Evolving Soft Robots with Vibration Based Movement

Andrew Danise

March 24, 2014

Abstract

Creating effective designs for soft robots is extremely difficult due to the large number of different possiblities for shape, material properties, and movement mechanisms. Due to the lack of methods to design soft robots, previous research has used evolutionary algorithms to tackle this problem of overwhelming options. A popular technique is to use generative encodings to create designs using evolutionary algorithms because of their modularity and ability to induce large scale coordinated change. The main drawback of generative encodings is that it is difficult to know where along the ontogenic trajectory resides the phenotype with the highest fitness. The two main approaches for addressing this issue are static and scaled developmental timings. In order to compare the effectiveness of each of these two approaches, I have implemented a framework capable of evolving soft robot designs that utilize vibration as a movement mechanism.

Contents

List of Figures

1 Introduction

Most people when asked to imagine a robot would conjure a vision of a traditonal rigid body robot, however, now there are new types of robots that are composed of soft, elastic materials. These new soft robots have many advantages over traditional robots due to their flexible, deformable structures. One advantage is that a soft robot can squeeze and deform its shape to access areas that would be impossible for any rigid body robot to reach. The applications of this are endless. For example, these robots could be harnessed for search and rescue missions to slip under piles of rubble and squeeze into small cracks to search for survivors.

Another benefit of soft robots is that since they have no rigid parts, they have nearly unlimited flexibility making them much more mobile than their rigid body counterparts. However, this flexibility is one of the causes of the major drawback of soft robots: design complexity. Researchers still struggle coordinating the limited degrees of freedom of a rigid body robot due to the complexity of the task. So when faced with creating a design that must coordinate the unlimited degrees of freedom of a soft robot in a way that leads to locomotion, researchers have little to no analytical reasoning for choosing particular types of designs over others.

Part of what makes this problem so difficult is that there are so many different options available to generate movement in a soft robot. Just as traditional robots move by turning wheels or moving legs, soft robots have methods of generating movement too such as the oscillation of the materials of the robot [8]. However, the effectiveness of a movement mechanism for a soft robot depends heavily on the shape of the robot. Robot shapes that are effective with one movement mechanism might fail with others and vice versa. This creates a chicken or egg problem of whether the body or the movement mechanism should be designed first. This massive design space makes it very difficult to approach this problem in an analytical fashion.

2 Background and Related Work

Due to the complexity of the problem and without any analytical design approach, researchers have harnessed evolutionary algorithms to generate soft robot designs optimized for locomotion [5, 1, 14, 8].

2.1 Evolutionary Algorithms

Evolutionary algorithms are a method of slowly evolving effective solutions for problems where little analytical inituition exists. An evolutionary algorithm begins with the creation of an initial population of individuals. These individuals can be anything from the shape of soft robot [5, 1, 14] to the design of a NASA satellite antenna sent into outer space [9]. This initial population of individuals is the starting point for evolution in the algorithm. After the initial population has been established, the main phase of the evolutionary algorithm can begin. The first part of this main phase is to evaluate each individual in the population and assign them a fitness level based on a set of pre-determined criteria. This process of evaluation frequently occurs in simulation. For example, in the case of soft robot designs, many researchers correlate the fitness of a soft robot with its locomotive prowess. To evaluate locomotive prowess, these researchers create the soft robot designs of each individual in the population inside physics engines, such as Bullet Physics [6] and NVidia PhysX [13], to determine how well they are able to move. This technique of using physics engines was also used to evolve tensegrity robots capable of locomotion [15]. The next part in the evolutionary algorithm is to cull the individuals with the lowest fitnesses from the population. This helps ensure that time is not wasted trying to evolve designs that are inferior to other designs. In the final step, new individuals are created and added to the population to replace those that have been culled. Afterwards, the main phase of the evolutionary algorithm is repeated. Over the course of many repetitions, the designs in the population will hopefully evolve to have higher levels of fitness.

2.1.1 Coevolution

An interesting form of evolutionary algorithms allows for the coevolution of two different features. In a standard evolutionary algorithm, such as the one described above, only one feature of an individual is evolved. For example, a standard evolutionary algorithm could evolve either the shape or the movement mechanism of a soft robot. This limits the number of possible designs since evolution of either the shape or the movement mechanism means that the other feature has been predetermined. For this reason, researchers have leveraged coevolution in an attempt to overcome the chicken or egg problem of whether to create the morphology or the movement mechanism of a soft robot first. Using coevolution, the morphology and the

