CENTRAL PATTERN GENERATOR GAIT EVOLUTION OF A ROBOTIC MODULAR TETRAHEDRAL TENSEGRITY

A Thesis

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Science

with a

Major in Mechanical Engineering

in the

College of Graduate Studies

University of Idaho

by

George P. Korbel

December 2013

Major Professor: Eric Wolbrecht, Ph.D.

AUTHORIZATION TO SUBMIT THESIS

This thesis of George P. Korbel, submitted for the degree of Master of Science and titled "Central Pattern Generator Gait Evolution of a Robotic Modular Tetrahedral Tensegrity," has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor _____Date_____
Eric Wolbrecht

Committee Members _____Date_____
Matthew Riley

_____Date_____
Terence Soule

Department Administrator _____Date_____
John Crepeau

Discipline's College Dean _____Date_____
Larry Stauffer

Final Approval and Acceptance by the College of Graduate Studies

_____Date_____
Jie Chen

ABSTRACT

Presented in this thesis is a method for determining locomotion gaits for a modular tetrahedral tensegrity robot in simulation. The biologically inspired tensegrity-based robot is comprised of multiple rigid tetrahedrons connected with a network of adjustable tension members. Locomotion gaits are produced using oscillatory signals from Central Pattern Generators (CPGs) that are optimized by an Evolutionary Algorithm (EA). After initial generation of an EA population of random gaits, as defined by parameters of the CPG, the EA operates on the population in an attempt to find an optimum gait, as defined by the performance, or fitness, of the evolved locomotion. The results of the EA optimization show that the fitness of the populations were increased by an average of 32 percent over 10 trials, and 51 percent in the best trial. The approach could be applied to a more complex tetrahedral tensegrity or other tensegrity structures. Future work will implement this approach on a robotic prototype.

ACKNOWLEDGEMENTS

I would like to thank the Idaho Space Grant Consortium for funding my research and for their support during my education. I would like to thank Dr. Eric Wolbrecht for his guidance throughout my graduate studies. I would also like to thank Dr. Terry Soule for his support and Dr. Matt Riley for serving on my master's committee.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1. Introduction

Tensegrity structures are composed of axially loaded compression elements suspended within a network of tension elements. The term tensegrity comes from the combination of the words tension and integrity; it's the idea that the stability, or integrity, of these structures is due to the tension network. In such a structure, applied loads are distributed as only tension and compression. As typically materials are stronger in pure tension and compression, tensegrity structures, including tensegrity-based robots, can exhibit a high strength to weight ratio.

The field of tensegrity robotics has emerged over the last decade and has potential for large application, including in space technology and other fields. Tensegrity is a class of structure that first appeared just over half a century ago [1]. Since its introduction tensegrity has been applied to multiple areas including engineering, architecture, medicine, biology, and art [2].

Tensegrity robots have the ability to change shape by changing their equilibrium [3]. This can be achieved by varying the lengths of either the compression or tension elements. Such changes can be used for a different static function or for locomotion.

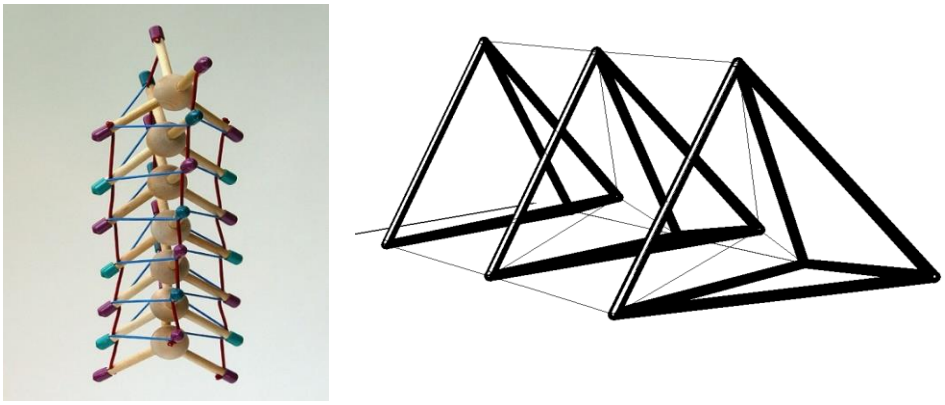


Figure 1. Tetrahedral vertebrae mast structures (Left: Flemons', Right: Modified).

The Intelligent Robotics Group at NASA Ames Research Center is currently researching tensegrity structures for robotic applications. The unique characteristics of tensegrity could enable these robots to perform in ways not currently possible with traditional robotic design; deformable,

light weight and compactible design are excellent characteristics for space transportation and exploration in difficult and uncertain terrain.

The specific tensegrity structure of interest is based on Tom Flemons' tetrahedral vertebrae mast [4] as seen on the left side of Figure 1. In Flemons' original design, compression members extend out from a central node. The external endpoints of the compression members are connection points for tension members. In Flemons' model the points of contact change as the lengths of the tension members are changed, which complicates robotic control of the structure. The work presented here uses a modified version of Flemons' tetrahedral vertebrae mast (shown in the right side of Figure 1), refered to herein as a five-tetrahedron tensegrity robot. This modified structure has predictable points of contact which simplifies the control of the structure. A robotic prototype of the modified tetrahedral spine, as pictured in Figure 2, was constructed at the University of Idaho and continues to be investigated by researchers at NASA Ames.

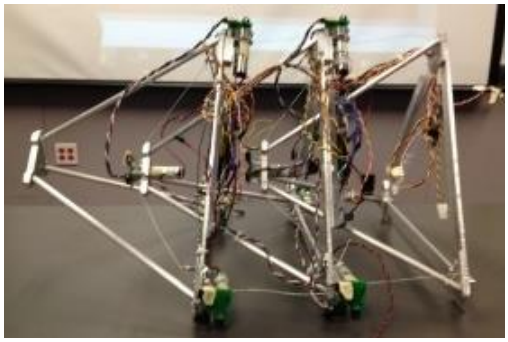


Figure 2. Tetrahedral tensegrity robot prototype currently at NASA Ames.

Features of tensegrity robots that are advantageous in one respect are sometimes disadvantageous in another. Structural complexity is one such trait. As the number of compression members and connecting tension members is increased, so are the degrees-of-freedom of the structure. As such, it becomes progressively difficult to determine kinematic and kinetic reltionships within the structure. Furthermore, tensegrity structures are inherently nonlinear, adding to the

difficulty of their analysis [5]. These factors make prescribing locomotion gaits for robotic tensegrity control a significant challenge.

Because of the complexity of tensegrity structures, closed form solutions of equations of motion may be difficult to determine. Thus, dynamic simulation of tensegrity structures is desired. Researchers at NASA Ames are developing a robotic tensegrity modeling software called NASA tensegrity robotics toolkit, or NTRT, that is based in the open source game physics engine Bullet. This software is capable of simulating soft body dynamics needed to model deformable tensegrity robots. This software models forces on the compression members as point forces. However, modeling the forces on compression members using distributed inertia is a more realistic method.

The goal for the work presented in this thesis is to develop an alternative method for dynamic simulation, gait generation and optimization for tensegrity-based robots, specifically for the UI's five-tetrahedron tensegrity robot. SimMechanics, a toolbox for The Mathworks Simulink simulation environment, is used to model and simulate the dynamics and ground contact forces of the five-tetrahedron tensegrity robot. SimMechanics provides pre-built functions for modeling forces on rigid and soft components in a three-dimensional environment. SimMechanics also gives the user the ability to model solids with distributed inertia, which is an accurate representation of the rigid compression members in the robot.

This tensegrity structure is biologically inspired, which suggests the use of biologically inspired methods, such as Central Pattern Generators (CPGs), for determining the locomotion gaits. In biological systems, CPGs function as non-linear neural connections and oscillators to create locomotion gaits. In robotics CPGs are used to create oscillatory signals that can be reference trajectories for joints and/or actuators of a robot. Multiple interacting CPGs can be applied to a combination of joints to create a locomotion gait [6].

There are a variety of CPG methods used to control different types of robots, including legged robots, robotic arms, tensegrity robots, and crawling and snake-like robots. A large amount of

research is in the field of gait production and control of legged robots. An overview of previous work on CPG controlled legged robots is provided in [7]. CPGs are versatile such that they can be applied to a variety of different robotic movements. Generally each degree-of-freedom of a robot is controlled by one CPG oscillator [8]. This idea is not only applied to robots with the goal of locomotion but is also applied to robotic arms as in [9], where CPG oscillators are used to learn the dynamics of a robotic arm in order to more effectively control it. CPGs have also been applied to tensegrity structures because the rhythmic nature of CPGs exploits the natural resonance of deformable tensegrity structures [10]. CPGs are used to create gaits for snake-like robots that move in a serpentine motion. One such robot, named AmphiBot I, was created by Crespi [11]. This robot takes its inspiration from a lamprey eel or sea-snake and is able to propel itself forward using a lateral undulatory mode of actuation where a repeating wave travels through it. A similar CPG controlled snake-like robot was created by Inoue [12].

For the work presented in this thesis, the Matsuoka neural oscillator model with a mutual-inhibition network, as presented in [13], was selected as the CPG for the five-tetrahedron tensegrity robot. The Matsuoka CPG uses a set of coupled differential equations to model the oscillating network of neurons. The Matsuoka CPG has been applied to a variety of types of robots including bipedal robots [14] and crawling and snake-like robots [6]. Depending on the configuration of the Matsuoka CPG numerous parameters define the output. Hand tuning such parameters to produce a desired locomotion gait is difficult and unlikely to produce an optimal result. Thus, a systematic method is needed to determine the parameters of the CPG that create the desired locomotion gait.

An Evolutionary Algorithm (EA) can be used to improve the performance of CPG walking gaits [15]. An EA takes a population consisting of a number of individual solutions, and attempts to improve the performance of that population. EAs search for an optimum by carrying traits of the strongest performing individuals through to the next generation while traits that create weak individuals are discarded. The performance of these individuals is called their fitness, and is defined

according to the application. The goal of an EA can be to minimize or maximize the fitness of the population depending how the fitness is evaluated [16]. Common termination criterion for EAs are elapsed simulation time, number of iterations, or a measure of the population diversity.

For the work presented in this thesis each set of parameters defining a unique instance of the Matsuoka CPG is considered a member of the EA population. The fitness of each member is evaluated via dynamic simulation of the CPG driven five-tetrahedron tensegrity; generally, fitness of a member depends on the quality of the locomotion gait produced in simulation and larger fitness values represent better performing individuals. For the work presented in this thesis the fitness primarily depends on the forward distance the robot travels during dynamic simulation. The EA operates in steady state, meaning two parents are selected from the population, operated on, and the resulting offspring replace current members of the population. Selection of parents and replacement of offspring is accomplished using a tournament method, where multiple candidate parents are randomly selected from the population and the candidate with the best fitness value becomes a parent. After parents are selected they are varied by uniform cross-over and mutation. In the uniform cross-over step each of the corresponding parameters of the two parents have a probability of being swapped. In the mutation step each of the parameters of the two parents have a probability of a small change in value.

The implemented EA is terminated if the diversity of the population falls below a threshold or if a maximum number of iterations is reached. Similar methods have been used for real time control of bipedal robots using CPGs [17]. However, the CPG model used in the work presented in this thesis does not incorporate sensory feedback, rather a generated CPG gait defines open-loop force trajectories for certain tension members of the five-tetrahedron tensegrity robot. The following sections discuss the methods used to generate and evolve gaits, the results of those methods, and conclusions drawn from process.

Chapter 2. Methods

The specific structure that is dynamically modeled is a five-tetrahedron tensegrity robot as seen in Figure 3. The compression members that make up the sides of the five tetrahedron are modeled as rigid rods. All tension members (cables) are modeled as springs with damping as described in the tension member model section. The lower, outer tension members are actuated by applying the CPG output signal as a superimposed force in the tension members. Each pair of the actuated tension members connecting adjacent tetrahedron are actuated using the same CPG signal. The following four sections go over the methods used to evolve CPG locomotion gaits for this robot in a simulation environment. The first section details the dynamic modeling of the robot and ground interaction forces. The second section describes the implementation of the CPG. The third section covers the EA optimization procedure. The fourth section describes how these models and algorithms were implemented in the Matlab, Simulink, and SimMechanics software environments.

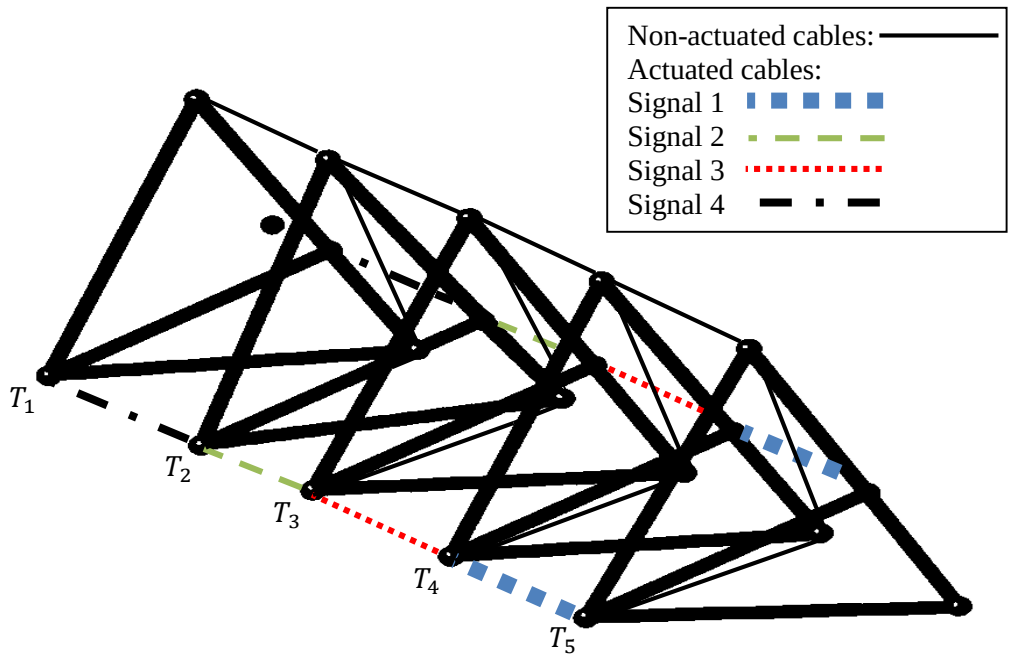


Figure 3. Five-tetrahedron tensegrity robot in SimMechanics with tetrahedra (T_1-T_5).

