A Scale-Invariant Generative Design Process for 2D Soft Robot Actuators

by

Naudé Thomas Conradie

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Mechatronic) in the Faculty of Engineering at Stellenbosch University

Supervisor: Dr. M. P. Venter

April 2021

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: 2020/11/20 .....

Abstract

A Scale-Invariant Generative Design Process for 2D Soft Robot Actuators

N. T. Conradie

Department of Mechanical and Mechatronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MEng (Mech)

April 2021

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations:

My wife, Anmari Conradie, for her unending love, support and patience throughout this project

My parents and in-laws for their love and assistance

DebMarine Namibia, for sponsoring my studies

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Manufacturing capabilities have greatly increased over the past decades. The advent of three dimensional (3D) printing and other advanced manufacturing technologies have allowed for the production of novel geometries that were previously unattainable. (Buchanan and Gardner, 2019; Luis *et al.*, 2020)

Development of new products and systems may now be limited by current design capabilities of humans. Improving and creating new design methods may lead to innovative designs not yet seen before. Design methods involving collaboration with computers executing generative design procedures allows for exploration of complex geometries. (Shea *et al.*, 2005)

The field of soft robotics is particularly suited to creative and novel approaches to the design of components. Soft robotic geometries are often highly complex and free-form. Soft robotic actuators may move in imprecise and non-trivial manners (Whitesides, 2018). Materials used in the construction of soft robotic components may behave with non-linear responses (Boyraz *et al.*, 2018). Novel soft robotic actuator designs with new and non-trivial behaviours are actively being created and implemented (Ellis, 2020).

A computationally efficient generative design approach may be used to explore the design space of soft robotics. Hyper-elastic non-linear material models are computationally expensive. Simplifying the model representation or evaluation is thus desirable (Niroomandi *et al.*, 2010). Generative design is a powerful automated approach to design that evaluates the performance of many different potential design solutions. Generative design is useful when performing manual design evaluations becomes tedious or impossible within realistic time constraints (Brose, 1993).

1.2 Aim and Objectives

The aim of this project is to implement a scale-invariant generative design process that results in replicable soft robotic actuator components. A scale-invariant process would allow for an efficient encoding that can be applied to a wide range of resolutions yielding similar performances. A generative design process would entail a computationally focused design process with initial parameters defined by a human user, the design process largely carried out independently by the software pipeline and appropriate designs selected and refined by the user. The design process should be compatible with multiple objective functions for the actuators. The design software should be easily modifiable and adaptable. To this end, the project's objectives are outlined as:

1. Review existing approaches to modelling and designing soft bodies to determine appropriate methods to be investigated
2. Implement a scale-invariant generative design process capable of generating manufacturable soft robotic actuator components fulfilling a given objective function
3. Verify the simulated performance of generated bodies by constructing and examining a physical model

1.3 Scope and Assumptions

Some assumptions are made to reasonably limit the scope of the project:

1. The design approach will be generalised as much as possible to allow for easy modification of physical dimensions, material properties and objective functions
2. The design approach will be limited to the consideration of soft materials
3. The design space will be limited to 2D solutions
4. Existing FEM software will be used for material modelling

1.4 Project Overview

Chapter 2 consists of a literature review exploring the field of soft robotics in general, as well as focusing on soft robotic actuators and design approaches. The literature review also discusses generative design approaches including genetic algorithms, Monte Carlo simulations, L-Systems, and CPPNs. A short

discussion on the phenomenon of emergent properties is also included. Chapter 3 encapsulates the materials and methods relevant to this project. The materials used in the simulation and practical testing thereof are discussed. The generative design methods as applied are discussed in detail. The software implementation is elaborated upon. Chapter 4 discusses results obtained from running simulations according to specified parameters. Chapter 5 showcases a physical replication of models generated during the simulations. A conclusion to the project is presented in Chapter 6.