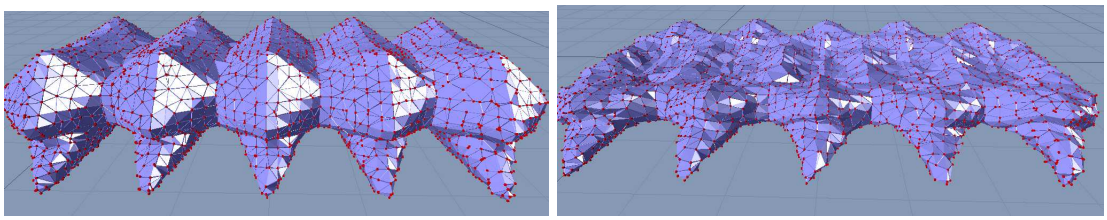

Figure 1: Illustration of the effect that material properties can have on a soft robot. The left design has high material stretching stiffness while the right design has low stretching stiffness [8].

control mechanism can be evolved together. Pollack et al. [11], used the technique of coevolution to develop the controller software of traditional rigid robots simultaneously with the morphology. Joachimczak and Wrbel [6] utilized coevolution to to design the shape and control of a swimming soft robot. Knox and Rieffel [8] coevolved the material properties of a soft robot design with the movement mechanism of the robot. The researchers accomplished this by periodically switching the feature being evolved in the simulation after a certain number of evolution iterations. For example, the evolution would focus on adapting the material properties and then switch to adapting the movement mechanism. Figure 1 illustrates the effect that different material properties can have on the shape of the robot. This figure demonstrates why coevolution is necessary since the movement mechanism for a design with high material stretching stiffness will be different than one with low stretching stiffness. The coevolution used in this paper allows for the movement mechanism to take into account the material properties and vice versa.

2.2 Types of Encodings

In an evolutionary algorithm, each individual corresponds to a physical representation that is evaluated based on the pre-determined fitness criteria. This physical representation is known as the phenotype of the individual. The actual design of the individual that is used to create its physical representation is known as the genotype of the individual. Encodings are ways to specify the genotypes of the individuals in the population. Two main types of encodings are direct encodings and generative encodings.

2.2.1 Direct Encodings

In a direct encoding, the genotype is the same as the phenotype. In other words, the genotype would be the actual physical representation of the individual. For example, Plavcin and Petrovic [10] created a direct representation for the creation of wind turbines. In their representation, the genotype contains the vertices of a triangle mesh that forms the wind turbine. This genotype is a direct mapping to what the actual wind turbine design will be. The benefits of an approach such as this are that it is simple and can deliver promising results. However, the main drawback of direct encodings is that it is difficult to create widespread coordinated change. For example, to represent a four-legged table in a direct encoding, each of the four legs would need to be represented seperately. This would make it difficult to increase the length of all the legs of a table since it would require changes to each of the legs in the representation.

2.2.2 Generative Encodings

In a generative encoding, the genotype is a set of the design changes or rules that can be applied repeatedly to create a phenotype for the individual. When used in an evolutionary algorithm, the design rules that can build the phenotype evolve and produce entirely new families of designs that all have common features. Clune and Lipson [2] used generative encodings to create interesting three dimensional objects. The objects seen in Figure 2 were created using a type of generative encoding called a Compositional Pattern Producing Network (CPPN). A CPPN is a directed graph where each node contains a mathematical function. Different mathematical functions have different effects on the evolved design. For example, a sine function produces repetition and a gaussian function produces symmetry. This example highlights some of the key strengths of generative encodings: repeatability and symmetry. Since the genotype is a set of rules for creating the phenotype it is possible to manipulate the rules to form repeated characteristics and symmetry in the physical representation of an individual.

Lohn et al. [9] used a generative encoding to evolve the design of an antenna, shown in Figure 3, that went on to be used on a NASA satellite. Their encoding consisted of a list of actions that could be used to evolve an antenna. These actions included the ability to increase the length of the antenna in the current direction and actions to rotate the current direction of the antenna around the x, y, and z plane. A ruleset

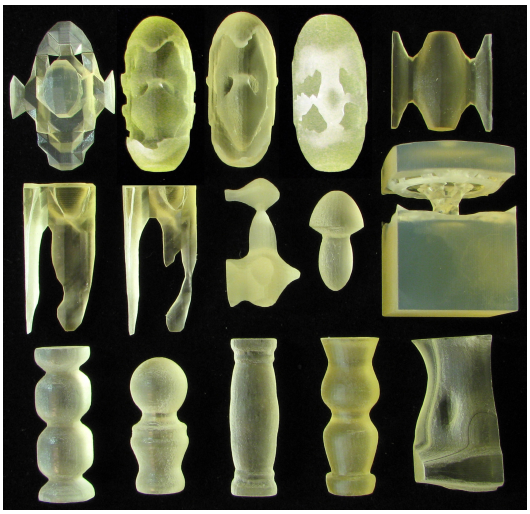

Figure 2: Objects evolved using a generative encoding from [2].

can be created from this encoding that when iterated over yields an antenna design. An important aspect of this example is that while the created satellite has four antenna, the generative encoding used to create the design only encoded one antenna and simply repeated the same design four times. This reinforces the benefits of repeatability when designing objects.