2.1 Dynamic Model

In the dynamic model, the actuated cables (as shown in Figure 3) are actuated by a simulated CPG as described in section 2.2. Each pair of actuated cables connecting two adjacent tetrahedra is actuated by one signal of the CPG. The rigid structure of the tetrahedra is modeled with 6 inch long struts each with a diameter of 0.125 inches. A single modeled tetrahedron was used to test and verify the contact and fiction models in the simulation environment. The three sections below describe the models used in the dynamic simulation.

2.1.1 Normal Force Model

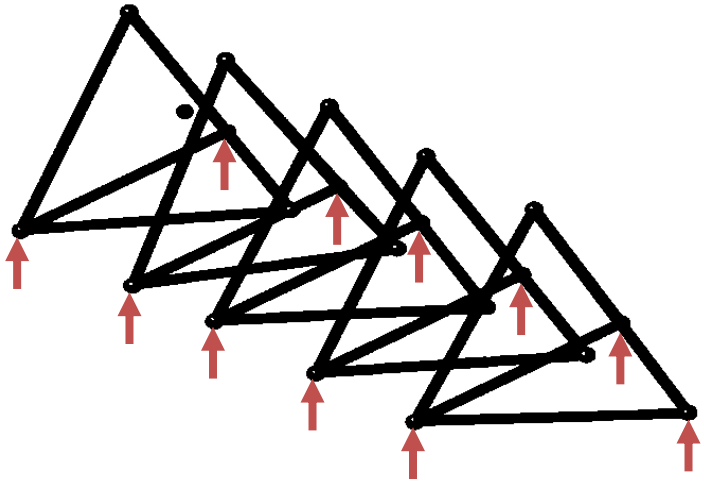


Figure 4. Nodes where ground and friction forces are modeled (represented by arrows).

The normal force, or ground force, is modeled as a nonlinear spring and damper, as presented in [18]. When the position of a node is above the ground surface, defined as zero on the vertical axis, the normal force F_N is zero. When the position of a node is at or below the ground surface the equation for normal force, F_N, due to ground contact is

$$F_N = -k_N \delta^n - \lambda \delta^n \dot{\delta}, \qquad \text{Eq. 1}$$

where k_N is the spring constant, δ is the deformation (defined as positive when below the ground surface), λ is the damping constant and n is a constant that accounts for the geometry of the colliding

objects (n = 1 for impacting planes and n = 1.5 for impacting spheres) [18]. For this work we assume n = 1. A method for defining λ and n using the coefficient of restitution (e) and the impact velocity (v_i) is given in [18]. If v_i is small we can use

$$e = 1 - \alpha v_i \text{ and} \qquad \text{Eq. 2}$$

$$\lambda = \frac{3}{2}\alpha k_N,$$

where α is a velocity correction constant. Simulating the ground model with the single tetrahedron positioned just above the ground (zero on the vertical axis) showed that the impact velocity was approximately 0.12 inches per second. A value of 0.9 was used for the coefficient of restitution in order to allow for elastic collisions. A spring constant of 25 pounds per inch was used as an initial guess of the stiffness of a fairly rigid carpet material that could be used for future testing of a prototype. These values gave an estimate of λ = 31.1 pound second per inch using equation 2 above. These values were implemented into the model as initial estimates then tuned by visually comparing dynamic simulation results to a full scale physical model of a single tetrahedron on a representative ground surface. The values listed in Table 1 define the normal forces between the five-tetrahedron robot and the ground during dynamic simulation.

Table 1. Normal Force Model Parameters

Coefficient of restitution (e)	0.9
Velocity correction coefficient (α)	0.72 $\frac{s}{in}$
Spring constant (k_N)	25 $\frac{lb}{in}$
Damping constant (λ)	27 $\frac{lb \cdot s}{in}$

In an effort to reduce the simulation time, ground forces are not modeled on the top and front nodes of each tetrahedron, as these points will never make contact with the ground during expected locomotion behavior. However, ground forces are modeled on the front most node on the lead tetrahedron, as it is allowed to contact the ground during locomotion (see Figure 4). Figure 5 shows

the simulated contact model behavior of a single node of the tetrahedron as it settles from an initial position just above the ground (above zero on the vertical axis). Because the parameters defining this model are determined using a relatively low impact velocity, v_i, the model is valid when the nodes of the tetrahedron do not contact the ground with a large velocity. During expected locomotion the impact velocities are low and this model is valid.

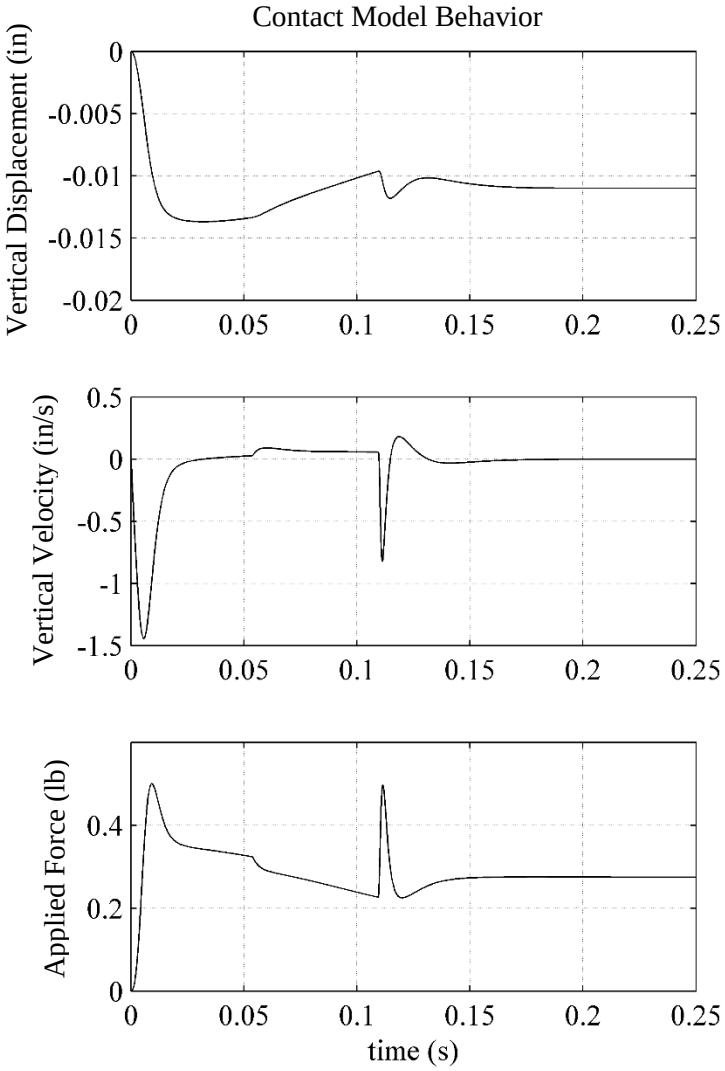


Figure 5. Displacement and normal force during ground contact with a single, settling tetrahedron node.

2.1.2 Friction Force Model

As with an object moving over land, the five-tetrahedron robot simulated in this research uses interaction with the ground to achieve forward mobility. Therefore, it is critically important to have an accurate model of friction. For this application, a reset integrator friction model was used. This model is a good approximation of the more accurate bristle model but is more computationally efficient [19]. The bristle model is designed using the concept that friction between surfaces is caused by small structures, or bristles, that flex as lateral forces are applied and break free at a threshold force or displacement. This allows for damped deformation to occur during static friction, or stiction. The reset integrator model mimics this behavior by resetting the initial integration point to zero when the model breaks free of stiction. The model defines stiction velocity according to

$$\frac{dx}{dt} = \begin{cases} 0 & if (v > 0 \; and \; x \geq p_o) \; or \; (v < 0 \; and \; x \leq p_o) \\ v & otherwise \end{cases}, \quad \text{Eq. 3}$$

where v is the velocity of the node, x is the relative distance the node has moved while in stiction, and p_o is the stiction range, or the range the node is allowed to operate in static friction. Stiction is activated using the equation

$$a(x) = \begin{cases} a_c & if \; |x| < p_o \\ 0 & otherwise \end{cases}, \quad \text{Eq. 4}$$

where a_c is a constant. Eq. 4 reduces the force due to stiction when the node breaks free by setting $a(x)$ to zero. These equations combine to define the frictional force, F_F, as

$$F_F = \sigma_o v\big(1 + a(x)\big)x + \sigma_1 \frac{dx}{dt}. \quad \text{Eq. 5}$$

The value of the stiction range p_o, damping coefficient during stiction σ_1, and stiction coefficients a and σ_o were tuned ad hoc via simulation and using a physical model of a single tetrahedron to resemble friction on a rough surface. The values used in this model are listed in the Table 2. Figure 6 shows the behavior of the reset integrator friction model when one tetrahedron is simulated with initial velocity. This figure demonstrates that the node breaks free of, and re-enters static friction

before coming to a rest. This behavior is desired because intuitively the five-tetrahedron tensegrity robot uses static friction as a tool to generate locomotion.

Table 2. Friction Parameters

Stiction range (p_o)	0.004 in
Damping under stiction (σ_1)	0.4 $\frac{lb \cdot s}{in}$
Stiction switching coefficient (a_c)	0.7
Proportional stiction coefficient (σ_o)	0.3 $\frac{lb \cdot s}{in^2}$

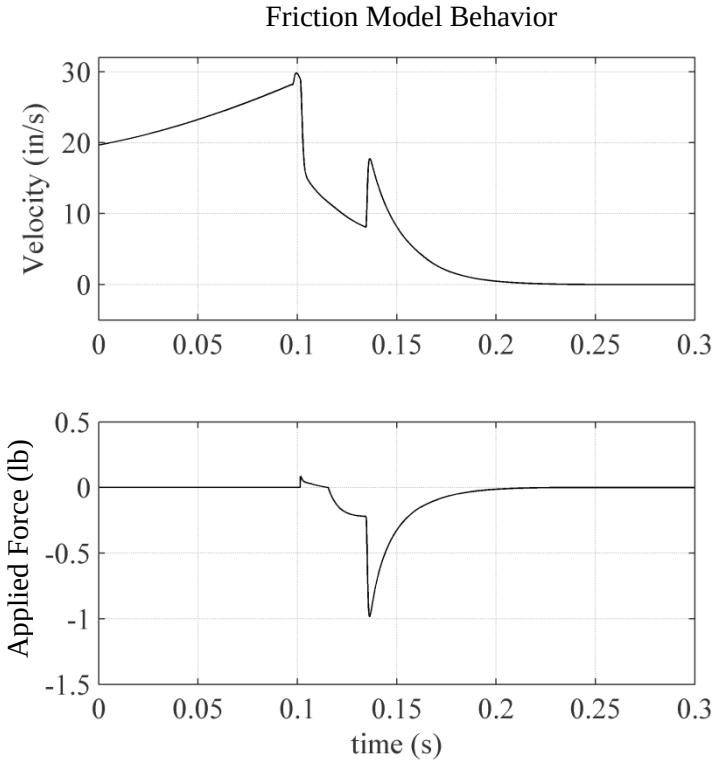


Figure 6. Friction force and velocity of a single node during simulated sliding on a rough surface.

2.1.3 Tension Member Model

The tension members are modeled as a spring force, F_s, with damping included as

$$F_s = k_s(x_o - x) + c(\dot{x}), \qquad \text{Eq. 6}$$

where k_s is the spring constant, x_o is the free length of the spring, c is the damping constant, $\dot{x}$ is relative velocity, and x is the relative displacement. The outer cables that connect the base triangles of one tetrahedron to those of adjacent tetrahedra have one spring constant and the inner cables that connect the tip of each tetrahedron to its adjacent base tetrahedrons have a different spring constant (see Figure 3). The damping constants are the same for both inner and outer cables. The free length is set to an appropriately small number in order to mimic cables that are always in tension. In a physical version of the five-tetrahedron robot, these cables would never experience compressive forces. Changing the value of the free length changes the average force in the tension members. This value was tuned ad hoc along with the spring constants to keep the five-tetrahedron tensegrity robot in an upright static position during simulation with no actuation. Table 3 shows the parameters used for both the inner and outer cables.

Table 3. Spring Model Parameters

Outer cable spring constant (k_s)	$0.2\ \frac{lb}{in}$
Inner cable spring constant (k_s)	$0.3\ \frac{lb}{in}$
Damping constant (c)	$0.05\ \frac{lb \cdot s}{in}$
Free length (x_o)	$0.04\ in$

The value of the spring constants mimic that of typical, non-stiff, metal extension springs. The value of the damping constant is kept low to mimic the damping response of a metal extension spring. The accuracy of the damping constant has not yet been verified. Because it is important to know how accurately the spring model mimics a real system of spring connections, future experiments must be conducted using actual hardware from a robot prototype. Increasing this

constant takes the model away from the desired similarities to metal extension springs and slows the response of the tension members. This changes the dynamics of the structure and EA optimization would likely produce different gaits.

Actuation of the five-tetrahedron tensegrity robot is achieved by changing the length of the tension members. Three methods were considered to do so. The first method is to change the free length, x_o, of the spring. The second method is to directly change the length of the spring by changing the displacement, x. The third method is to superimpose a force on the tension members. The third method was chosen because a superimposed force could realistically represent the force caused by an actuator as the superimposed force performs like an actuator in parallel with a spring.

2.2 Central Pattern Generator

The CPG used to generate locomotion gaits for the five-tetrahedron tensegrity robot consists of a system of Neural Oscillators (NOs) as introduced by Matsuoka [13]. Four such NOs are used to generate locomotion gaits and are arranged as seen in Figure 7. The NO output is applied as shown in Figure 3 where NO labeled 1 in Figure 7 corresponds to Signal 1 which actuates the first pair of cables.