Chapter 2

Literature Review

2.1 Soft Robotics

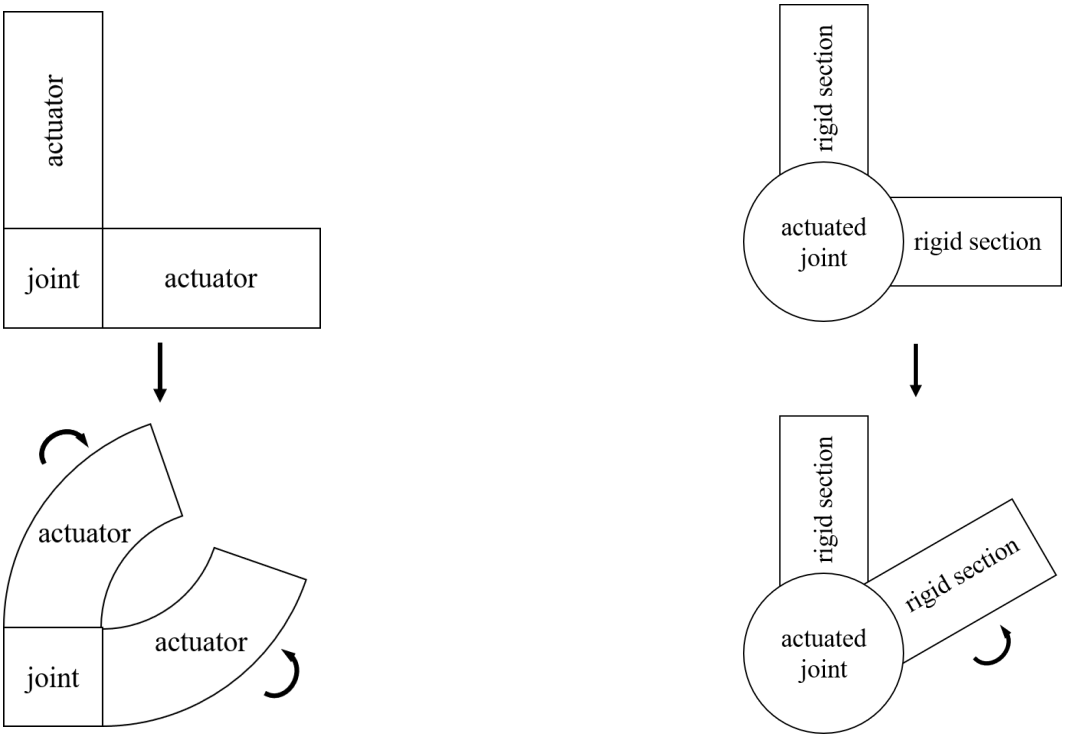
The field of soft robotics comprises robots and robotic actuators constructed from compliant and pliable materials, or exhibiting compliant behaviour (Wang *et al.*, 2015; Ilievski *et al.*, 2011). Soft robots often aim to mimic biological muscles and locomotion. Hyper-elastic materials are often used in the construction of soft robots.

Soft robots generally have fixed joints and locomotive actuators between joints, as opposed to traditional hard robots which usually have locomotive actuated joints connected by rigid sections (Whitesides, 2018), as illustrated in Figure 2.1.

Soft robots offer many advantages over traditional robots. Soft robots are safer in working environments alongside humans. Traditional robots are generally constructed using non-compliant hard, dense, and heavy materials such as steel or aluminium. Traditional robots usually have fixed motion paths and do not allow for much deviation. Humans working alongside traditional robots may easily be injured by these robots if they are struck or trapped by them. Conversely, soft robots are built with compliant and lightweight materials. Soft robots also generally exert lower forces than their traditional counterparts. Soft robots are unlikely to cause injuries to humans. (Whitesides, 2018; Martinez *et al.*, 2013)

Soft robots are more suited to working with fragile and irregular objects such as fruit. In addition to the reasons listed above, soft robots easily adjust to deviations when deforming. (Whitesides, 2018; Martinez *et al.*, 2013)

Combining soft robots' safeness around humans and ability to work with fragile and irregular objects has led to many applications of soft robotics in the biomedical engineering field. Soft direct cardiac compression devices can assist in providing enough blood flow. Soft surgical manipulators and grippers provide safer, compliant and less invasive tools for surgery. Soft rehabilitative devices assist in the diagnosis and treatment of patients. (Hsiao *et al.*, 2019)



(a) An example of soft robot actuation

(b) An example of traditional robot actuation

Figure 2.1: Differences between soft and traditional robot actuaction

2.2 Actuators

Actuators are components that cause controlled motion, generally used in robotics and machinery (Sekhar and Uwizeye, 2012). There are currently a few major types of soft robotic actuators in use (Boyraz *et al.*, 2018).

2.2.1 Soft Actuator Types

Shape Memory Alloys (SMA) are metallic alloys capable of being formed into a specific shape while above an inherent transformation temperature, as well as being formed into another shape below the transformation temperature. When the material is then heated or cooled above or below the transformation temperature, it reforms into those respective shapes. This property exists due to the transition between the martensite phase of the material below the transformation temperature, and the austenite phase above the transformation temperature. SMA actuators are heated by applying a current directly to the material. SMA actuators are small, lightweight, silent, and have a high force-to-weight ratio. When shaped straight, they can exert high forces, but only achieve small displacements relative to their length. When coiled, they can extend more, but exert smaller forces. (Villoslada *et al.*, 2014)