Later attempts have been made to utilize generative encodings to develop soft robot designs. Rieffel and Smith [14] created a generative encoding based around operations on tetrahedral meshes to evolve soft robot designs. Cheney et al. [1] utilized the CPPN-NEAT generative encoding to evolve multi-material soft robots. All of these examples help to highlight the benefits of generative encodings.

However, the one major drawback of generative encodings stems from the fact that it is only evolving a set of rules for producing a design and not the actual design itself. Since the phenotype is determined from repeatedly applying the design rules, it is possible to end up with many different designs with different levels of fitness from a single genotype. This landscape of possible phenotypes for a genotype is referred to as the ontogenic trajectory [4]. Therefore, the main drawback of generative encodings is that is difficult to determine where to stop along the ontogenic trajectory in order to produce the phenotype with the best possible fitness. This problem has been dubbed the halting problem [3].

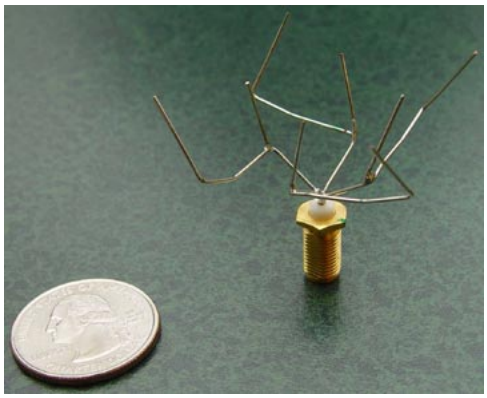

Figure 3: Evolved antenna used on NASA satellite from [9].

2.3 The Halting Problem

The halting problem is a major problem for generative encodings. It means that the final phenotype produced from the genotype could have a level of fitness that is far below the maximum possible level of fitness for that particular genotype. This problem stems from the fact that traditional generative encodings use static developmental timings. A static developmental timing is when the number of times that design rules will be repeated to produce a phenotype is fixed in advance [12]. This opens the door for problems such as those illustrated in Figure 4. The top and the middle graph in the figure, show what happens when the static developmental timings perform well. The last repetition in each case seems to pinpoint the phenotype with the highest possible fitness along the ontogenic trajectory. However, the bottom graph shows the major problem with static developmental timings. In the graph, the last repetition of the generative encoding created a phenotype that had much lower fitness than was possible along the ontogenic trajectory. This means that the phenotype for that genotype has much lower fitness than a phenotype that could be chosen with a shorter or longer static developmental timing.

One alternative to static developmental timings that could help address the halting problem is scaled developmental timings. With scaled developmental timings, the number of repetitions to perform on the design rules is no longer a fixed value. Instead, now the number of repetitions to perform steadily "rachet" up to higher levels [12]. However, an increase in the number of repetitions is only allowed after the best phenotype, in terms of fitness, prior to the last increase is surpassed by the best phenotype after the last