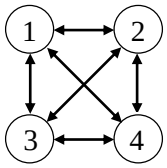


Figure 7. Mutual inhibition network of four NOs.

Each NO consists of two mutually inhibiting neurons, the flexor and the extensor. These neurons interact as seen in Figure 8 to generate an output signal. The equations that govern each flexor and extensor neuron, denoted with subscripts f and e, respectively, are

$$\tau_A \dot{u}_{i,(f,e)} = u_{i,(f,e)} - \sum_{j=1}^{n} w_{ij} y_{j,(f,e)} - \beta v_{i,(f,e)} + u_o + f_i, \text{ and} \qquad \text{Eq. 7}$$

$$\tau_B \dot{v}_{i,(f,e)} = -v_{i,(f,e)} + y_{i,(f,e)},$$

where u_i is the inner state of the neuron, v_i represents the self-inhibition effect of the neuron, w_{ij} are the inhibitory weights between i_{th} and j_{th} neurons, τ_A is a time constant that effects rising time of the output signal of an NO, τ_B is a time constant that effects adaptation, u_o is the external input state which defines the maximum amplitude of the CPG output signal S_{out} as defined in Eq. 9, β is the weight of self-inhibition on the inner state, y_i is the output of a neuron, and f_i is the feedback term.

The output of a neuron y_i is found with

$$y_i = \max(0, u_i), \quad \text{Eq. 8}$$

which is a positive threshold function for neuron output. A pair of extensor and flexor neurons form a neural oscillator with

$$S_{out} = y_e - y_f, \quad \text{Eq. 9}$$

where S_{out} is the output signal of a NO, y_e is the output of an extensor neuron, and y_f is the output of a flexor neuron. Table 4 contains the CPG parameters used.

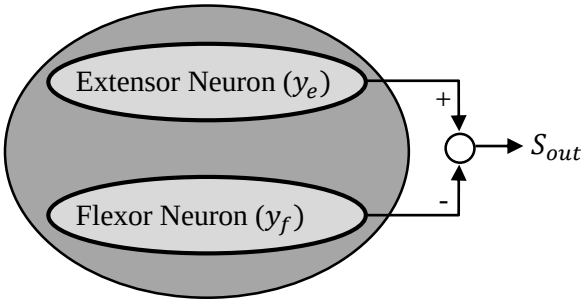


Figure 8. Matsuoka's neural oscillator.

Table 4. CPG Parameters

Weight of self-inhibition (β)	2.5
External input state (u_o)	0.5
Initial inner state (flexor and extensor) (u_i)	0.3
Initial extensor self-inhibition (v_i)	0.1
Initial flexor self-inhibition (v_{i+1})	0.5

In the current implementation, the output of each NO (S_{out}) is applied as a superimposed force in the actuated tension members as shown in Figure 3. The external input state, u_o, defines the maximum output amplitude of the oscillatory output signal S_{out}, meaning its value maps to a maximum change in the superimposed force in an actuated tension member. The value of the external input state in Table 4 was used in an attempt to keep the net tensile force low during locomotion as

each actuation signal has an amplitude of twice the external input state or one pound. This low net force reduces the likelihood of separate tetrahedron coming in contact with one another during dynamic simulation, eliminating the need for contact detection between tetrahedron. The value for the other CPG parameters were tuned ad hoc to drive the output of one NO to resemble a sinusoidal wave with an initial value of zero.

In the current implementation there are a total of 32 mutual inhibition network weights, w_{ij}. However, eight of these weights are set as constants as described in the Evolutionary Optimization section, leaving 24 weights and the time constants of the CPG to be changed by the EA to adapt the oscillatory output.

2.3 Evolutionary Optimization

The first step of the EA is to create an initial population. In this case, a member of the population consists of a unique set of parameters defining the CPG and the population of each trial contains 50 such members. A population size of 50 was chosen after experimentation with a variety of population sizes ranging from 10 to 200 individuals with the goal of the diversity of the population being reduced to a small value after 1000-2000 EA iterations. The procedure for creating the initial population begins with an analysis of a single neural oscillator. The adopted process was to begin with a CPG output that closely resembled a sine wave with the desired amplitude and a relatively slow frequency of approximately 0.7 Hertz. This frequency is a function of the weights and time constants that define an instance of the CPG. The values chosen were used because they created the expected gait as viewed in the SimMechanics visualization of the simulated robot.

When one NO is considered, the mutual inhibition weights consist of two weights which affect the interaction between the neural oscillator's flexor neuron and extensor neuron. Figure 10 illustrates how changing the weights of the CPG change the shape and frequency of the output. The upper plot in Figure 10 resembles a sine wave so values of 1.5 are used as the weights between each neural oscillator's flexor and extensor neuron. The time constants τ_A and τ_B are related by $\tau_B = 2\tau_A$. The factor of two relating τ_B to τ_A also helps shape the output to resemble a sine wave. Decreasing this factor below two causes the NO output to fall to zero and increasing it above two causes the output to stop resembling a sine wave, as seen in Figure 9. Thus, the values inhibitory weights between the extensor and flexor neurons of w =[1.5 1.5] and the time constants $\tau_B = 2\tau_A$ creates a CPG output signal from a single NO that resembles a sine wave.

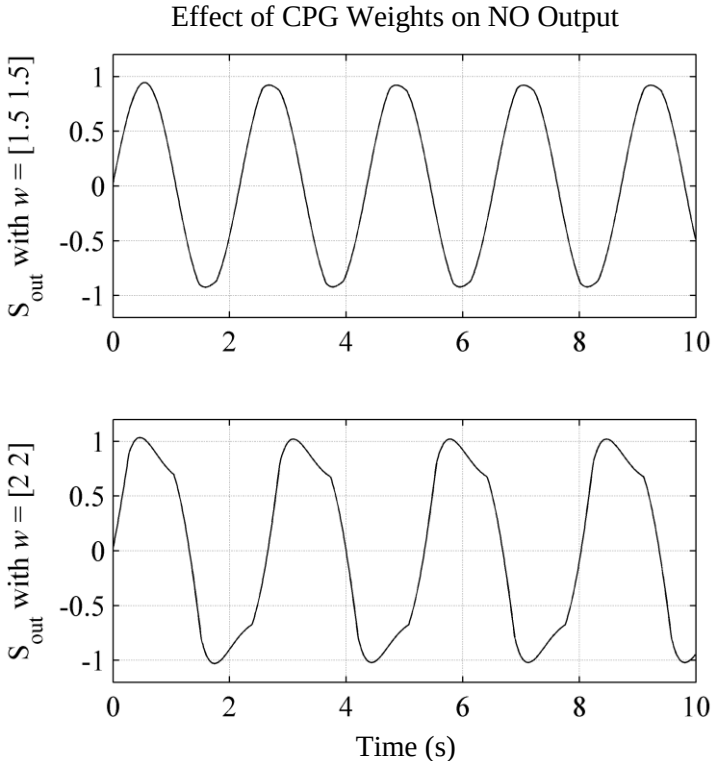


Figure 10. Comparing NO output signals with τ_A=0.3 and τ_B=0.6 for weights of [1.5 1.5] (top) and [2 2] (bottom).

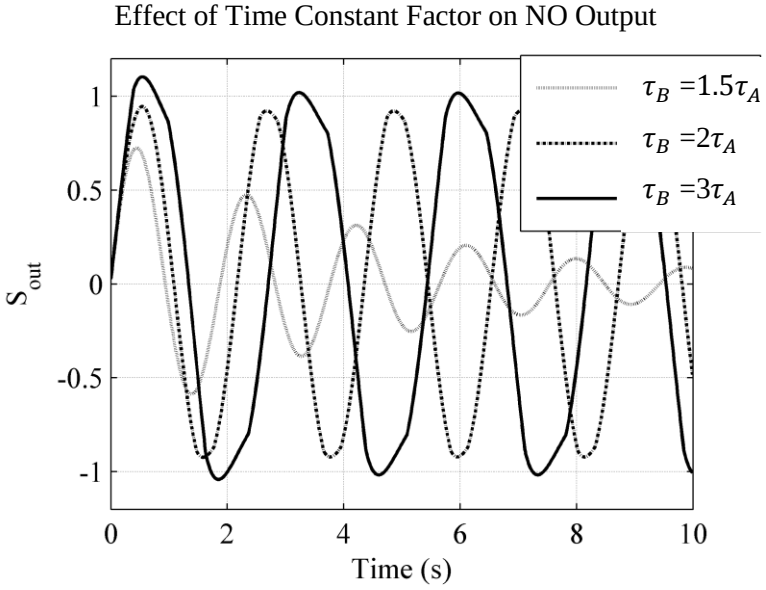


Figure 9. NO output signals with weights [1.5 1.5] and time constant factors of 1.5, 2, and 3.

Inhibitory terms are added as more neural oscillators are added and the additional weights on those terms reshape the outputs. The four NO arrangement used, as seen in Figure 7, has a total of 32 mutual inhibition weights. Because each NO has two neurons, eight weights are set as constant values of 1.5, leaving 24 weights to be changed by the EA.

The values of the weights and time constants are bounded to help ensure a usable oscillatory output signal from the CPG. The bounds for the 24 inhibitory weights were narrowed ad hoc to ensure the output signals produced were steady state oscillations in the initial populations. Outside of these bounds the output signals have a tendency to become unstable or fall to zero. Similarly, τ_A is bounded, and, as stated earlier, $\tau_B = 2\tau_A$ in order to crease a steady state output that resembles a sinusoidal wave. These bounds are listed in Table 5. Using these bounds the initial population is filled, or seeded, with 50 randomly generated, fitness evaluated individuals, which is used as the initial population for the EA iterations. This process is illustrated in Figure 11.

Table 5. CPG Parameter Initial Bounds

Bounds of weights (w_{ij})	1 - 3.4
Bounds of time constant (τ_A)	0.2 - 0.5

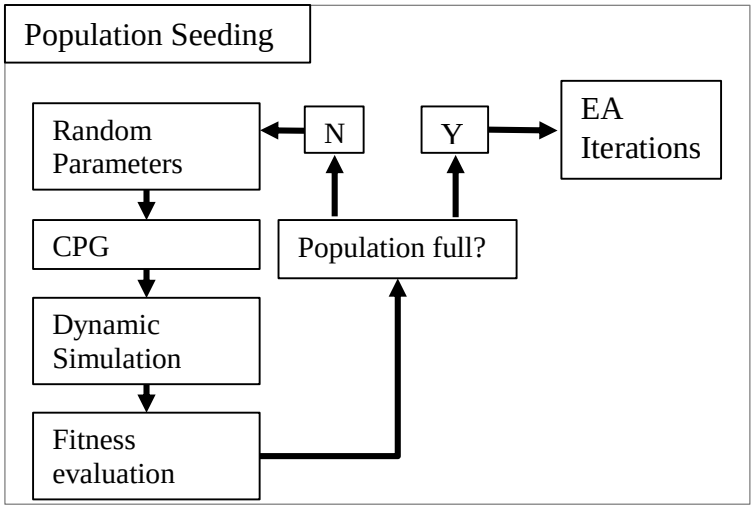


Figure 11. Initial EA population generation flow diagram.

EAs use two different types of operations to evolve the population of a system. These operations are selection and variation. Selection consists of collecting and replacing individuals from and into the population, and variation consists of changing the defining parameters themselves.

Once the initial population is generated, two parents are selected from the population to create two children which have a chance to replace individuals in the population depending on their performance. This method is known as steady state and is commonly used because the size of the population remains constant, meaning the population of each trial always contains 50 individuals. Each parent consists of the 24 CPG weights (ω_{ij}) and the 2 time constants (τ_A, τ_B).

The two parents are selected via tournament selection, where a group of five candidate parents are randomly selected from the population and the candidate with the highest fitness is chosen as a parent. Next, the two parents go through the uniform cross-over operation where they are mated, or have their parameters swapped randomly, producing offspring, or children. In uniform cross-over, a different probability of swapping is included on each CPG parameter of the parents (values listed in Table 6). A random number between zero and one is generated and compared against the probability value listed in the table to determine if the operation will occur. In addition, cross-over between parents can only occur between corresponding neurons. Each of the parameters of the resulting two children have a chance of a small amount of mutation, of the value defined in Table 6, before being used to evaluate a CPG output signal. The mutation is applied as a random value that has an equal chance of being added to, or subtracted from the existing value.

The output signals from the CPG are applied to the robot in the dynamic simulation. Data is taken from the simulation and is used to compute the fitness, as defined later in this section, of each of the resulting children. Each child is then placed into the population, replacing an individual that has a lower fitness. A method similar to tournament selection, called tournament replacement, is used for replacement. If the tournament replacement will not improve the fitness of the population the child instead replaces the individual in the population with the overall lowest fitness, as long as this

improves the average fitness of the population. This procedure of attempting tournament replacement before resorting to directly replacing the worst individual helps to loosen the replacement pressure which can widen the solution space searched. Figure 12 displays the EA operations for each iteration. This process is repeated until a termination criterion is met as defined later in this section. This entire process including initial population generation was repeated for 10 trials. Table 6 details the parameters used.

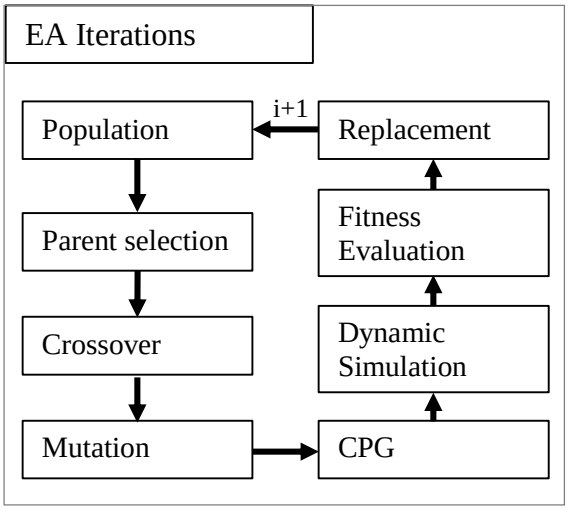


Figure 12. EA process overview flow diagram.