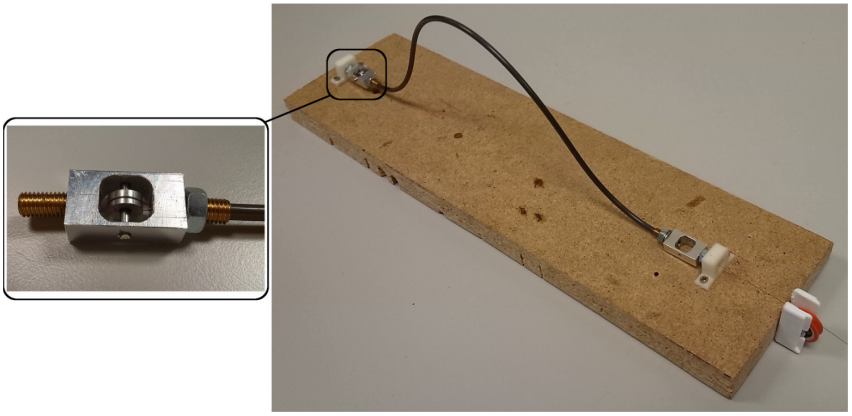


Figure 2.2: A flexible SMA actuator designed by Villoslada *et al.* (2014)

Shape Memory Polymers (SMP) are similar to SMAs, consisting of smart polymers with the same shape memory properties, instead of metallic alloys. The initial shape is determined during the manufacturing process. The transformed shape is obtained by cooling the SMP and shaping it as desired. SMPs use electricity or light as a heat source for transformation (Behl and Lendlein, 2007). They have a high deformation capacity and shape recovery. They are lighter, cheaper and easier to produce than SMAs. They are limited in size due to their low recovery stresses (Huang *et al.*, 2005; Rodriguez *et al.*, 2016; Behl and Lendlein, 2007).

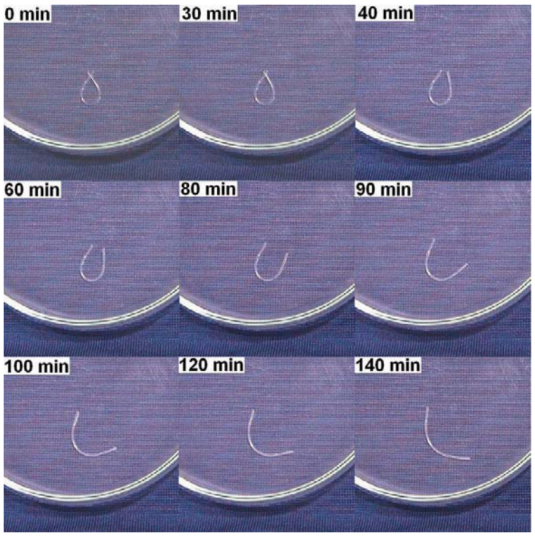


Figure 2.3: SMP water-driven recovery over time (Huang *et al.*, 2005)

Dielectric/Electrically-Actuated Polymers (DEAP) consist of layers of polymers interspersed with conductive material. When the conductive material receives an electrical input, a chemical reaction occurs that causes a change in

volume across the layers. This causes the layers to bend in a predetermined direction. DEAPs are lightweight, silent and use little energy. They are biocompatible and functional in water. They are well-suited to mimicking real muscles. Their reactions under high voltages are not fully understood yet and accurately modelling them is highly complicated. (Mutlu *et al.*, 2014)

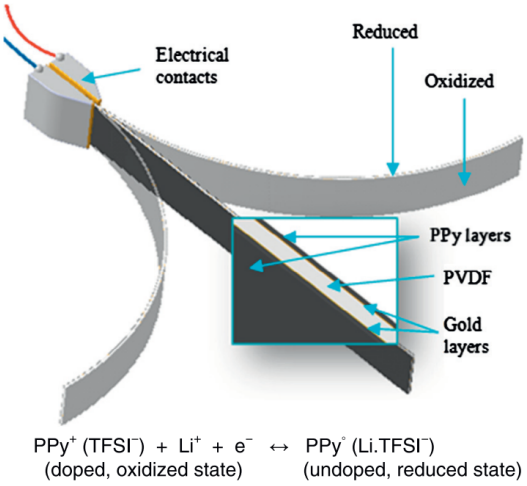


Figure 2.4: EAP actuator structure and operating principles (Mutlu *et al.*, 2014)

Electro-Magnetic Actuators (EMA) make use of magnetic microparticles within a polymer matrix. The particles are manipulated to cause motion by an external magnetic field from an electromagnet. This allows for a wide range of motion by varying the orientation and magnitude of the electromagnetic field. EMAs are small, require low voltages and are efficient. They have quick response times and high dynamic ranges. They are still an emerging technology in the early stages of development. (Do *et al.*, 2018)

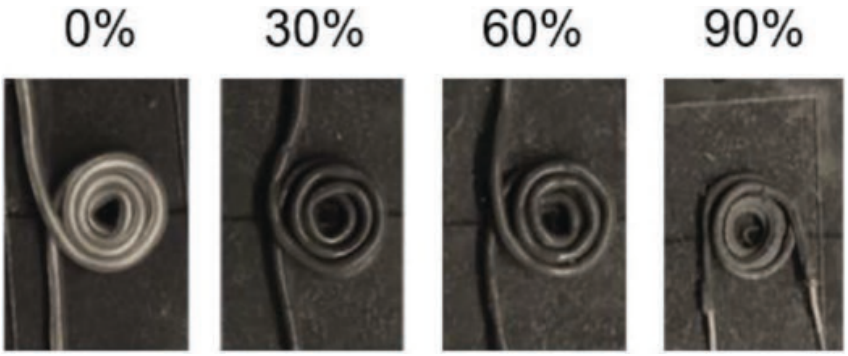


Figure 2.5: EMA coil heating with differing mass ratios of EGaIn and polymer (Do *et al.*, 2018)