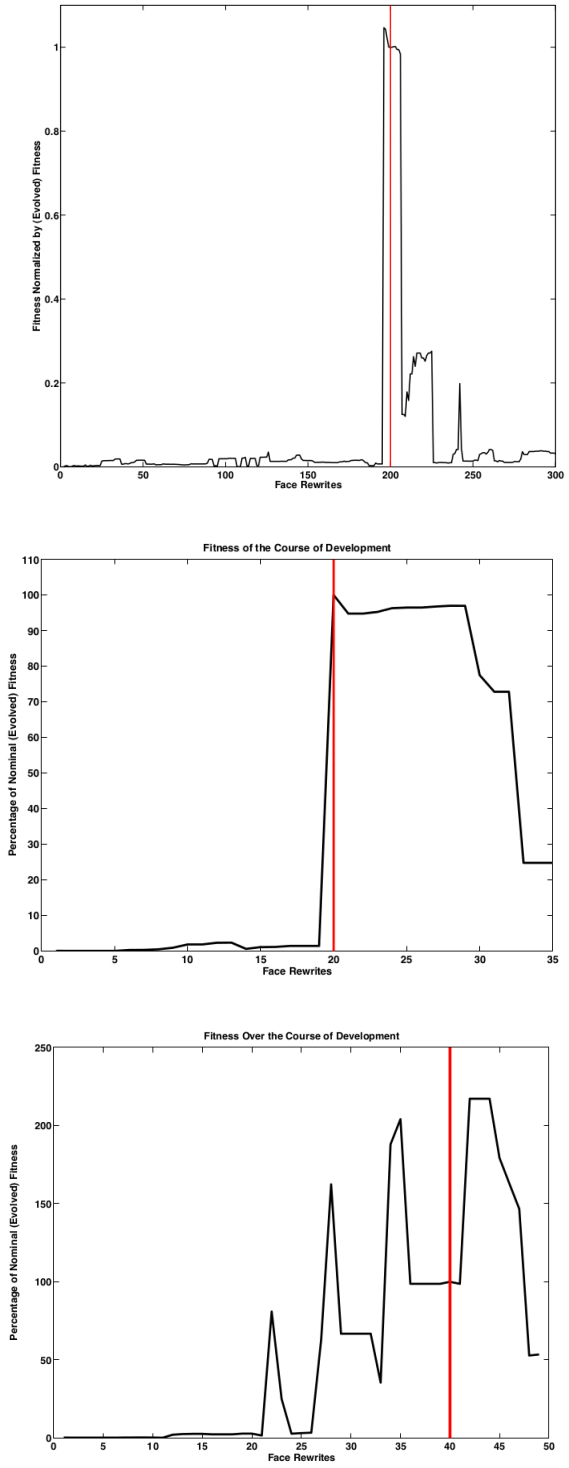

Figure 4: The black line in each graph represents the fitnesses of all phenotypes along the ontogenic tragectory. The red line represents the last iteration of the generative encoding. The intersection of the black and red lines is the fitness of the final phenotype. From [12].

increase. This is extremely useful because as the number of repetitions increase, the phenotypes could become worse. This prevents useless increases in the number of repetitions. Scaled developmental timings offer many benefits over static developmental timings such as increased computational efficiency and reduced design complexity of the final phenotypes. However, they have not yet been shown to eclipse static developmental timings in terms of final phenotype fitness.

3 Experiment Design

My main research goal is to evaluate how much better scaled developmental timings are at addressing this halting problem than static developmental timings. To accomplish this goal, I planned to evolve soft robots optimized for locomotion from a generative encoding using both static and scaled developmental timings and comparing the results. In order to be able to perform this evaluation, I first had to create a framework that was capable of evolving soft robot designs. However, creating such a framework is a major task that has was riddled with many challenges.

3.1 Choosing a Generative Encoding

The first challenge that I had to address was creating a generative encoding that was capable of evolving a soft robot. Instead of building a completely new generative encoding from scratch, I decided to utilized the generative encoding developed by Rieffel and Smith [14]. I chose to use this grammar because it has proven effective at evolving soft robot designs and I did not want to waste precious time redoing what has already been done. The principles of this generative encoding are illustrated in Figure 5. The encoding relies on performing operations that correspond to the label of an unblocked face. The three operations, which are detailed in the figure, include subdividing a face (and the underlying tetrahedron), relabeling a face, or growing new a tetrahedron onto the face. This encoding produces the genotypes for soft robot designs composed of tetrahedra meshes by evolving a series of rules based on these operations. Then by iterating over the ruleset, a phenotype for the design can be created. An example ruleset is shown in Figure 6. In this example ruleset, in every repetition of the encoding, the next unblocked face would apply the rule that corresponds to its label. For example, a face labeled A would apply the grow rule resulting in a new

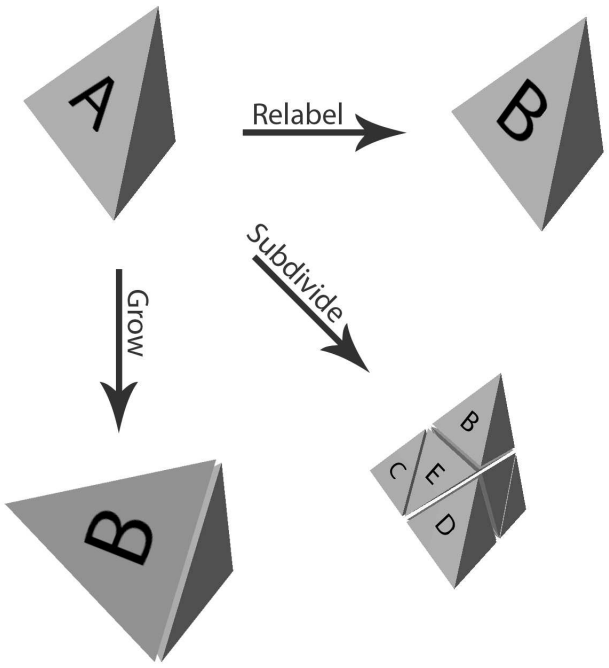

Figure 5: Tetrahedral face encoding grammar to design soft robots. Picture from [14].

tetrahedron with the label D, B, and F being placed on top of the face. Figure 7 shows the process in action as a small tetrahedral mesh evolves into an increasingly complex design.