Two termination conditions were used for the evolution. The first is based on the diversity of the population and works by referencing the change in the average fitness of the population over a range of iterations. This method uses the increase of the average fitness values over the past 100 iterations to determine diversity. If the change in average fitness of the population falls below 0.03 over 100 iterations the simulation is stopped. The values of average fitness change and number of iterations were determined ad hoc by reviewing initial test trials. The second termination condition is based solely on the maximum number of iterations as defined in Table 6.

Table 6. EA Parameters

Simulation time	15 sec
Population size	50
Tournament size	5
Number of trials	10
Maximum iterations per trial	2000
Crossover type	Uniform
Crossover probability	20%
Mutation probability	70%
Mutation range for weights	0 - 0.2
Mutation range for time constant	0 - 0.05

The 15 second simulation time was chosen to ensure that enough time had passed to provide a full picture of the motion of any individual gait. A simulation time of 10 seconds was used in initial trials but was discarded because some gaits were not able to produce multiple steady-state oscillations in the simulated robot within that amount of time. The number of trials performed was determined by the elapsed simulation time for each dynamic simulation as each trial took between 5 and 15 hours to complete with the computer that was used. The crossover type was chosen to ensure that the EA was swapping corresponding weights between CPG signals. This can limit the solution space searched but the limiting effect can be managed with a large enough number of trials. The probability of the crossover and mutation were tuned ad hoc using results from multiple initial trials. The small probability of crossover and large probability of mutation, and the small mutation value limits the ability of the optimization to make large parameter jumps between individuals. Instead, the populations of the trials will converge slowly to optimums.

The fitness value is calculated via a manually tuned cost-function based on steady state data from the dynamic simulation. Figure 13 illustrates the locations where performance data is taken from

the dynamic simulation of the five-tetrahedron tensegrity robot. To allow for steady state conditions to be achieved a 3 second delay was added prior to data collection. This time delay removes transients caused by initial position errors, spring forces, and the CPG signal.

The fitness equation is defined as

$$Fitness = \phi_1 dx_1 + \phi_2|x_1 - x_2| + \phi_3(x_3 - x_1) + \phi_4|y_1|, \quad \text{Eq. 10}$$

where ϕ_{1-4} are constants, x_1, x_1, x_3, and y_1 are distances, as illustrated in Figure 13, recorded at the end of a dynamic simulation, and dx_1 is the distance x_1 recorded under steady state conditions, or between 3 seconds and the end of the simulation. Table 7 lists the constants used.

Table 7. Fitness Function Constants

ϕ_1	1
ϕ_2	-10
ϕ_3	0
ϕ_4	-1

The main contributor of the fitness value is the distance the robot traveled in the allotted time, thus, the value of ϕ_1 is unity. This makes the forward distance, dx_1, the basis for the fitness with the other three factors, $\phi_{2,3,4}$, functioning as penalties. The second term of the fitness function multiplied by ϕ_2 is a reflection of the twist the robot experiences while moving forward. This constant is much larger compared to the other constants that define the fitness because the value of $|x_1 - x_2|$ is often small and must be amplified to have an effect on the calculated fitness value. This term is included to add a penalty on twisting about the vertical axis that may have been caused by unwanted initial conditions caused by the initial spike in the CPG actuation signal. The third term multiplied by ϕ_3 is a represents the direction of motion relative to the desired direction. The desired direction of the robot is always in the positive x direction. The third constant ϕ_3 is set to zero because it is only needed when the robot is moving with a slithering, lateral undulatory motion which is not currently

implemented. The forth term multiplied by ϕ_3 represents the offset perpendicular to the desired direction of locomotion, or y direction.

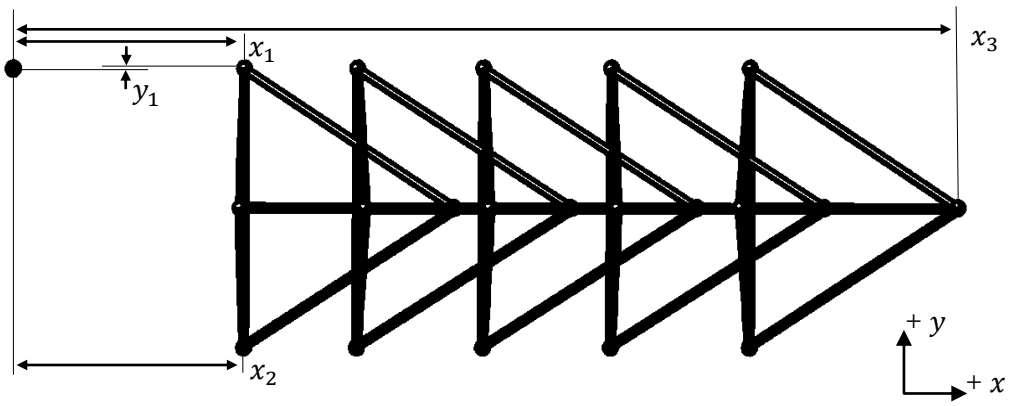


Figure 13. Location of x_1, x_2, x_3, and y_1 on the five-tetrahedron tensegrity robot.

2.4 Simulation Implementation

The EA (see Appendix A) and CPG (see Appendix B) were implemented using Matlab scripts and functions. The values and signals produced by the EA and CPG are passed to Simulink and SimMechanics for dynamic simulation. SimMechanics is a tool within the Matlab simulation environment Simulink, and is used to model mechanical systems.

The rigid tetrahedrons are modeled using SimMechanics. Rigid body transformations and rotations are applied to struts (see Appendix C.1) and SimMechanics blocks called Weld Joints (see Appendix C.2) fix the struts in their relative position. At each point where rigid struts are connected are 0.2 inch diameter spheres, also connected using Weld Joints. The points where struts are connected together are used as connection points for tension members while the spheres are used as connection points for the ground force models.

The ground force model is implemented using a combination of Simulink and SimMechanics function blocks (see Appendix C.4). Data is taken from sensor ports of SimMechanics six degree-of-freedom (DOF) joints. These six-DOF blocks supply Simulink blocks with data on the vertical position and velocity of the connecting node. The Simulink blocks are arranged to create the ground force model. The resulting value of force is fed back into actuation ports on the six-DOF SimMechanics joints.

The friction model is implemented in a similar fashion to the ground force model in that it uses a combination of the SimMechanics six-DOF joints and Simulink block functions (see Appendix C.4). The velocity of the node relative to the ground is taken from sensor ports of the six-DOF joints and is passed through the friction force model that is modeled using Simulink. The resulting force value is passed back into the six-DOF SimMechanics joints. This friction model is applied in both the x and y directions, where x is the direction of travel of the robot and y is perpendicular to the direction of travel.

The tension members are also modeled using six-DOF joints from SimMechanics with Simulink blocks calculating spring forces (see Appendix C.5). The relative distance and velocity between corresponding connection nodes on the tetrahedron are taken from sensor ports on the six-DOF joints and the Simulink block functions use that data to compute the force to be applied to each of the connecting points. This value of calculated spring force is passed into the actuation ports on the SimMechanics six-DOF joints.

The dynamic simulation presented in this thesis could be applied to any tensegrity structure, including more complex tensegrity structures and true tensegrity structures (no rigid connections between compression members). The challenges in doing so would most likely be caused by initial conditions distance in the SimMechanics six-DOF joints. These initial conditions must be close to the values that would define a static pose for a tensegrity structure. This adds the issue of over defining the initial states of the structure which can cause SimMechanics to crash.

One iteration of the EA constitutes two dynamic simulations, one for each of the two offspring, and takes an average of 22 seconds to complete. The elapsed simulation time is governed by the dynamic simulation as the CPG and EA Matlab execution takes very little time. The dynamic simulations 15 seconds long as defined in Section 2.3 meaning that the current dynamic simulation is actually slower than real time.

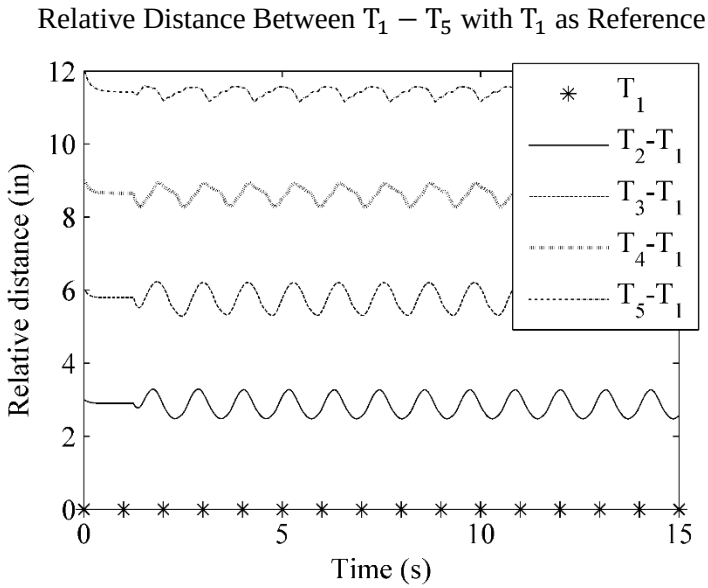


Figure 14. Relative distance between tetrahedra during locomotion with reference T_1.

Chapter 3. Results

The motion achieved by this robot is generally as illustrated in the overhead-view given in Figure 15, and the in graph of relative distance between tetrahedra as the robot generates forward motion in Figure 14. The following sections present the results of the CPG gait evolution of the five-tetrahedron tensegrity robot. The first section shows gaits resulting from the initial gait generation, where gaits were created using pseudo-random number generator and the Matsuoka CPG. The second section shows the gaits produced by the evolutionary algorithm and the forces at work in the structure during locomotion. The third section covers the behavior of the EA optimization during the 10 trials.

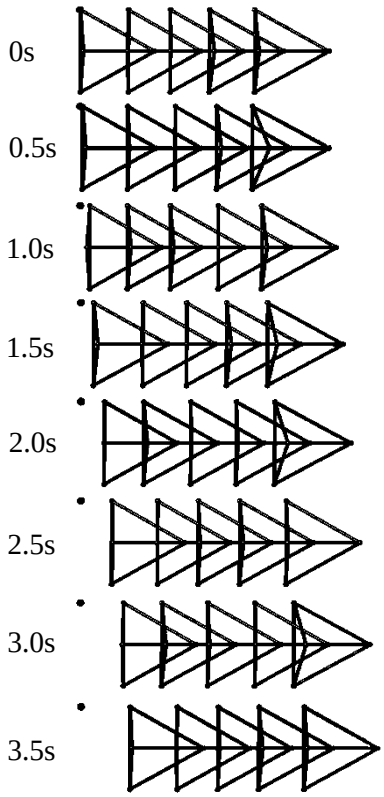


Figure 15. Time lapse of locomotion.

3.1 Initial Gait Generation

A sampling of the performance of the randomly generated initial gaits, i.e. the initial population, is presented in Figure 16, which shows the forward distance traveled over a 15 second simulation, x_1. This figure displays the best and worst of the 50 individuals of the initial populations from each of the 10 trials performed. Figure 17 shows the performance, in terms of fitness, of the 50 individuals from the initial population of one sample trial. These figures demonstrate the variety of gaits in the initial populations as nearly half of the individuals travel opposite the desired direction of travel. This variety helps to ensure that more of the solution space is searched by the EA.

The final distance reached in the best performing gait from the initial population is approximately 6.6 inches. The data shows periodic surges in the distance traveled from the origin; this is because, by the undulatory nature of the locomotion, no one section of this robot has a constant velocity during locomotion.

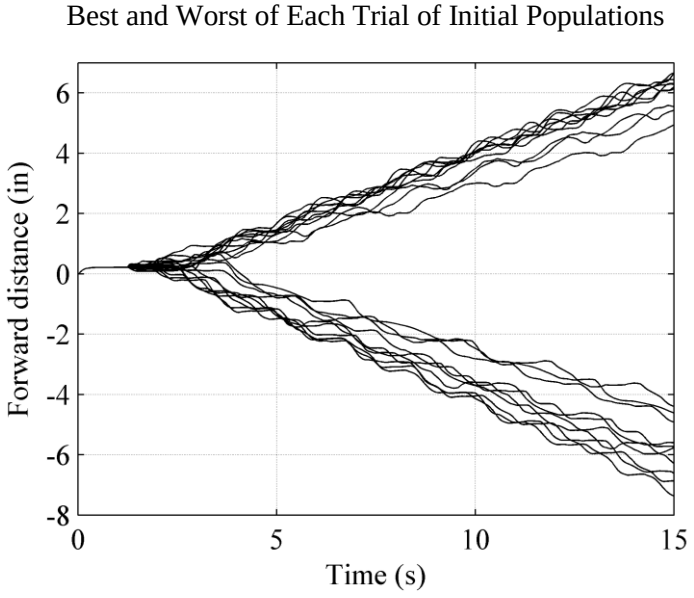


Figure 16. Distance traveled of best and worst individual of each of the 10 trials.

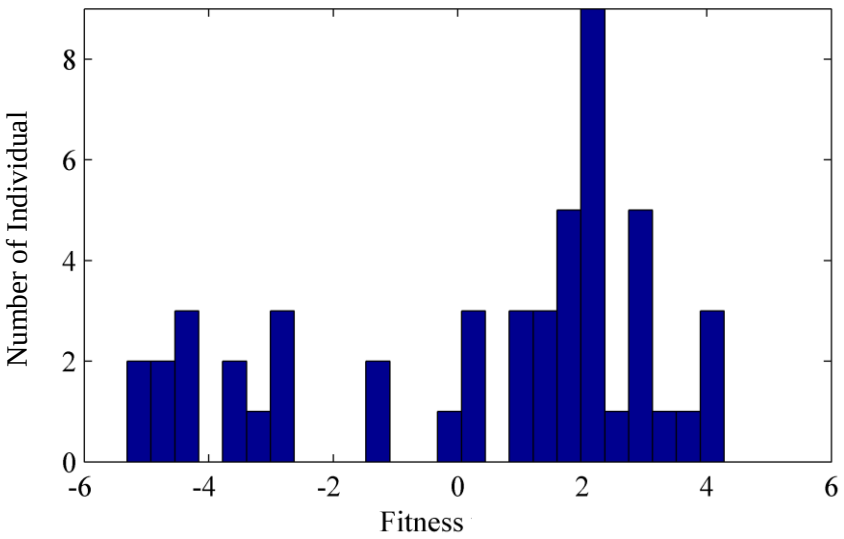


Figure 17. Distribution of fitness of initial population from a sample trial.