Fluid Elastomeric Actuators (FEA) use soft polymeric structures with internal geometry designed for specific types of motion when driven by fluid pressure. Fluid pressure may be obtained from pressurized containers or chemical reactions. They are simple to design, manufacture and control, and are lightweight and usually inexpensive. They are scalable to different sizes and resistant to many types of damage. (Shepherd *et al.*, 2011; Onal *et al.*, 2017)

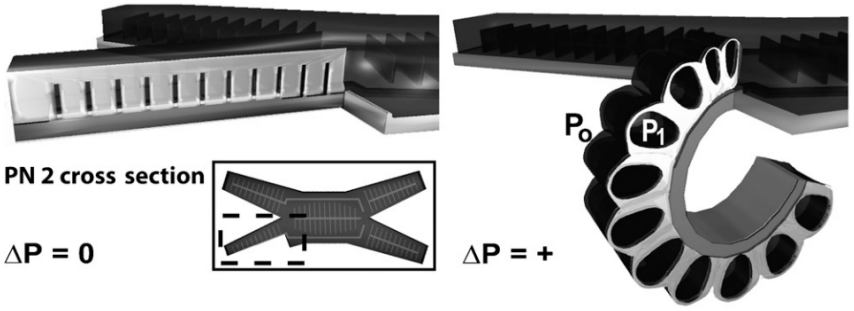


Figure 2.6: Cross section of an FEA at atmospheric and actuated pressures (Shepherd *et al.*, 2011)

2.2.2 Soft Actuator Designs

Some soft robotic actuators implement linear extension, which has many applications. Extending actuators allow for an increase in reach of robotic bodies. Extension may be used to activate levers or switches. Extension may be used to lift objects or bodies. (Martinez *et al.*, 2012)

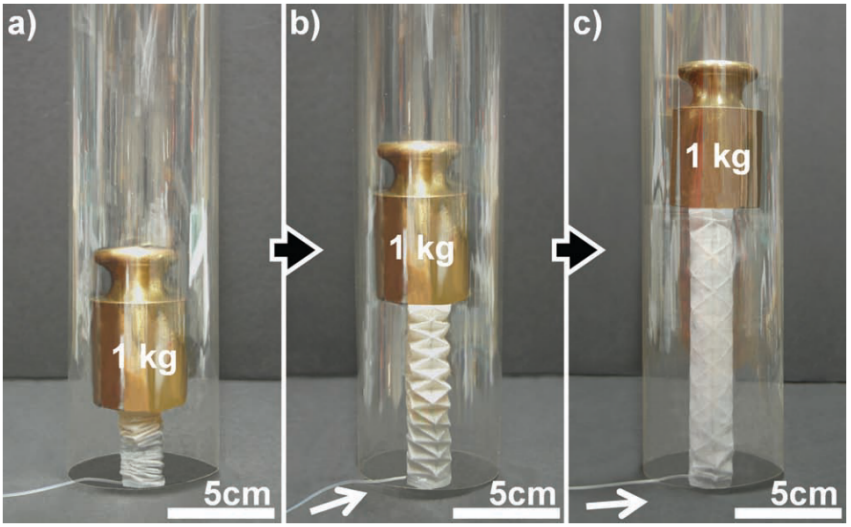


Figure 2.7: Linearly extending soft actuator lifting 1 kg (Martinez *et al.*, 2012)

Torsional extension is a variant of linear extension. Torsionally extending actuators twist as they extend, resulting in an angular difference between the ends of the actuator. In addition to the uses of linearly extending actuators, torsionally extending actuators may be used to twist, turn or screw components or objects. (Yan *et al.*, 2018)

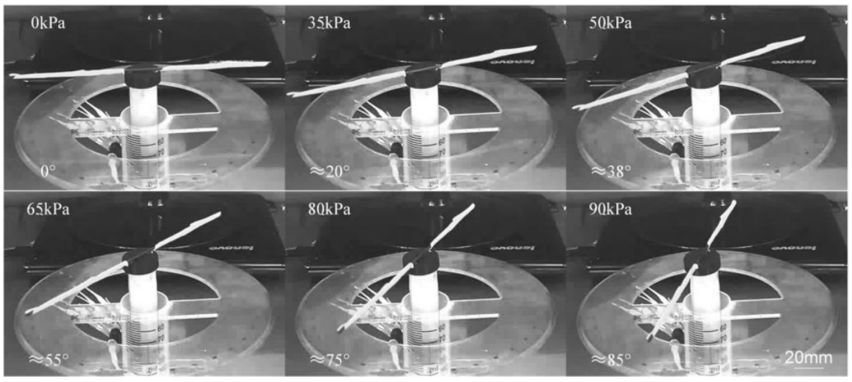


Figure 2.8: Torsionally extending soft actuator displaying rotation (Yan *et al.*, 2018)