3.2 Implementation In Bullet Physics Engine

Another challenge that I had to overcome was the decision of which physics engine to use. The physics engine plays an integral role in the evolutionary algorithm as it is required to simulate the phenotypes constructed from the generative encodings. This is the most critical part of the entire evolutionary algorithm because this is where the phenotypes are assigned their fitness. Due to its importance to the entire project, the selection of which physics engine to use was the first decisions I made. There are quite a few physics engines available, however, the two main ones are NVidia PhysX and the Bullet physics engine. I ended up choosing the Bullet physics engine primarily due to its vastly better support of soft body dynamics. This is important since my plan requires physics simulations that test the fitness of the evolved soft robot designs. However,

$Initialize[AGCA]$			
A	$\rightarrow$	grow	$\{DBF\}$
B	$\rightarrow$	grow	$\{ADF\}$
C	$\rightarrow$	grow	$\{EDF\}$
D	$\rightarrow$	relabel	(D)
E	$\rightarrow$	grow	$\{DCF\}$
F	$\rightarrow$	divide	$[DDDG]$
G	$\rightarrow$	grow	$\{DDG\}$

Figure 6: Example ruleset. Picture from [12].

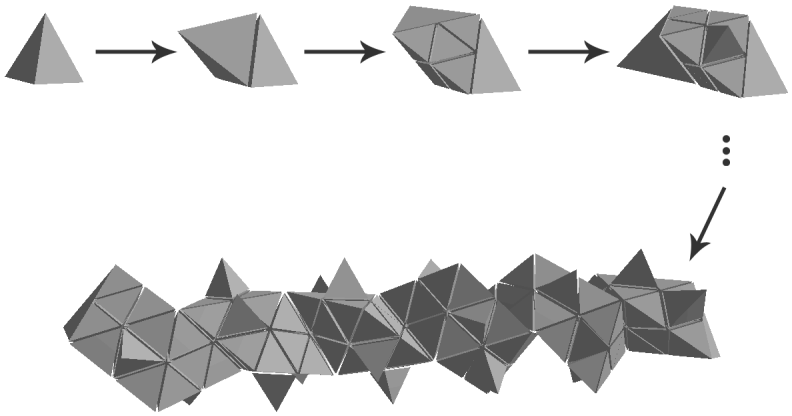

Figure 7: Demonstration of evolution. Picture from [14].

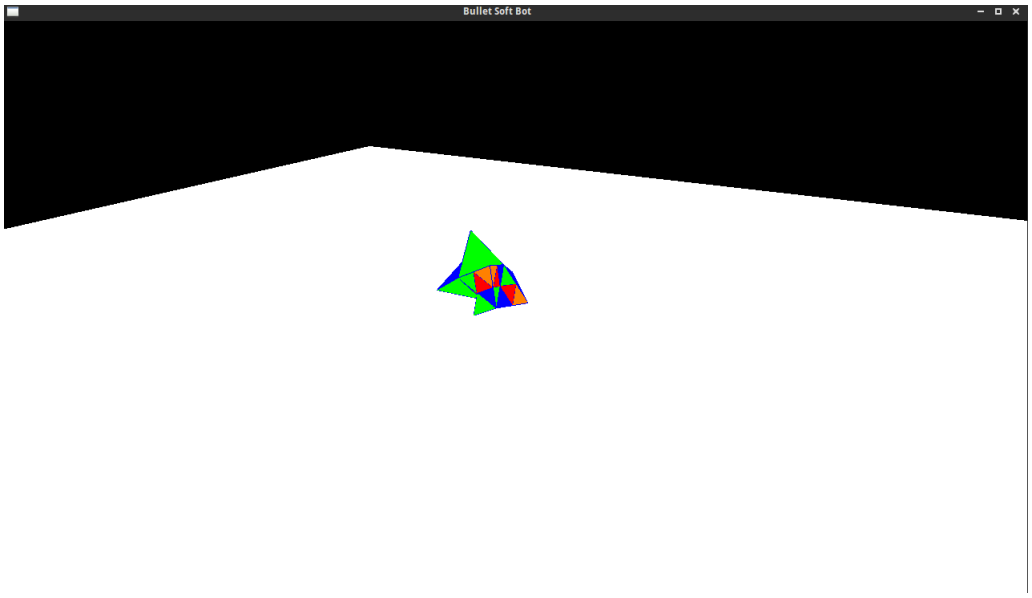

Figure 8: A sample of a tetrahedral mesh after numerous grammar iterations. The colors of the exposed faces correspond to the label of the face. Green = A, Red = B, Blue = C, and Orange = D.

Bullet also has the added benefit of being open source software with a plethora of documentation. This was important because I needed to quickly learn how to use the physics engine in order to implement the framework. Figure 8 shows an example soft robot design I created using the generative encoding simulated inside the Bullet physics engine.