3.2 Gait Optimization

As defined earlier in section 2.3, all ten EA trials had a population size of 50 and used the main fitness performance metric of maximizing the distance traveled. Minor factors that affect the fitness value are minimizing the distance traveled perpendicular to the desired direction of travel, and minimizing the twist about the vertical axis of the robot. The minor factors were small throughout the trials and were nearly eliminated by the end of all trials.

Figure 18 is a comparison of the forward distance traveled by the best of each of the initial and evolved populations. The average performance increase over 10 trials is 32 percent and the best performing individual from all 10 trials increased the final distance traveled by 51 percent with a distance traveled of 14.34 inches in the 15 second dynamic simulation.

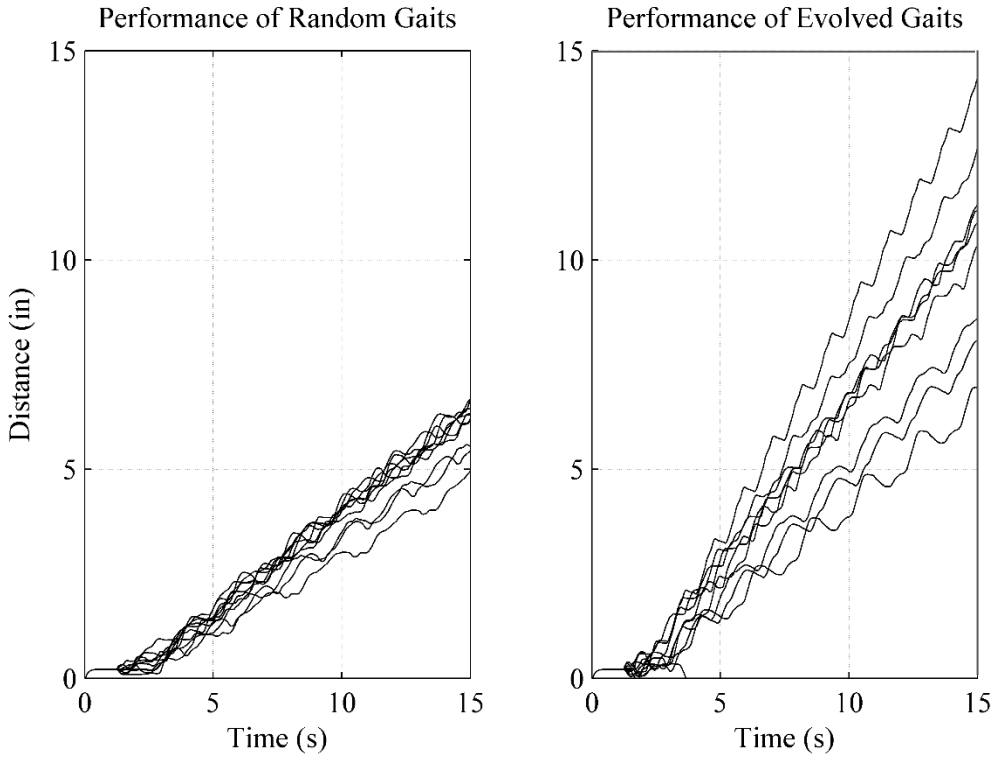


Figure 18. Distance traveled performance of the best individual from the initial population (left) and after evolving (right).

Next we look at the CPG signals that were used to generate the gaits and the resulting forces in the robot. As stated earlier, each pair of tension members between adjacent tetrahedron are actuated by one CPG signal, thus, the forces in the tension members and in the ground contact of the tetrahedron are symmetric and only one of each pair is shown in the following figures. Figure 19 shows the CPG signal that was superimposed on the spring force in the actuated tension members, and the resulting force in those tension members. Comparing the CPG input and tension forces we can see that the dynamics of the robot introduced a time delay into the controlled tension member's force as well as reshaped them.

Figure 20 shows the frictional and normal forces at work on each of the five tetrahedron in the robot. This graph demonstrates how the weight of the robot is distributed between the points of ground contact and how friction is utilized as the robot moves forward.

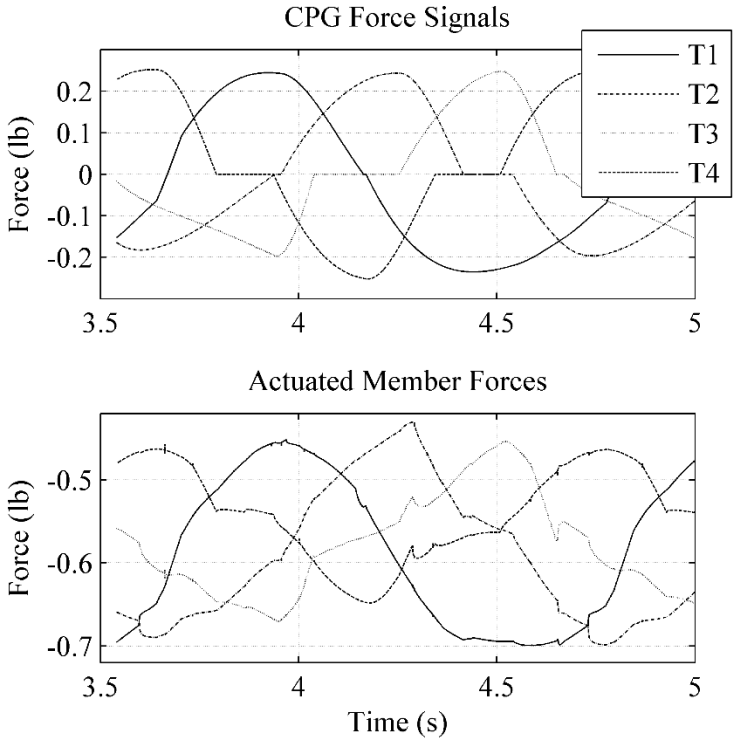


Figure 19. Graph of CPG signal and corresponding tension force in actuated tension members.

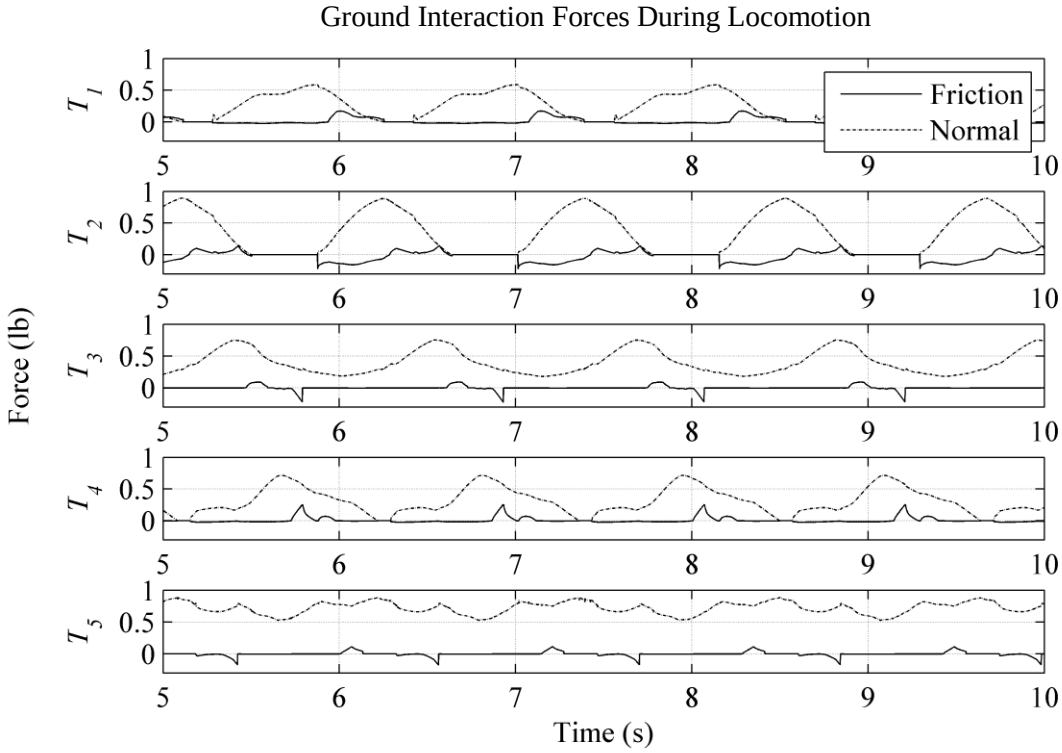


Figure 20. Friction and ground forces of tetrahedron T_1, T_2, T_3, T_4, and T_5 during locomotion.

Currently, directional friction is not implemented in the dynamic simulation of this robot. However, it is noteworthy that adding directional friction to the dynamic simulation of this robot may improve its performance. Because directional friction is not modeled, the evolved gaits must rely on a different main mechanism for producing forward motion. This mechanism is seen as lifting a section, or tetrahedron, of the robot off the ground before moving it forward and the results show that this structure is able to generate forward motion using this mechanism. This behavior is apparent in Figure 20 whenever the normal force on a tetrahedron due to the ground is equal to zero. This is made possible by the mode of actuation where tetrahedron are allowed to twist about their axis. In this simulation such twisting is achieved by actuating only the bottom outer tension members, however, actuating all members could potentially achieve the same behavior. Figure 21 illustrates this effect with traced points overlaid on images of the robot.

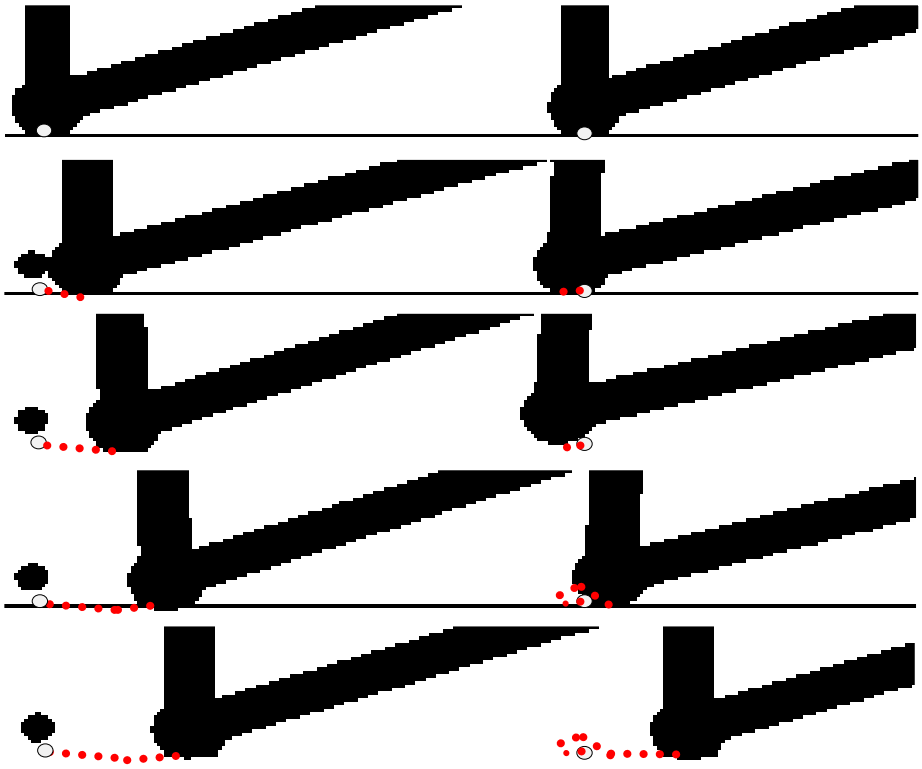


Figure 21. Time lapse of one step: motion achieved by shifting weight between tetrahedra; lifting tetrahedra off the ground before moving them forward.

3.3 Optimization Behavior

Reviewing the data shows the final results of the optimization, but it is also important to analyze the path the EA took to reach those results. These paths are best represented in this model by the average fitness of the population.

Figure 22 shows the progression of the average fitness of the 10 trials performed. During the trials the EA found multiple local maximum but in most cases was able to break out and continue increasing performance. It is noteworthy that the performance of the best trial was still increasing when it met the maximum iteration termination criterion, meaning it may have been capable of finding a better gait if the maximum number of iterations was increased. As this is an initial approach the maximum number of iterations was not increased; instead this result can be used in future work to improve the size of the searched solution space, allowing for the possibility of better performing populations.

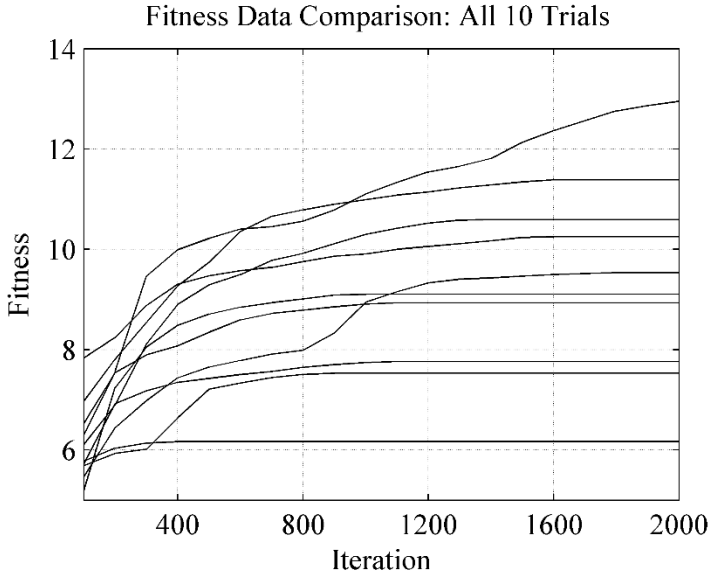


Figure 22. Average fitness progression of all 10 Trials.