FEAs can be built to curl while expanding and straighten out while contracting. This has a range of applications, especially when multiple of these FEAs are used in conjunction with one another. One application is as a gripper, where a number of these FEAs are arranged similarly to a hand or tentacles all curling inward. Grippers are well-suited to picking up and manipulating soft and/or irregularly shaped objects. (Ilievski *et al.*, 2011)

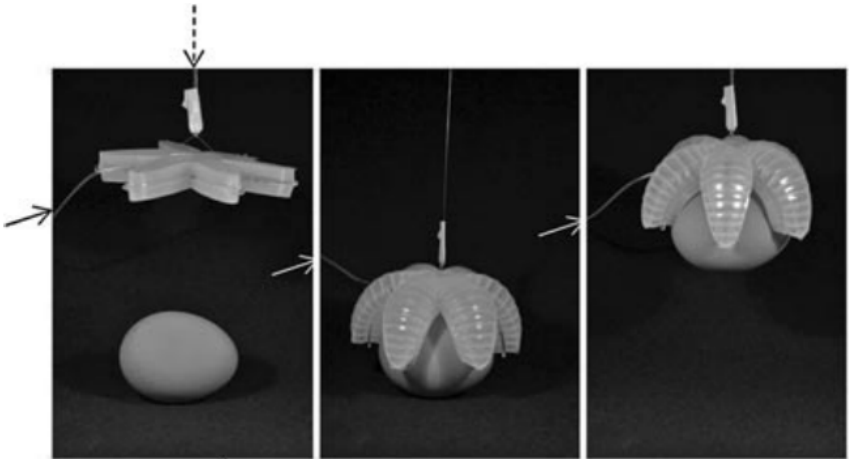


Figure 2.9: Soft robotic gripper picking up an egg (Ilievski *et al.*, 2011)

More complex actuators are manufacturable. A bimodal FEA was developed by Ellis (2020). The actuator has a crimped paper strip acting as a

strain limiter embedded within a stiff layer of Ecoflex 0030. The strain-limiting layer is alongside multiple Mold-Star 15 cells. When pressurised by a linearly increasing single pressure source, the actuator first curls inwards, then back outwards while extending linearly. (Ellis, 2020)

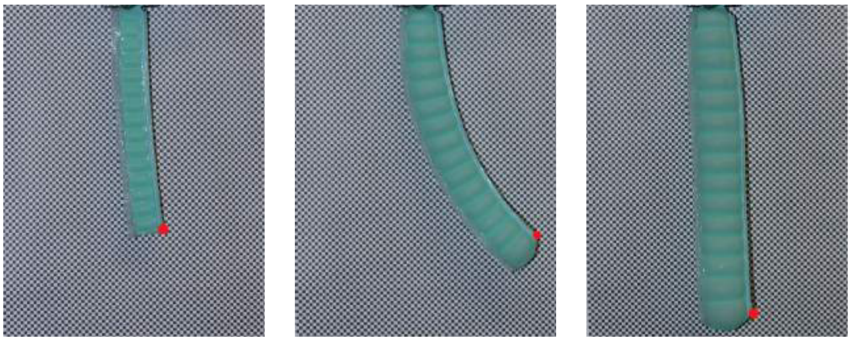


Figure 2.10: Bimodal FEA exhibiting different responses as pressure increases (Ellis, 2020)

2.3 Soft Robot Modelling Approaches

To prevent the necessity of physically constructing every design of a soft body, a computationally efficient and accurate digital model can be constructed.

2.3.1 FEM

The Finite Element Method (FEM) has been used to successfully model soft robotic actuators and components. FEM models made use of the Ogden material model or the Yeoh material model. Models were validated with experimental data. (Elsayed *et al.*, 2014; Runge *et al.*, 2017)

2.3.2 Voxels

Voxels are three dimensional (3D) pixels. Voxels are usually cubical. Realistic physical properties may be applied to voxels in specific software packages, such as the free VoxCAD software. Bodies may be constructed and undergo deformation when constructed from voxels in these software packages. It is relatively simple to construct bodies from voxels. (Cheney *et al.*, 2013, 2015)

Cheney *et al.* (2013) used voxels to model soft bodies in VoxCAD. VoxCAD is free voxel-modelling software. Soft bodies were evolved using CPPN-NEAT. CPPN morphologies were found to appear natural and produce interesting and varying results. Three materials were used for the construction of the soft bodies. The three materials differed in hardness, being compliant, partially compliant and stiff. Compliant and partially compliant voxels acted as the actuators. Soft bodies were tasked either with traversing along a linear

path or squeezing through a tight space. Simulations were run multiple times with different penalty functions. Penalty functions included costs for actuated voxels, voxel connections and the total number of voxels. It was found that with differing penalty functions, different bodies evolved and performed better. Final bodies are 3D printable. They are actuated by placing them within a pressure chamber where the pressure is sinusoidally varied. This limits their practical application. (Cheney *et al.*, 2013, 2015)

