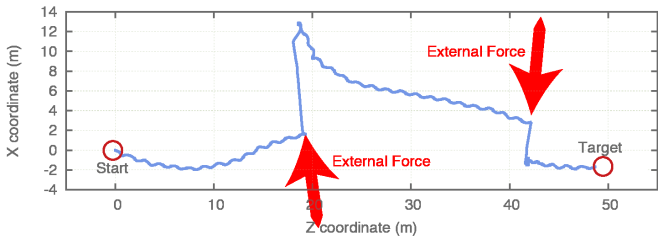

Fig. 8. The 2D path followed by the robot with two intense unexpected external forces. The robot is basically smashed with an impulse that causes the robot to roll outside its path. After stabilization, the robot starts rolling to its target.

the pattern used for one of the muscles. Since rolling is a repeating behavior, the same pattern repeats itself until the robot changes direction. During this first phase, the rest length of that muscle oscillates between 0.8m and 1.05m. The actual length of the muscles (since they are elastic) is slightly higher then the rest length, which provides the tension analyzed in the previous graph. When the robot starts moving towards target 2 (around the 30's), the direction of rolling is different and the same muscle has a different role in the rolling locomotion. It is shorter and oscillates around the 0.7 - 0.85m range. The third phase is completely different and it can be seen that the pattern is different once again.

To test how the learned behavior handles an unexpected external force, Figure 8 shows the path of the robot when it is pushed sideways (twice) during the rolling motion. The robot smashed by the impulse rolls sideways but stabilizes and starts rolling towards its target again.

Next, we test the ability of the algorithm to overcome a terrain with different types of hills. After learning using maps with randomly-generated hills, we take the best policy and analyze its performance. The robot can continuously climb an inclined uniform terrain with a 20% grade (not pictured). In terms of nonuniform terrain with small random hills, Figure 9 shows a 3D graph of the path of the robot with learned behavior. The robot climbs over small hills with grades of up to 33%. As a comparison, these hills have similar grades to the steepest streets of San Francisco.

Our next tests are about the applicability and robustness of the learned behavior. We first test the algorithm with

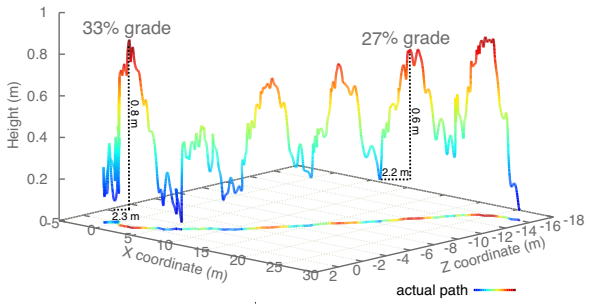

Fig. 9. Flop and roll trained and tested on a terrain with steep hills. The robot climbs hills of up to a 33% grade.

slower motor speeds. We train the robot using the default configuration of simple motors models with a constant speed of $0.2m/s$ independent of the load. We take the best policy and test it with motors as slow as $0.02m/s$. Figure 10 shows that the robot can still move in the desired direction. The learned behavior does not necessarily require fast motors. The slower motors mean slower rolling, but we obtained a distributed rolling algorithm independent of motor speeds.

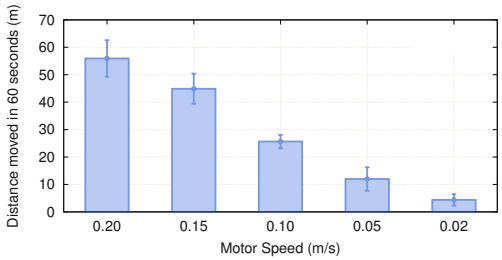

Fig. 10. Flop and roll tested with slower motor speeds. Even if it is trained with a robot that has motors with 0.2 m/s speed, it can still roll with slower motors.

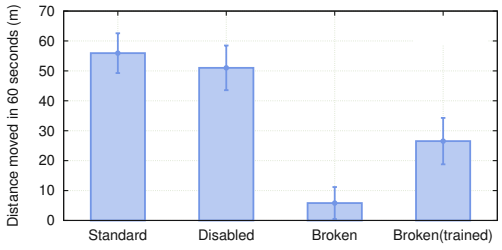

Fig. 11. Flop and roll tested with non-functioning controllers and broken muscles. With one controller not functioning, the robot still rolls close to its original performance. When one of the muscles breaks completely (does not exist anymore) and symmetry is broken, the robot can perform rolling when it is trained for breakage of one random muscle.