Figure 23 shows the performance of the best trial when compared to the average of all ten trials. The dotted line represents the value of the average fitness of all ten trials and the bars represent

the standard deviation of the fitness of all ten trials. The trend of all solutions was to increase the fitness as expected and solutions other than the "best" solution had higher values earlier in the trials.

The results of the EA show that the average fitness of the population was increased in all trials. The resulting populations in all trials showed that the EA attempted to minimize the time constants of the CPG τ_A and τ_B to their lower limits. The final values of the weights of the CPG did not show any discernable patterns between trials, meaning individuals with good performance do not necessarily have similar CPG weights to other individuals that perform well. This is possibly because a wide range of gaits can be achieved by this system (i.e. walking, running). This idea is validated in part by Figure 22, where it is apparent that there not an optimum that the majority of the populations in the 10 trials agreed upon.

The results of the two best individual from the best performing trial show that the values of the inhibitory weights can vary by a large amount and still result in the same fitness value, meaning that there are many valid solutions to this problem. The fitness value of these two solutions varies by 0.0006 and the time constants have exactly the same value, but, the values of the weights varies wildly with differences as low as 0 and as high as 0.38. Table 8 demonstrates this with the values of the absolute difference of the fitness and parameters defining the two best individuals produced in the best performing trial. Although some of the parameters of these individuals are similar (shown by small values in the table below), these similarities are not necessarily shared with even the next best individual in the same population. In addition, the best solutions of all trials show the same wide variety of inhibition weight values.

Table 8. Absolute Difference of the Two Best Solutions of the Optimization.

Fitness	w_{1-4}	w_{5-8}	w_{9-12}	w_{13-16}	w_{17-20}	w_{21-24}	τ_A, τ_B
0.0006	0.0594	0.1184	0.0874	0.0738	0.0288	0.0125	
	0.1063	0	0.064	0.2638	0.1233	0.0244	0
	0.0919	0.11	0.1928	0.1313	0.1689	0.0357	0
	0.0329	0.1364	0.3157	0	0.3806	0.1322	

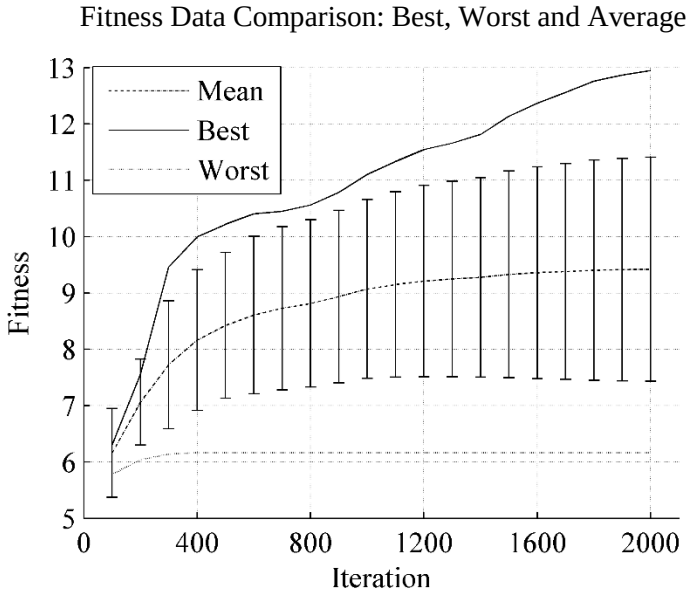


Figure 23. Average fitness of trials: best, worst, and average of all 10 trials.

Chapter 4. Conclusion and Discussion

This thesis presents a method to create locomotion gaits for a modular tetrahedron tensegrity robot using an Evolutionary Algorithm (EA) to optimize the parameters of a Central Pattern Generator (CPG). The results from simulation demonstrate the efficacy of the approach with the performance metric total of distance traveled; however, other goals could be applied using the same method. For instance the range of optimized parameters could be extended or offset to allow for a wider solution space or for choosing specific types of gaits. Specifying different ranges of parameters could produce different characteristics in gaits such as walking or running. It may also be advantageous to add a factor to the fitness evaluation that promotes smoothness of CPG signals.

The produced locomotion is limited by the mode of actuation where the same CPG output signal is applied to both actuated tension members between adjacent tetrahedra to create a caterpillar or inchworm type locomotion where tetrahedra are lifted and step forward. This structure may also be capable of generating forward motion using a snake-like undulation motion. This motion could be achieved by adding individual CPG Neural Oscillators (NOs) to each of the lower outer tension members, making a total of eight separate NOs.

The main limitation of this simulation is computational speed. As stated in Section 2.4, the dynamic simulation is slower than real time. The method presented here is meant to be an initial attempt to validate the efficacy of the methods used. However, if in the future the simulation speed were increased, more processing power were added, or the trials were allowed to run for a longer period of time, more tetrahedron and CPG signals could be simulated for a more interesting structure and possibly more efficient locomotion.

This simulation was performed with the goal to apply the findings to a physical prototype. This would also help validate the CPG used, the effectiveness of the EA optimization, the actuation methods, and some the parameters used in the ground, friction, and tension member models. The prototype at NASA Ames could be used but many adjustments would have to be made to the

simulation to match the physical prototype. There is currently a new prototype being constructed at the University of Idaho that fits the design of the robot in simulation. Future work will be performed to compare the simulation to this prototype.

The gaits produced by the EA showed that the time constants were minimized. For this reason the values of these parameters could be set as constants depending on the desired locomotion cadence. There is a possibility that this would limit the performance of a population, but, current findings suggest that this possibility is small. The advantage of setting these parameters to constants could mean fewer iterations of the EA to reach the same outcome. In addition the two time constants are related by a factor of two, which may limit the variety of gaits produced. Future trials could allow the time constants to be changed independently within appropriate ranges.

CPGs are used in the robotics community partly because of their resistance to perturbations even without feedback. For this reason the simulation could be modified to include environmental obstacles such as inclines, steps, or objects. This would demonstrate the robustness of the final optimized gait. In addition feedback pathways could be added to the simulation to aid in obstacle maneuverability.

References

[1] V. G. Jauregui, "Controversial Origins of Tensegrity," in *Internatioan Association for Shell and Spacial Structures Symposium*, Valencia, 2009.

[2] D. E. Ingber, "Tensegrity," *Scholarpedia*, p. 7(2):8344, 2012.

[3] R. E. Skelton and M. C. d. Oliveirra, Tensegrity Systems, La Jolla, CA: Springer, 2009.

[4] T. Flemons, "Intension Designs," [Online]. Available: http://www.intensiondesigns.com/models.html. [Accessed November 2013].

[5] K. Kebiche, M. Kazi-Aoual and R. Motro, "Geometrical non-linear analysis of tensegrity systems," *Engineering Structures 21*, pp. 864-876, 1999.

[6] J. Yu, "A Survey on CPG-Inspired Control Models and System Implememtation," *IEEE Transactions on Neural Networks and Learning Systems*, 2013.

[7] Q. D. Wu, C. J. Liu, J. Q. Zhang and Q. J. Chen, "Survey of locomotion control of legged robots inspired by biological concept," *Sci. china Ser.*, vol. vol. 52, no. 10, pp. 1715-1729, 2009.

[8] Z. Xiuli, "Study on dynamics of Central Pattern Generator Model for a Quadruped Robot," Unpublished, Beijing, 2005.

[9] M. M. Williamson, "Neural control of rhythmic arm movements," *Neural Networks*, vol. 11, no. Special Isssue, pp. 1379-1394, 1998.

[10] T. K. Bliss, "CPG control of a tensegrity morphing structure for biomimetic applications," *Advances in Science and Technology*, pp. 137-142, 2008.

[11] A. Crespi, "AmphiBot I: an Amphibious snake-like robot," *Robotics and Autonomous Systems*, pp. 163-175, 2005.

[12] K. Inoue, "Neural oscillator network-based controller for meandering locomotion of snake-like robots," in *IEEE Int. Conf. Robot. Autom.*, New Orleans, LA, USA, 2004.

[13] K. Matsuoka, "Sustained Oscillations Generated by Mutually Inhibiting Neurons with Adaptation," *Biological Cybernetics,* pp. 367-376, 1985.

[14] G. Endo, J. Nakanishi, J. Morimoto and G. Cheng, "Experimental studies of a neural oscillator for biped locomotion with QRIO," in *IEEE International conference on robotics and automation*, Barcelona, Spain, 2005.

[15] C.-S. Park, J.-K. Yoo, Y.-D. Hong, K.-B. Lee, S.-J. Ryu and J.-H. Kim, "Walking Pattern Generator Using an Evolutionary Central Pattern Generator," in *FIRA Robot World Congress*, India, 2010.

[16] A. E. Eiben, Introduction to Evolutionary Computing, Springer, 2007.

[17] T. Reil and P. Husbands, "Evolution of central pattern generators for bipedal walking in a real-time physics environment," *IEEE Transactions on evolutionary computation,* vol. Vol. 6, no. 2, pp. 159-168, 2002.

[18] R. Lafarge, "Contact force modeling between non convex objects using a nonlinear damping model," in *North American ADAMS User Conference,* Ann Arbor, MI, 1988.

[19] H. Olsson, "Friction Models and Friction Compensation," *European Journal of Control* , pp. 176-195, 1998.

Appendix A. Evolutionary Algorithm

The following Matlab code is the Evolutionary Algorithm (EA), and the Central Pattern Generator (CPG), and included sub functions. A.1 is the initial population generator and was used to create bounded, fitness evaluated populations of random solutions which are made up of parameters that define the CPG and fitness that define the performance. A.2 is the initial fitness evaluation and takes the initial population and computes the fitness of each individual, then stores the results in the population that is used by EA main. A.3 is the main EA script with sub functions included as A.3.1-5

A.1 Initial population generator

```
%% George Korbel
% Initial Population generator
% Generates values for CPG and simulates
% This file was created to work specifially with simulink file:
% tetra5tv5
% Notes: --dimensions are ips
% Tetra5tv5: simplified addition of tetrahedron and NOs
% Changed sensing points and method of sensing
% Updated structure of pop
% Tetra5_v4: version adds 3 neural oscillators in Central Pattern Generator (CPG) and population
seeding
% working matrix name dist_max changed to pop
% created by: George Korbel
 %%
clc
 global w tau dtau
 % Number of simulations
stop = 50;
 % Time of simulation
simtnum = 15; % simulation time in number form
simtime = num2str(simtnum);
set_param('tetra5tv5_fas','MaskedZcDiagnostic','none','StopTime',simtime);
% for running one sim at a time
if stop == 1
    pop = 0;
end
%Check if pop exists
ex = exist('pop','var'); % 1 if yes, 0 if no
%start counter on new row of pop matrix
if ex == 1
    [start, na] = size(pop); % current # of rows in pop = start
    if pop(start,1)==0 % if the last entry in the first column is zero
```

```
        start = start;      % start on that row
    else
        start = start+1;    % else start on a new row
    end
else
    error('pop is not defined') % if variable pop dne throw error
end
% Set friction params from guessing and watching
% (these values were taken from a random search simulation and adapted by hand to
% help with convergence of the simulations)
s0 = .3;   % Sigma 0 = damping during sliding
s1 = .4;   % Sigma 1 = damping during sticking
a = .7;    % Coefficient multiplied by sigma0 (turns friction on or off with z detection)
p0 = .004; % Distance node can move under static friction
% One way friction ratio
% removed in this version
%f_ratio = 1; % 1 = no one way friction, 0 = complete one way friction
% Random value arrays
N = 2000; %how many values between bounds
wa = 1:(3.4-1)/N:3.4; %value array for weights
taua = 0.2:(.5-.2)/N:.5;  %value array for tau
for k = start:stop
    display(num2str(k)) %print iteration #
    % CPG parameters
    % Weights
    wnum = 24;
    for wk = 1:wnum
        w(wk) = wa(randi([1,N]));
    end
    % Time constants
    tau = taua(randi([1,N]));
    dtau = 2*tau;
    tic           %start timer
    display('CPG') %print progress
    CPGmain() %run cpg
    display('Simulating') %print progress
    sim('tetra5tv5_fas')      %simulate current model
    % Save results in pop
    % First tetrahedron to origin
    dist_x = sx1(end) - sx1(2);
    pop(k,1) = dist_x;      %end distance in x of node that starts at origin
    pop(k,2) = abs(sx1(end) - sx2(end));     %end difference in distance of rear two nodes that begin at
origin
    pop(k,3) = 0;  %end distance between front tetra tip and node that begins on origin
    pop(k,4) = abs(sy1(end));   %end y distance of node that begins on origin
    % Save CPG parameters
    for ct = 1:wnum
        pop(k,ct+4) = w(ct);
    end
    pop(k,29) = tau;
```

```
    pop(k,30) = dtau;
    toc %output total time for last run
    display(' '); display(' '); %returns for display in command window
end
uni_pop = unique(pop,'rows'); %deletes any repeating rows (entire row must be the same)
sort_pop = sortrows(pop,[-1]); %sorts in descending order by first column
% plot results
figure;h1=stem3(sort_pop(:,2),sort_pop(:,3),sort_pop(:,1),'filled');
```

A.2 Initial fitness evaluation

```
%% Evolutionary Algorithm fitness initial evaluation
% unrated population is named pop
% largest value is best fitness
% load pop
% Weights for EA fitness evaluation
% origin-tetra1
W1 = 1;   % total distance traveled % larger the better % needs to be rescaled so that it does not
overshadow bad effects??
W2 = -10;   % absolute difference between two nodes that start on x = 0 (in x dir) % this value is
generally worse when its abs is larger
W3 = 0;   % difference between two nodes that start on x = 0. non-absolute
W4 = -1; % y distance of origin node
fit_w = [W1 W2 W3 W4];
[L,tmp] = size(pop);
population = zeros(L,27); % initialize population size
% Evaluate fitness and store in population
for k = 1:L
    temp = pop(k,:); % separate out one row of unevaluated population
    %     if temp(1,1) < 0 % If it moves backwards give it a small fitness
    %         fit = -5;
    %     else
    Lfw = length(fit_w);
    for m = 1:Lfw
        fitsum(m) = temp(1,m)*fit_w(m);
    end
    fit = sum(fitsum);
    population(k,:) = [fit, temp(1,5:end)];
end
```