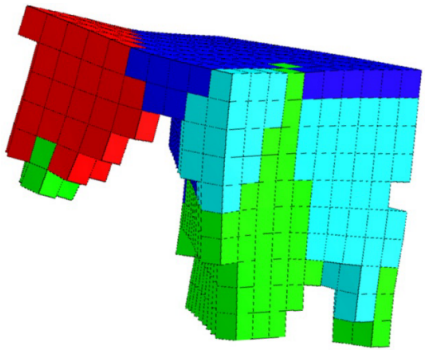


Figure 2.11: Soft robot constructed from voxels (Cheney *et al.*, 2013)

2.3.3 Tetrahedral Meshes

Rieffel *et al.* (2013) et al applied evolutionary algorithms to soft bodies. They used NVidia's PhysX engine to model soft tetrahedral meshes. The physics engine allows for the manipulation of a material's stiffness and damping coefficient. Three properties of the bodies were used to control the evolution. The properties were the body shape, body motion, and material properties. Three different simulations were run. For each one, one of the three properties were fixed, while the other two were allowed to evolve. The mesh resolution was also manipulated. Higher resolutions allowed for smoother surfaces at the cost of computing power. The final bodies are 3D printable, but lack a simple actuation mechanism. (Rieffel *et al.*, 2013)

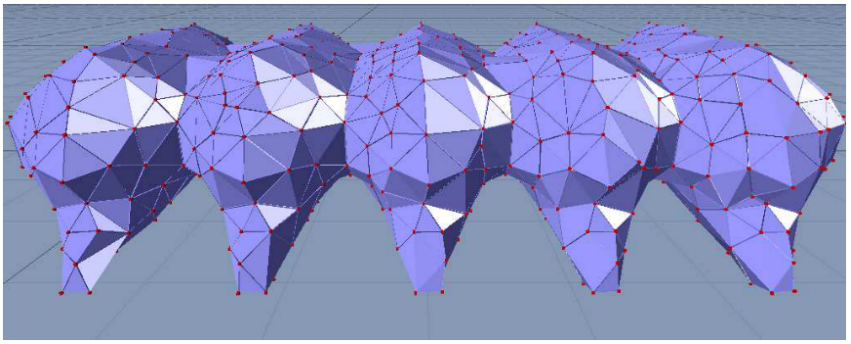


Figure 2.12: Soft robot constructed from a tetrahedral mesh (Rieffel *et al.*, 2013)

2.3.4 Gaussian Mixtures

The Gaussian mixtures representation (Pernkopf and Bouchaffra, 2005) uses a density field analogy to a level-set method. The density field is initialised as having zero density. Points of density with Gaussian falloff are added within the field. The points may have positive or negative weights. Positive weights contribute to the density, while negative weights subtract from it. If a single point was used, a solid sphere would be the thresholded result. (Hiller and Lipson, 2012)

In 2012, Hiller and Lipson modelled soft amorphous bodies using the Gaussian mixtures representation. Gaussian points are listed in a 3D workspace. Densities and falloff ranges are associated with the points. This results in smooth, free-form shapes. Material properties are stored in the points and distributed accordingly between points. The Gaussian mixtures representation is a computationally efficient method of storing and representing complex smooth shapes. Shapes were evolved using evolutionary algorithms. The Gaussian points' coordinates, densities, falloff ranges and material indices were used as parameters of interest during evolution. (Hiller and Lipson, 2012)

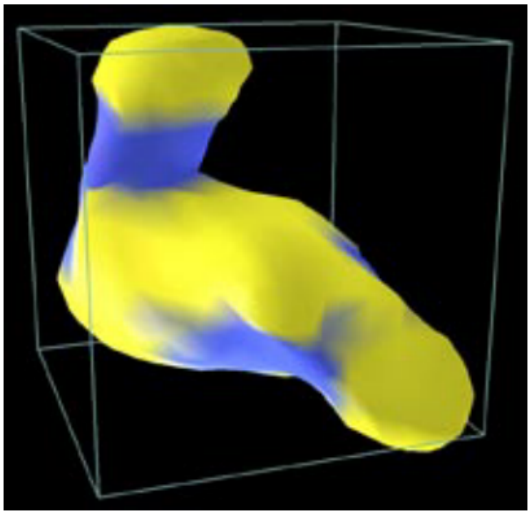


Figure 2.13: Soft body represented by a Gaussian density mixture (Hiller and Lipson, 2012)

2.4 Evolved Virtual Bodies

Optimization of discretely represented mathematical model problems, such as virtual bodies, is an active area of research. These problems have potentially massive solution spaces. An automated search of the design space making use of random and targeted methods is desirable.