3.3 Movement Mechanisms

The most difficult challenge that I faced was what kind of movement mechanism to use. The movement mechanism is a crucial part of the design of soft robots. The chosen movement mechanism will greatly impact the evolved design of the robot because the same evolutionary adaptation that may positively impact fitness for one movement mechanism, may have zero or even negative impact on fitness when a different movement mechanism is used. There are many different options of features to use as the movement mechanism for soft robots. For example, in [1], the movement mechanism was based on the differing properties of the materials used to create the soft body robots. The resulting designs evolved to utilize the materials to cause locomotion. When the experimenters added new materials, the increase in options led to different

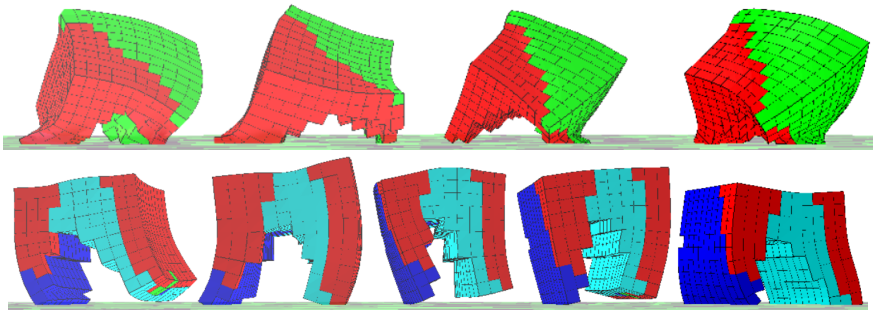

Figure 9: Two examples of soft robots created with a material based movement mechanisms. The top design uses an inching locomotion strategy while the bottom design uses a galloping behavior [1].

designs and locomotion strategies as shown in Figure 9. In this example, the different colors signify different materials that each have different properties. The dark blue color represents a stiff material, the light blue is a soft material, and the red and green materials each undergo periodic equal and opposite volume actuations. The difference of behaviors caused by adding more materials demonstrate the importance of choosing an appropriate control mechanism. This example demonstrates the effect that different movement mechanisms have on the evolution of designs.

3.3.1 Selecting a Movement Mechanism

While there are many conceivable movement mechanisms that could be utilized for soft robot design, in reality the possible movement mechanisms are limited to the capabilities of the Bullet physics engine since the movement mechanism has to be simulated. After exploring the capabilities of Bullet, I decided to use vibration as a movement mechanism. The inspiration to use vibration came from recent research into the movement of tensegrity robots via vibration [7]. The main benefit of using vibration as a movement mechanism is that real life vibration motors exist, shown in figure 10 that could be used to produce a real world soft robot capable of locomotion.

3.3.2 Implementation

The next challenge that needed to be addressed was to implement vibration in Bullet. The design is composed of two cylinders: one cylinder that is permanently attached between two vertices of the original tetrahedron

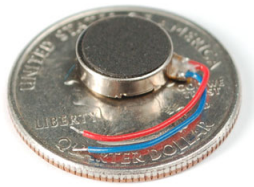

Figure 10: A vibration motor that could be attached to a real life soft robot to generate locomotion.

of the soft robot and another cylinder that rotates around the first cylinder. For this design to work, the cylinders are set to ignore collisions with the soft robot. This design is illustrated in figure 11. In the figure, the red cylinder is attached to the original tetrahedron and the pink cylinder rotates around the red cylinder. As detailed in bottom half of the figure, when the generative encoding expands the soft robot design, the cylinder remain attached between the same two vertices.

3.4 Designing a Fitness Function

The last challenge that I had to address was how to create the fitness function for the simulation. The fitness function is used by the simulation to assign the fitness of every individual that gets simulated in the physics engine. This challenge is particularly important to an evolutionary algorithm because the fitness function determines if a design is good and should be kept or if a design is bad and needs to be discarded. As a result, the fitness function needs to be created so that it assigns higher fitness to the individuals that are more proficient at locomotion. The fitness function that I used in my framework is based on the fitness function utilized in [14]. The fitness function I created simply assigns fitness as the absolute value of the distance between the coordinate where the individual begins and the coordinate where the individual ends up after a specified time period of utilizing the movement mechanism. Since I am trying to generate designs that are optimized for locomotion, using the distance that the simulated soft robot design is able to travel seemed like an effective method to differentiate a good design from a bad design. One important note to take into account is that distance was measured by how far the design was able to move in the XZ plane. In other words, when measuring the distance between the start and the end point, the y coordinates where set to zero. This was done because slight changes in the y coordinate do not effect how far the design was able