A.3 Evolutionary Algorithm main

```
% Evolutionary Algorithm main
% Low probability of swapping
% High probability of mutation
% Low mutation value
% Details:
% - Selects five candidate parents at random
```

```
% - Takes two best of five selected for actual parents
% - Swaps random weights and time constants
clear tmppop
global w tau dtau Rp
% Number of simulations
stop = 2000;
ex = exist('fit_ave');
%start counter on new row of fit_sum array
if ex == 1
   start = length(fit_ave); % current # of rows
   start = start + 1;      % start on that row
else
   error('fit_ave is not defined') % if variable pop dne throw error
end
% Set friction params from guessing and watching
% (these values were taken from a random search simulation and adapted by hand to
% help with convergence of the simulations)
s0 = .3;   % Sigma 0 = damping during sliding
s1 = .4;   % Sigma 1 = damping during sticking
a = .7;    % Coefficient multiplied by sigma0 (turns friction on or off with z detection)
p0 = .004; % Distance node can move under static friction
% One way friction ratio
f_ratio = 1; % 1 = no one way friction, 0 = complete one way friction
% Find size of population
[Rp, Cp] = size(population);
wnum = 24;
for k = start:stop
   display(num2str(k)) %print iteration #
   %% Selection
   % Select five random candidate parents
   [parent1] = EA_select(population);
   [parent2] = EA_select(population);
   %% Crossover
   sw_prob = rand(1,Cp-1);  % probability of swapping
   for n = 1:(Cp-1)
      if sw_prob(n) > .8 % if the probability is over threshold swap columns
         swtmp(1,n) = parent1(1,n);
         parent1(1,n) = parent2(1,n);
         parent2(1,n) = swtmp(1,n);
      end
   end
   parent1(1,Cp) = parent1(1,Cp-1)*2; % if tau was changed also change dtau
   parent2(1,Cp) = parent2(1,Cp-1)*2; % if tau was changed also change dtau
   %% Mutation
   % weights
   mut_op = rand(1,wnum+2);        % probability of mutation operator (0:.49 subtracts .5:1 adds)
   mut_prob = rand(1,wnum+2);      % probability of mutation
   mut_fact_w = rand(1,wnum+2)/5;     % factor of mutation for w (between 0 and .2)
   parent1 = EA_mut_w(parent1, mut_op, mut_prob, mut_fact_w); % mutates weights
   parent2 = EA_mut_w(parent2, mut_op, mut_prob, mut_fact_w); % mutates weights
```

```
    mut_fact_tau = rand/20; % factor of mutation for tau (between 0 and .05)
    parent1 = EA_mut_tau(parent1, mut_op, mut_prob, mut_fact_tau);
    parent2 = EA_mut_tau(parent2, mut_op, mut_prob, mut_fact_tau);
    %% Re-simulate with children
    child = [parent1; parent2];
    for i = 1:2
        %tic;
        w = child(i,1:wnum);
        tau = child(i,wnum+1);
        dtau = 2*tau;
        %% Run Central Pattern Generator
        %display('CPG') %print progress
        CPGmain() %run cpg
        %% Run Simulator
        set_param('tetra5tv5_fas','MaskedZcDiagnostic','none'); % turn off warning for zero crossing
        %display('Simulating') %print progress
        sim('tetra5tv5_fas')      %simulate current model
        %% Save results in tmppop
        % First tetrahedron to origin
        dist_x = sx1(end) - sx1(2);
        tmppop(1,1) = dist_x;      %end distance in x of node that starts at origin
        tmppop(1,2) = abs(sx1(end) - sx2(end));     %end difference in distance of rear two nodes that
begin at origin
        tmppop(1,3) = 0; %This only needs to be non zero during snake like motion  %sx2(L) - sx1(L);
%end distance between front tetra tip and node that begins on origin
        tmppop(1,4) = abs(sy1(end));   %end y distance of node that begins on origin
        % Save CPG parameters
        for ct = 1:wnum
            tmppop(1,ct+4) = w(ct);
        end
        tmppop(1,wnum+5)  = tau;
        tmppop(1,wnum+6) = dtau;
        %% Evaluate fitness of new child solution
        fit = EA_fitness(tmppop);
        %% Place child back in population
        pos = EA_replace(population);   % worst fitness out of group of 5 random
        minfit = min(population(:,1)); % find worst fitness overall
        [row, col] = ind2sub(size(population),find(population==minfit));
        or_pos = row(1); % row of worst overall
        current_fit = population(pos,1);
        if fit > current_fit
            population(pos,:) = [fit, w, tau, dtau]; %replace worst out of the 5
        elseif fit > minfit
            population(or_pos,:) = [fit, w, tau, dtau]; %or replace worst overall
        end
        %toc
    end
    fit_ave(k) = sum(population(:,1))/Rp; %save the average fitness
    % Print progress by summing fitness column
    %display(num2str(fit_ave(k)))
```

```
   % Stopping criterion
   if k > 200
      diff_fit_ave = fit_ave(k) - fit_ave(k-100);
      if diff_fit_ave < .03
         for i = 1:1000
            beep;
            pause(1);
         end
      end
   end
   if k == 2000
      for i = 1:10000
         beep;
         pause(1);
      end
   end
   clc
   %display(' '); display(' '); %2 returns for display in command window
end
```

A.3.1 EA Fitness

```
 function [fit] = EA_fitness(tmppop)
% Function Evolutionary Algorithm fitness evaluation
% largest value is best fitness
% load pop
% Weights for EA fitness evaluation
% origin-tetra1
W1 = 1;   % total distance traveled % larger the better % needs to be rescaled so that it does not
overshadow bad effects??
W2 = -10;   % absolute difference between two nodes that start on x = 0 (in x dir) % this value is
generally worse when its abs is larger
W3 = 0;   % difference between two nodes that start on x = 0. non-absolute
W4 = -1; % y distance of origin node
fit_w = [W1 W2 W3 W4];
temp = tmppop;
% if temp(1,1) < 0 % If it moves backwards give it a small fitness
%     fit = -5;
% else
for m = 1:4
   fit(m) = temp(1,m)*fit_w(m);
end
fit = sum(fit);
```

A.3.2 EA select

```
function [ parent ] = EA_select( population )
%Selects parents for next combination and mutation
%   Takes five candidtate parents and selects the best two out of the five
global Rp select_pop
can1 = randi([1,Rp]);
can2 = randi([1,Rp]);
can3 = randi([1,Rp]);
can4 = randi([1,Rp]);
can5 = randi([1,Rp]);
can = [can1, can2, can3, can4, can5]';
select_pop = sortrows(population([can1;can2;can3;can4;can5],:),[-1]); % select parents out of
population and sort descending
select_pop = [select_pop, can];
parent = select_pop(1,2:27); % Take best of the five selected (don't take fitness value)
end
```

A.3.3 EA weight mutation

```
function [ parent ] = EA_mut_w(parent2, mut_op, mut_prob, mut_fact_w )
% Mutation function for weights array w
%   When called in EA_main randomly mutates values of w
for m = 1:24
   if mut_op(m) > 0.5
      if mut_prob(m) > 0.3 % if the probability is over threshold mutate
         parent(1,m) = parent(1,m) + mut_fact_w(m);
      end
   end
   if mut_op(m) < 0.5
      if mut_prob(m) > 0.3 % if the probability is over threshold mutate
         tmp = parent(1,m) - mut_fact_w(m);
         if tmp > 0 % but only if it will not cause a negative value
            parent(1,m) = parent2(1,m) - mut_fact_w(m);
         end
      end
   end
end
end
```

A.3.4 EA time constant mutation

```
function [ parent ] = EA_mut_tau( parent, mut_op, mut_prob, mut_fact_tau )
% Mutation function for tau
%   When called in EA_main randomly mutates values of tau
m = 26; % Only working on tau
if mut_op(m) > 0.5
   if mut_prob(m) > 0.3 % if the probability is over threshold mutate
```

```
        parent(1,m) = parent(1,m) + mut_fact_tau;
    end
end
if mut_op(m) < 0.5
    if mut_prob(m) > 0.3 % if the probability is over threshold mutate
        tmp = parent(1,m) - mut_fact_tau;
        if tmp > .2 % but only if it will not go below threshhold
            parent(1,m) = parent(1,m) - mut_fact_tau;
        end
    end
end
parent(1,m+1) = 2*parent(1,m); % dtau = 2*tau
end
```

A.3.5 EA replace

```
function [ pos ] = EA_replace( population )
%Selects worst
%   Takes five candidtates and selects the worst
[Rp, Cp] = size(population);
can1 = randi([1,Rp]);
can2 = randi([1,Rp]);
can3 = randi([1,Rp]);
can4 = randi([1,Rp]);
can5 = randi([1,Rp]);
can = [can1, can2, can3, can4, can5]';
select_pop = population([can1;can2;can3;can4;can5],:); % select parents out of population and sort
descending
select_pop = [select_pop, can];
select_pop = sortrows(select_pop,[-1]);
replace = select_pop(5,:); % Take worst of the five selected
pos = replace(end);
end
```

Appendix B. Central Pattern Generator

The following is Matlab code used to model the CPG. B.1 is the main script of the CPG. B.2 is the body of the Matsuoka CPG. B.3 is the positive threshold function.

B.1 CPG main

```
%% CPG main
% close all;
global u v y tau dtau Ts NeuralN
NeuralN = 8;
Ts = 0.0005;
TestT = 15;
Fs = 1/Ts;
TestS = TestT/Ts;
Time = (0:TestS-1)*Ts;
Dper1 = 1;
Dper2 = 40;
if(Dper1 == 1)
   Dindex1 = 1;
else
   Dindex1 = TestS * Dper1 / 100;
end
Dindex2 = TestS * Dper2 / 100;
u(1) = 0.3;
u(2) = 0.3;
u(3) = 0.3;
u(4) = 0.3;
u(5) = 0.3;
u(6) = 0.3;
u(7) = 0.3;
u(8) = 0.3;
v(1) = 0.1;
v(2) = 0.5;
v(3) = 0.1;
v(4) = 0.5;
v(5) = 0.1;
v(6) = 0.5;
v(7) = 0.1;
v(8) = 0.5;
for i=1:NeuralN
  y(i) = MaxF(u(i));
end
output = zeros(TestS,8);
dout   = zeros(TestS,4);
for i=1:TestS
   output(i,:) = Matsuoka_CPG();
   dout(i,1) = output(i,1) - output(i,2);
```

```
    dout(i,2) = output(i,3) - output(i,4);
    dout(i,3) = output(i,5) - output(i,6);
    dout(i,4) = output(i,7) - output(i,8);
end
dout(1:2500,:) = 0;  % remove initial spike in actuation
dout = [Time', dout]; % puts simulation time and output in matrix for simulink to read
```

B.2 Matsuoka CPG Algorithm function

```
function CPG = Matsuoka_CPG()
global w u v y tau dtau Ts NeuralN
Uini = .5; % max peak to peak amplitude
beta = 2.5;
%Mutal inhibition network
sum(1) = -1.5 * y(2) -  w(1) * y(3)  - w(2)  * y(5) - w(3)  * y(7);
sum(2) = -1.5 * y(1) -  w(4) * y(4)  - w(5)  * y(6) - w(6)  * y(8);
sum(3) = -1.5 * y(4) -  w(7) * y(1)  - w(8)  * y(5) - w(9)  * y(7);
sum(4) = -1.5 * y(3) -  w(10) * y(2) - w(11) * y(6) - w(12) * y(8);
sum(5) = -1.5 * y(6) -  w(13) * y(1) - w(14) * y(3) - w(15) * y(7);
sum(6) = -1.5 * y(5) -  w(16) * y(2) - w(17) * y(4) - w(18) * y(8);
sum(7) = -1.5 * y(8) -  w(19) * y(1) -  w(20) * y(3) -  w(21) * y(5);
sum(8) = -1.5 * y(7) -  w(22) * y(2) -  w(23) * y(4) -  w(24) * y(6);
for i = 1:NeuralN
    du(i) = (-u(i) + sum(i) + Uini - beta * v(i)) / tau;
    dv(i) = (-v(i) + y(i)) / dtau;
end
for i = 1:NeuralN
    u(i) = u(i) + du(i) * Ts;
    v(i) = v(i) + dv(i) * Ts;
    y(i) = MaxF(u(i));
end
CPG = y;
end
```

B.3 MaxF function

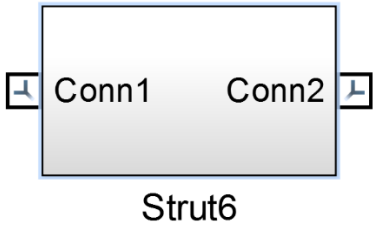
```
function num = MaxF(n)
    if(n > 0)
        num = n;
    else
        num = 0;
    end
end
```

Appendix C. Simulink and SimMechanics

The following sections show the Simulink and SimMechanics blocks used to create the dynamic simulation.