The last set of experiments test the algorithm against hardware failures. We compare standard conditions with broken controls and broken muscle. In the first scenario, we disable the controller, meaning that one randomly chosen muscle becomes stuck with the initial length. In the second scenario, we break one randomly chosen muscle assuming that the cable does not exist anymore. Figure 11 shows that if one of the controllers stops working, the structure can still roll successfully. Imagine a legged robot where the controller of one of the joints stops working. The algorithm for running will be highly affected by this failure. In our case, the distance covered by the robot in 60 seconds decreases from 55 to 50m. For the case of the broken muscle, one random part of the structure will be affected by destroyed cable causing the robot to be deformed and to lose symmetry. In this case, the learned algorithm is highly affected. On the other hand, the fourth column shows that if we train the algorithm for such cases, we can obtain a rolling behavior that is prone to broken muscles. Please note that during training the broken muscle is selected randomly so that the algorithm can be prone to failure for any of the 24 muscles.

VII. Hardware Implementation

Recent research includes examples of different tensegrity robots as discussed in section II. The flop and roll algorithm presented in this paper aims the tensegrity robots with a spherical shape, untethered, and which actuate by changing the lengths of muscles. The current robot with these capabilities is the ReCTeR (Figure 12 - left). ReCTeR has 6 motors, multiple sensors with a strut length of 1m and a total mass of 1.1kg including batteries. On the other hand, ReCTeR is a lightweight version with a passive outer shell and is actuated using 6 additional active muscles that connect with the outer shell. ReCTeR was used in different research studies and also to validate the NTRT simulator where results showed that the simulator matched ReCTeR with a small error.

The main target for flop and roll is the SUPERBall that is designed to include the lessons learned with ReCTeR. A new model SUPERball is currently under development at NASA Ames Research Center. Figure 12 , on the right side, shows one strut from the current design of the SUPERball, a modular tensegrity robot that can actuate the outer shell. The design specifications of both robots is given in table I. The final design of the SUPERball will be capable of being used in entry, descent, and landing and with surface mobility for space projects. The specifications of the robot that are used in this paper (length, weight) match the current development of the SUPERball that is explained in [3].

VIII. Conclusion and Future Work

We presented the flop and roll algorithm as the first distributed directional rolling algorithm for tensegrity robots. Flop and roll takes advantage of the pattern of the tensegrity robot and performs knowledge reuse by pooling. Knowledge reuse simplifies the policies dramatically, and the policies are optimized using coevolutionary algorithms and fitness-shaping. During coevolutionary algorithms, the policies to make the robot flop are optimized according to their overall rolling performance. Although the policies' job is to provide repeated flops, their fitness function provides the smooth and robust rolling behavior.

We used the NTRT simulator to train and test our algorithm. We analyzed the learned behavior and looked at its performance under a flat surface, with consecutive targets, and with terrain that included hills. Our results show

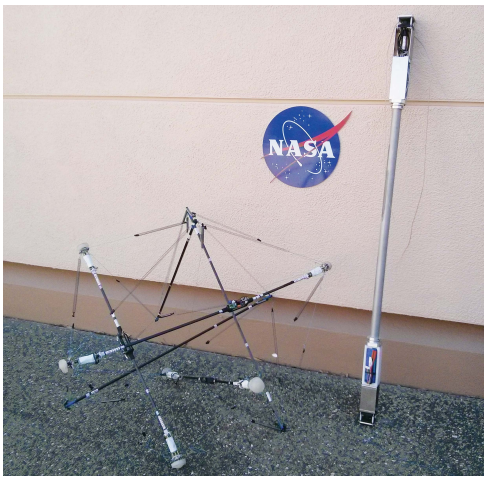

Fig. 12. Tensegrity robots used in this work. ReCTeR (left) is an untethered tensegrity icosahedron robot. The strut (right) belongs to the SUPERball, a modular tensegrity robot that is under development at NASA Ames Research Center.

TABLE I
SUPERBALL DESIGN REQUIREMENTS

.

	l_{strut}	Δl	$k_{passive}$	Ctrl. freq.	$\max \tau$
ReCTeR	1m	0.3m/s	28.4N/m	40Hz	0.03Nm
SUPERball	1.5m	0.26m/s	500N/m	100Hz	3Nm

successful rolling under different conditions with reasonable muscle lengths and tension values. Combined with the robust and distributed nature of tensegrities, we show that the robot can handle disabled and broken muscles. Moreover, since tensegrity robots are known for handling external forces very well, one can throw the robot in any direction and the robot can survive the impact and can still follow its target. A previous study shows that NTRT gives similar results simulating semi static behavior of tensegrity robots [4]. The algorithm that we present in this paper contains minimal amount of assumptions about the hardware (slower position control and contact sensors). Therefore similar results should be observed upon application to the hardware.