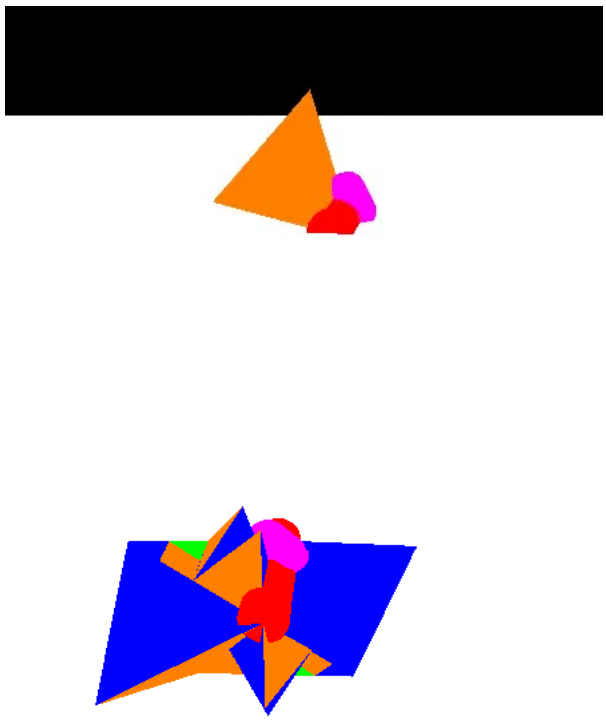

Figure 11: Two examples of the simulated softbots with the vibration movement mechanism. The design on top is the initial tetrahedron and the design on the bottom is a design after several iterations.

to locomote and, as a result, could have created some bias in the fitnesses of the individuals.

4 Evaluation

My main method for evaluating my simulation framework was to try to evolve effective soft robot designs that are capable of locomotion. Figure 12 shows a some of the designs that were evolved using my framework to move via vibration. Another method I used to evaluate the framework was to produce the generated designs in real life using Union's newly acquired 3D printer. Figure 13 displays the 3D printed design. This was a major accomplishment since the overall goal of the simulation framework is to generate designs of soft robots. The successful 3D printing of the designs ensures that the simulation is working properly and is producing designs that can be translated from the simulation into the real world.

After running the simulation a number of times, it quickly became clear that it requires a great amount of computational power. For example, to run the simulation with a population of ten individuals for five generations required 43 minutes. This presents a problem because evolutionary algorithms may run with a population of 400 individuals for 1000 generations. Clearly, running using this simulation would take a massive amount of time so something needs to be done to help speed up the process.

5 Future Work

There are still many more steps that I plan to take to further this research. For example, since the process of simulating each of the different generations of an evolutionary algorithm in a physics engine is so incredibly demanding on the resources of the computer running the simulation, I hope to speed up the simulation process. To do this, I plan to run my simulations on the Union College cluster computer. The plan is to have many different simulations running all at once on the cluster to decrease the overall amount of time it takes to create designs. The utilization of the computing resources of Union College should greatly decrease the amount of time needed to run large numbers of simulations and as a result will greatly increase the number of different simulations that can be run. This will be extremely helpful because I will be able to generate a vast amount of designs and collect huge quantities of data.