2.4.1 Generative Design

A generative encoding is a type of encoding that specifies the construction of a phenotype. A genotype is a programmed representation of a potential individual or problem solution. A phenotype is a set of characteristics of an individual resulting from the composite of its genotypes (Sims, 1994*b*). It may scale well because of its inherent self-similar and hierarchical structure. (Hornby and Pollack, 2001*b*)

An early approach to generative design used a model that encompassed the domain of all components and their possible interactions to design physical bodies with desired behaviours. This approach divided the behaviours and constructed appropriate design fragments for them. These fragments were incrementally composed until a design that obeyed the desired behaviours was obtained. (Brose, 1993)

2.4.2 The Monte Carlo Method

The Monte Carlo method may be applied to physical problems described by systems with moderate numbers of parts. The method involves producing large numbers of results of a given problem. The relative number of successful iterations may be analysed. The method will never yield results with complete certainty. The method may yield results within certain limits with great probability. The method is well suited to generative design problems of which individual iterations are not too computationally expensive (Metropolis and Ulam, 1949).

2.4.3 Evolutionary Algorithms

Evolutionary algorithms, also known as genetic algorithms, are a robust approach to solving optimization problems. They derive their name from their similarity to the evolution of biological organisms. They typically start with a randomly generated population of solutions to a given problem. Each individual solution's suitability is checked. The more suitable solutions are used to generate a new population, by "breeding" existing members of the population, and introducing some random variations. The new population is checked again, and this process is repeated for some number of generations (Groenwold *et al.*, 1999). Evolutionary algorithms are versatile. They can be applied to the optimization of functions and the evolution of complex behaviours and bodily morphologies. They have been specifically applied to the evolution of robotic bodies for many years (Sims, 1994*b*,*a*).

Evolutionary algorithms require a measure of the population's performance. Within the context of evolving simulated physical bodies, a realistic physically simulated goal is usually set. Examples include traversing the greatest distance within a set amount of time, jumping or climbing over an obstacle, or drawing

another object closer to it. Fitness measures may also be implemented as survival criteria in testing, such as energy requirements, size, and complexity of the respective bodies. (Sims, 1994*b*,*a*)

Evolutionary algorithms typically use direct encodings of solutions. They may struggle to successfully design highly complex systems using direct encodings. (Hornby and Pollack, 2001*b*)

In 1994, Karl Sims introduced the concept of evolving three dimensional virtual bodies with the aid of evolutionary algorithms. Four simulations with different fitness evaluation functions were run. The four functions were swimming as far as possible within a limited time period, moving across a flat surface as far as possible within a limited time period, jumping as high as possible from a stationary position, and following a light source. The bodies were simply modelled. Nodes describe the rigid parts of the body, the joints between parts and their parent parts, and the set of connections they have to other nodes. Rigid parts have specific dimension. Joints have different types and limits of motion. Node connections describe a part's relationship to its parent. The bodies are described as a set of nodes. These bodies are evolved in three dimensions according to their performance against one of the simulation fitness functions using evolutionary algorithms for a number of generations. The best-performing models were inspected at the end (Sims, 1994*b*). Further tests were done where two bodies were evolved in the same environment. The bodies had to compete with each other to keep a cube as close to themselves and as far away from the other body as possible. This study investigated the effect of competition between organisms on evolution and optimisation (Sims, 1994*a*).

2.4.4 Lindenmayer Systems (L-systems)

L-systems were originally conceived as a mathematical theory of plant development. They are used in theoretical biology to describe and simulate natural growth processes. They did not originally include enough detail to completely model higher-level plants. They focused on plant topology and not geometry. (Kolodziej, 2002; Prusinkiewicz *et al.*, 2004)

The main component of L-systems is rewriting. Rewriting is used to define complex objects by successively replacing parts (letters) of an initial, simple object (word) according to a set of rules (grammar). Grammars are applied in parallel and simultaneously replace all letters in a given word (Prusinkiewicz *et al.*, 2004). This makes L-systems suitable for describing and generating fractal structures. Words generated by L-systems can be used as genotypes for virtual bodies (Kolodziej, 2002).

There are many variations of L-systems. Deterministic and context-free L-systems (DOL-systems) use edge rewriting to replace polygon edges with figures and node rewriting to operate on polygon vertices. Stochastic L-systems implement randomization to obtain variation in productions. A context-sensitive

L-system's productions' expression may depend on its predecessors' context. (Prusinkiewicz *et al.*, 2004)

Original L-systems are discrete in time and space. Model states are known only at specific time intervals. Only a finite number of model states exist. Parametric L-systems allow for infinite model states due to the assignments of continuous attributes to model components. Parametric L-systems are not limited by all values being reduced to integer multiples of a unit segment. (Prusinkiewicz *et al.*, 2004)