Rolling tensegrity locomotion provides many advantages over other forms of locomotion. By the tensegrity principle, the locomotion is distributed and more robust to failures. The spherical shape eliminates the the balancing problem that legged locomotion encounter. The simulated robot rolls on its 12 endcaps, while using a symmetrical gait. Additional advantages of such a gait are being less sensitive to different motors speeds and terrain conditions. In terms of energy consumption, the rolling locomotion uses mass and gravity to its advantage. Although we did not show the energy consumption of the robot in this paper, investigation and optimization of energy efficiency is left as a future work.

The next step for this research is the application of flop and roll with the superballbot that is under development. Flop and roll is a relatively easy algorithm to apply to hardware. It uses a small amount of information (contact sensors), is suitable for different configurations, can handle different terrain conditions and external forces, and its distributed nature can handle failures.

ACKNOWLEDGMENT

This research was partially supported by the NASA Innovative Advanced Concepts (NIAC) Program.

REFERENCES

[1] A. Agogino, V. SunSpiral, and D. Atkinson. Super Ball Bot - structures for planetary landing and exploration. *NASA Innovative Advanced Concepts (NIAC) Program, Final Report*, 2013.

[2] N. Bel Hadj Ali, L. Rhode-Barbarigos, A. Pascual Albi, and I. Smith. Design optimization and dynamic analysis of a tensegrity-based footbridge. *Engineering Structures*, 32(11):3650–3659, 2010.

[3] J. Bruce, K. Caluwaerts, A. Iscen, and V. SunSpiral. Design and evolution of a modular tensegrity robot platform. In *IEEE International Conference on Robotics and Automation*, 2014.

[4] K. Caluwaerts, J. Despraz, A. Iscen, A. Sabelhaus, J. Bruce, B. Schrauwen, and V. SunSpiral. Design and control of compliant tensegrity robots through simulation and hardware validation. *Journal of the Royal Society Interface*, 2014.

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

[6] J. Despraz. Super ball bot - structures for planetary landing and exploration. Master's thesis, Biorobotics Laboratory, EPFL, 2013.

[7] B. Fuller. Tensegrity. *Portfolio and Art News Annual*, 4:112–127, 1961.

[8] D. E. Ingber. Cellular tensegrity: defining new rules of biologic design that govern cytoskeleton. *Journal of Cell Science*, 104:613–627, 1993.

[9] A. Iscen, A. Agogino, V. SunSpiral, and K. Tumer. Controlling tensegrity robots through evolution. In *GECCO*, 2013.

[10] A. Iscen, A. Agogino, V. SunSpiral, and K. Tumer. Learning to control complex tensegrity robots. In *AAMAS*, 2013.

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

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

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

[14] M. Masic and et al. Algebraic tensegrity form-finding. *International Journal of Solids and Structures*, 42:4833–4858, 2005.

[15] C. Paul, F. J. V. Cuevas, and H. Lipson. Design and control of tensegrity robots for locomotion. *IEEE Transactions on Robotics*, 22(5):944–957, 2006.

[16] J. A. Rieffel, F. J. Valero-Cuevas, and H. Lipson. Morphological communication: exploiting coupled dynamics in a complex mechanical structure to achieve locomotion. *J R Soc Interface*, 7:613–621, 2009.

[17] M. Shibata, F. Saijyo, and S. Hirai. Crawling by body deformation of tensegrity structure robots. In *ICRA*, pages 4375–4380. IEEE, 2009.

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

[19] K. Snelson. Continuous tension, discontinuous compression structures. united states patent 3169611, Feburary 1965.

[20] V. SunSpiral, G. Gorospe, J. Bruce, A. Iscen, G. Korbel, S. Milam, A. Agogino, and D. Atkinson. Tensegrity based probes for planetary exploration: Entry, descent and landing (EDL) and surface mobility analysis. *To Appear in International Journal of Planetary Probes*, July 2013.

[21] Tibert and et al. Review of form-finding methods for tensegrity structures. *International Journal of Space Structures*, 18:209–223, 2003.

[22] B. R. Tietz, R. W. Carnahan, R. J. Bachmann, R. D. Quinn, and V. SunSpiral. Tetraspine: Robust terrain handling on a tensegrity robot using central pattern generators. In *AIM*, pages 261–267, 2013.

[23] J. M. M. Tur and S. H. Juan. Tensegrity frameworks: Dynamic analysis review and open problems. *Mechanism and Machine Theory*, 44:1–18, 2009.