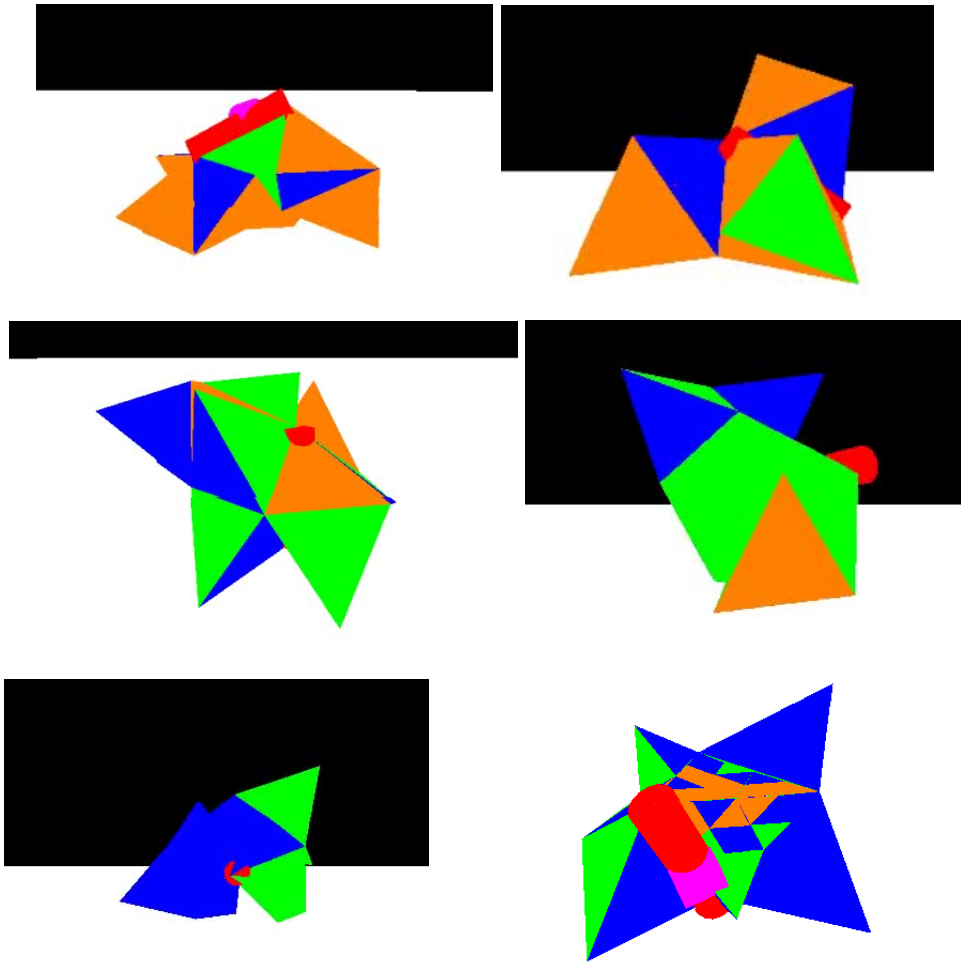

Figure 12: Six different designs evolved using the simulation framework.

Figure 13: A 3D printed robot designed using my simulation framework.

I plan to utilize this data to evaluate how much scaled development timings help to address the halting problem compared with static developmental timings. This will involve analyzing data collected on the phenotypes that each method chooses for many different ontogenic trajectories. Also, data about which phenotype in the ontogenic trajectory had the highest fitness could be analyzed as well. By comparing how much the fitness of the phenotype selected by each method deviates from the best possible phenotype, it is possible to gauge how much each method is limiting the halting problem. I also plan to compare the final fitnesses that each method arrives at in an attempt to confirm the results shown in [12]. If scaled developmental timings still perform worse than static developmental timings in this experiment, in terms of the final fitness selected, I plan to attempt to tweak the scaled developmental timings in an attempt to equal or surpass the prowess of static developmental timings. This entire process will involve collecting a large amount of data by running many simulations using the framework that I have developed.

Finally, I hope to utilize my simulation framework to produce many more designs for soft robots that utilize vibration to move. It would be incredible to 3D print these soft robot designs and use real world vibration motors to generate movement in the printed soft robots.

References

[1] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding. *GECCO*, 2013.

[2] Jeff Clune and Hod Lipson. Evolving three-dimensional objects with a generative encoding inspired by developmental biology. *ACM SIGEVOlution*, 5:2–12, 2011.

[3] A. Devert, N. Bredeche, and M. Schoenauer. Robustness and the halting problem for multicellular artificial ontogeny. *Evolutionary Computation, IEEE Transactions on*, 15(3):387–404, 2011.

[4] Stephen Jay Gould. *Ontogeny and Phylogeny*. Harvard University Press, Cambridge, 1977.

[5] Jonathan D. Hiller and Hod Lipson. Evolving amorphous robots. In *Proc. of the Alife XII Conference, Odense*, pages 717–724, 2010.

[6] Micha Joachimczak and Borys Wrbel. Co-evolution of morphology and control of soft-bodied multicellular animats. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, pages 561–568, 2012.

[7] Mark Khazanov, Ben Humphreys, William Keat, and John Rieffel. Exploiting dynamical complexity in a physical tensegrity robot to achieve locomotion. *ECAL*, 12, 2013.

[8] Davis Knox and John Rieffel. Scalable co-evolution of soft robot properties and gaits. In *Proceedings of the Eleventh European Conference on the Synthesis and Simulation of Living Systems*, 2011.

[9] J.D. Lohn, G.S. Hornby, and D.S. Linden. An evolved antenna for deployment on nasas space technology 5 mission. In *Genetic Programming Theory Practice 2004 Workshop*, 2004.

[10] Juraj Plavcan and Pavel Petrovic. Direct and indirect representations for evolutionary design of objects. In Nicolas Monmarch, El-Ghazali Talbi, Pierre Collet, Marc Schoenauer, and Evelyne Lutton, editors, *Artificial Evolution*, volume 4926 of *Lecture Notes in Computer Science*, pages 160–171. Springer Berlin Heidelberg, 2008.

[11] Jordan Pollack, Hod Lipson, Pablo Funes, Sevan Ficici, and Greg Hornby. Coevolutionary robotics. In *Evolvable Hardware*, pages 208–216, 1999.

[12] John Rieffel. Heterochronic scaling of developmental durations in evolved soft robots. In *GECCO*, 2013.

[13] John Rieffel, Frank Saunders, Shilpa Nadimpalli Harvey Zhou, Soha Hassou, and Jason Rife. Evolving soft robotic locomotion in physx. In *GECCO*, 2009.

[14] John Rieffel and Schuyler Smith. A face-encoding grammar for the generation of tetrahedral-mesh soft bodies. In *Proc. of the Alife XII Conference*, 2010.

[15] John Rieffel, Barry Trimmer, and Hod Lipson. Mechanism as mind: What tensegrities and caterpillars can teach us about soft robotics. In *In Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems*, pages 506–512, 2008.