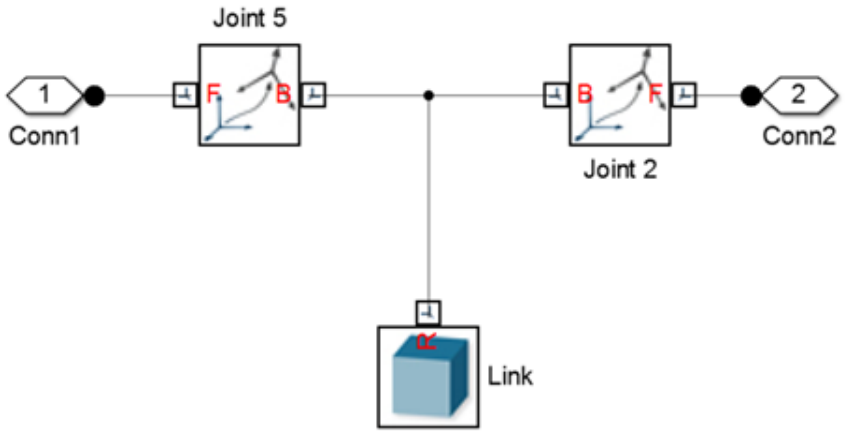
C.1 Strut and sphere

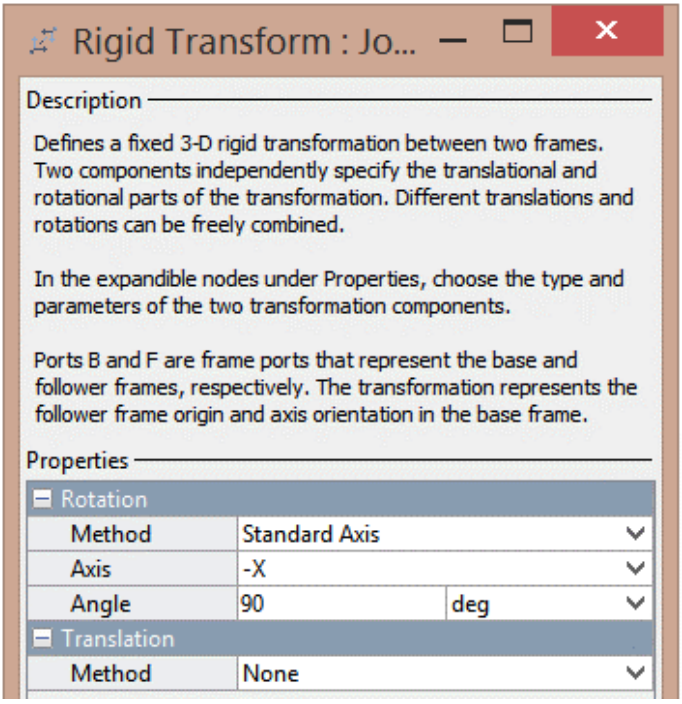
Each compression member, or strut, and sphere, is defined using the Simulink subsystem below. The terminals Conn1 and Conn2 are the connection nodes at either end of the strut, or in the case of spheres, are the same point at the center of the sphere.

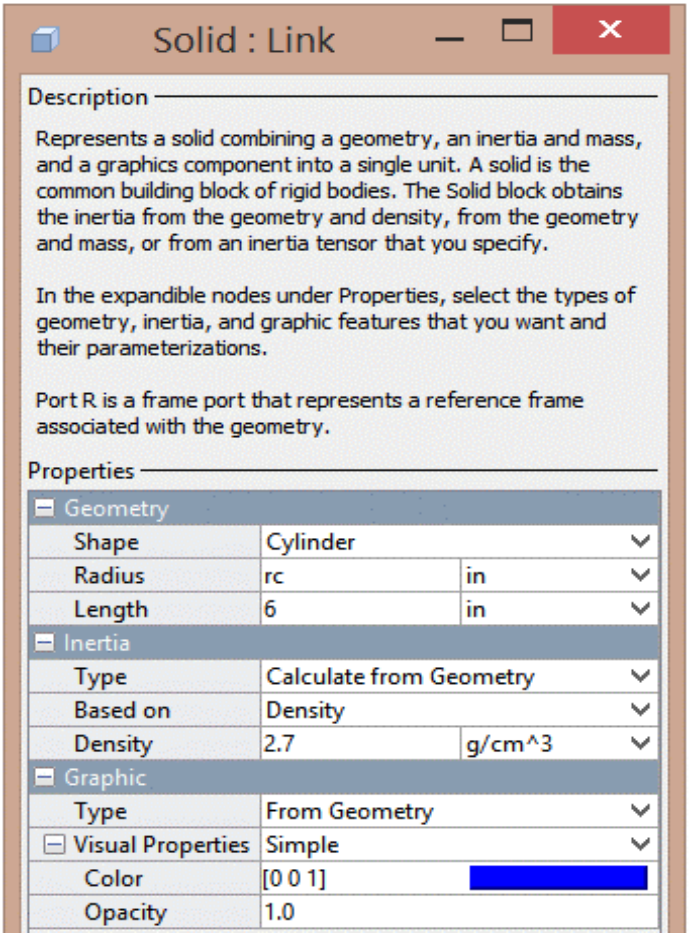


C.1.1 Strut and Sphere detail

Inside the Simulink subsystem block are the following SimMechanics blocks. Joint 5 and Joint 2 are SimMechanics “Rigid Transform” blocks and are used in struts to define the connection nodes. The block labeled link is a SimMechanics “Solid” block that defines the size, shape, inertia, and graphic properties of the struts. For spheres the Joint blocks are ommitted.

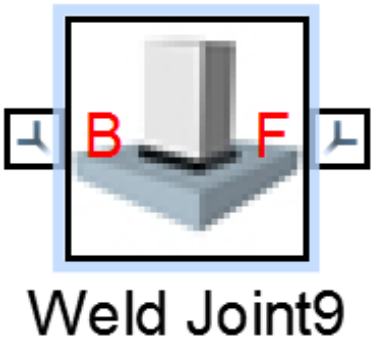


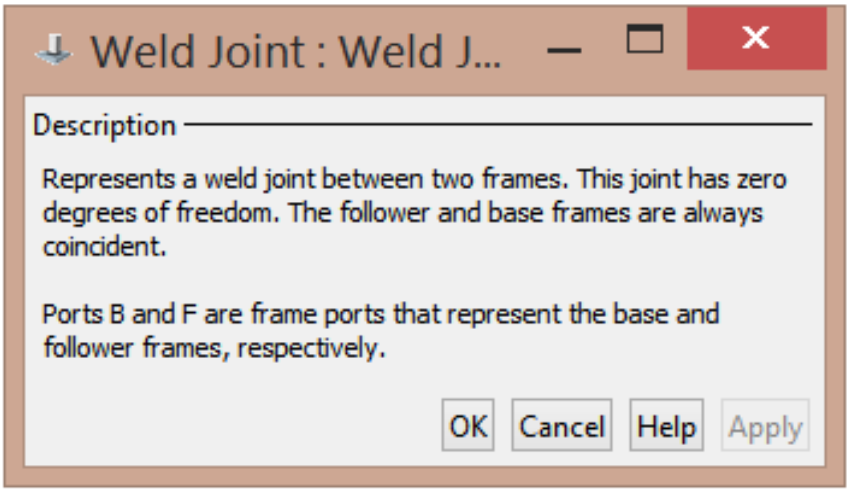

Rigid Transform : Jo...
Description
Defines a fixed 3-D rigid transformation between two frames. Two components independently specify the translational and rotational parts of the transformation. Different translations and rotations can be freely combined.
In the expandible nodes under Properties, choose the type and parameters of the two transformation components.
Ports B and F are frame ports that represent the base and follower frames, respectively. The transformation represents the follower frame origin and axis orientation in the base frame.
Properties
Rotation
Method
Standard Axis
Axis
-X
Angle
90
deg
Translation
Method
None


Solid : Link
Description
Represents a solid combining a geometry, an inertia and mass, and a graphics component into a single unit. A solid is the common building block of rigid bodies. The Solid block obtains the inertia from the geometry and density, from the geometry and mass, or from an inertia tensor that you specify.
In the expandible nodes under Properties, select the types of geometry, inertia, and graphic features that you want and their parameterizations.
Port R is a frame port that represents a reference frame associated with the geometry.
Properties
Geometry
Shape
Cylinder
Radius
rc
in
Length
6
in
Inertia
Type
Calculate from Geometry
Based on
Density
Density
2.7
g/cm^3
Graphic
Type
From Geometry
Visual Properties
Simple
Color
[0 0 1]
Opacity
1.0

C.2 Weld Joint

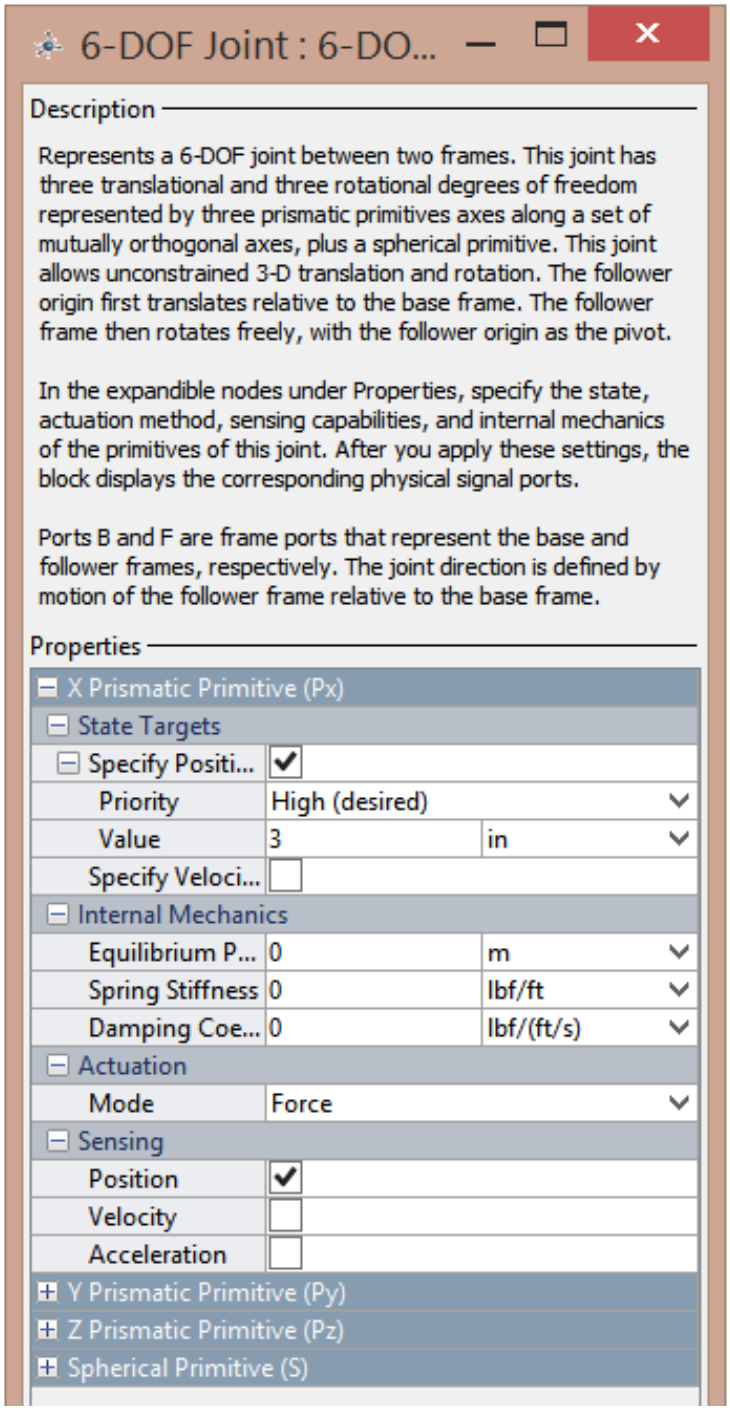
Struts and spheres are connected using SimMechanics blocks called “Weld Joints.” These blocks rigidly connect the nodes connected at the terminals in their current orientation.





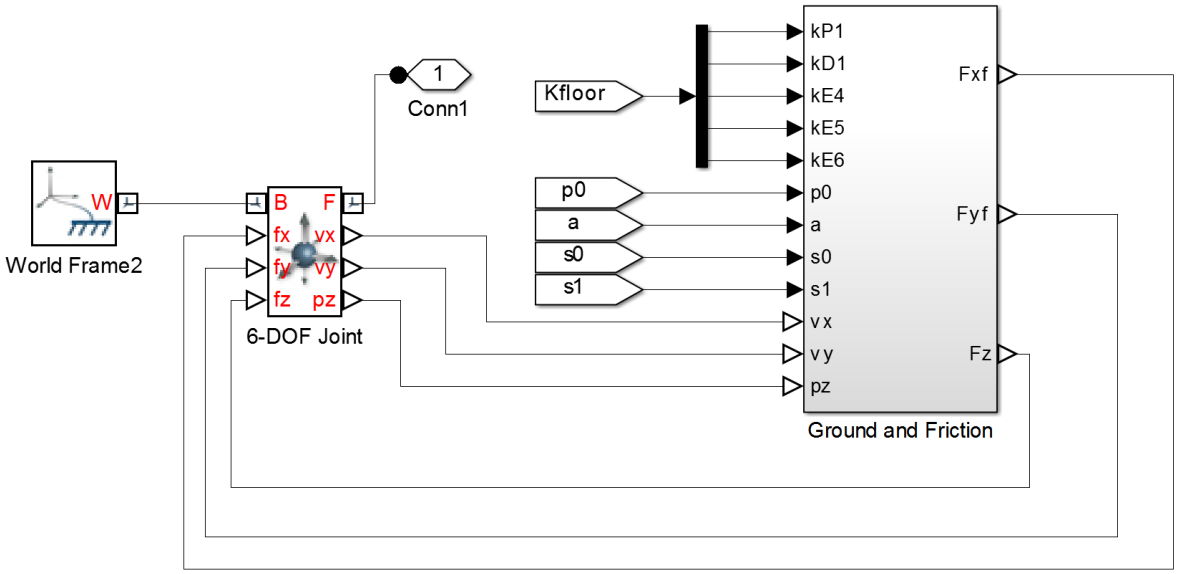
C.3 Six Degree of Freedom Joint

The SimMechanics block "6-DOF Joint" is used to simulate the ground, friction, and tension member models.



C.4 Ground and friction model

The following is the ground and friction model implimentation. The model references the absolute position of the node connected to Conn1 using SimMechanics block “6-DOF Joint” and connected “World Frame.” Data from the “6-DOF Joint” and defining parameters are passed to the subsystem block labeled Ground and Friction which contains both the friction and ground force models. The Ground and Friction subsystem block computes the frictional and ground forces and feeds them back into the “6-DOF Joint” via force actuation terminals.



C.5 Tension member model

The following is an image of the tension member model implimentation. Conn2 and Conn1 are the nodes between which the tension member is connected. Data is passed into the spring model subsystem by a "6-DOF Joint," the defining parameters ksi (an inner tendon denoted by the i in ksi) and damping constant cs, and the CPG actuation signal as denoted by the terminal labeled act on the subsystem block. The spring model substystem calculates forces in the x, y, and z directions to be applied to the joint and passes those values into force actuation terminals on the "6-DOF Joint.