Map L-systems allow for the formation of cycles in a production. Maps are finite sets of regions. Regions are surrounded by boundaries consisting of finite, circular sequences of edges meeting at vertices. Each edge has one or two vertices associated with it (only one if the edge forms a loop). Edges cannot cross without forming a vertex. There are no vertices not associated with an edge. Every edge is part of the boundary of a region. The set of all edges is connected. (Prusinkiewicz *et al.*, 2004)

Some simulations of the branching patterns often achieved by L-systems consider the interactions among the growing features, structures and environment. This makes models more realistic and introduces some complexity. (Prusinkiewicz *et al.*, 2004)

2.4.4.1 Compositional Pattern Producing Networks (CPPN)

To obtain a solution from a space encompassing high dimensionality, an indirect encoding mapping smaller numbers of genotypes to phenotypes may be necessary. Developmental encoding incorporates elements of development as seen in biological processes over time to indirect encodings. (Hornby and Pollack, 2001*a*)

CPPNs as an encoding describe the structural relationships that would develop without having to simulate the development process. The encoding is composed of functions based on gradient patterns as found in natural embryos. CPPNs structurally resemble neural networks and can make use of existing efficient evolutionary methods used for them. (Stanley, 2006)

Features obtained from the use of CPPN-NEAT algorithms include symmetry, reuse, and preservation and elaboration of regularities. (Stanley, 2006)

2.5 Emergent Properties

Data base amplification is the generation of seemingly complex objects from very concise descriptions (Prusinkiewicz *et al.*, 2004). Emergent properties occur when not all components of a given property satisfy that property. An emergent property is not satisfied by the constituent components of a system, but is satisfied by the overall system. If the required condition for a specific property to exist can be determined, it is possible to construct a system sat-

isfying that property from components that do not satisfy that property. If the target property of a system and the property of a component is known, it can be determined if the other component can have a property that will result in the system satisfying the target property. If that property exists, it can be found. (Zakinthinos and Lee, 1998)

Reactive systems consist of interconnected sub-components that are a part of structural links defining communication methods. These systems may exhibit emergent properties that are unpredictable even when complete knowledge of the systems is accessible. This implies the systems are complex in such a manner that they cannot be simplified to rules based on inferences from their properties. Knowledge of the rules of interactions between the sub-components is also necessary. (Aiguier *et al.*, 2008)

Emergent properties are sometimes encountered with generative design. Emergent properties may be complex behaviours that are difficult to predict (Aiguier *et al.*, 2008) and challenging to understand initially, that arise from the combination of the simple elements and rules used to construct the generative design algorithms. For example, virtually evolved bodies may end up being complexly constructed in such a way that the methods of completing their objectives are not initially obvious (Damper, 2000). These emergent properties are desirable, as one advantage of generative design processes is that they may arrive at original and unique designs that may be extremely difficult for a human to arrive at (Sims, 1994*b*).

Chapter 3

Materials and Methods

This chapter covers the materials considered during this project and the methodologies applied throughout. The methodologies encompass generative design approaches such as L-Systems, CPPNs, and evolutionary algorithms. Modelling methods, specifically the FEM settings, are outlined. A detailed overview of the software pipeline is provided.

3.1 Materials

3.1.1 Modelling

The modelling of materials undergoing relatively large deformations is a non-trivial problem. Stored strain energy density may be used to compute stress in hyperelastic materials. The strain energy density is defined using invariants of strain. The three invariants are given as (Kim, 2015)

$$I_1 = \lambda_1^2 + \lambda_2^2 + \lambda_3^2 \tag{3.1}$$

$$I_2 = \lambda_1^2\lambda_2^2 + \lambda_2^2\lambda_3^2 + \lambda_3^2\lambda_1^2 \tag{3.2}$$

and

$$I_3 = \lambda_1^2\lambda_2^2\lambda_3^2 \tag{3.3}$$

where λ_1^2, λ_2^2, and λ_3^2 are three eigenvalues. The undeformed state is used as the frame of reference. The three invariants will not change when using different coordinate systems. The three invariants must be positive for the deformation to be valid. The square root of I_3 measures the volume change of the material. $I_3 = 1$ if the material is incompressible. (Kim, 2015)

The distortional strain energy density is defined as (Kim, 2015)

$$W_1(I_1, I_2) = \sum_{m+n=1}^{\infty} A_{mn} (I_1 - 3)^m (I_2 - 3)^n \tag{3.4}$$

The Ogden model uses the principal stretches to define the distortional strain energy density as (Ogden, 1972)

$$W_1(\lambda_1, \lambda_2, \lambda_3) = \sum_{i=1}^{N} \frac{\mu_i}{\alpha_i} (\lambda_1^{\alpha_i} + \lambda_2^{\alpha_i} + \lambda_3^{\alpha_i}) \tag{3.5}$$

where N, μ_i, and α_i are material parameters. N is usually three. The principal stretches are the three eigenvalues of the deformation gradient. If the material is incompressible, the three principal stresses are not independent, meaning $\lambda_1\lambda_2\lambda_3 = 1$. The shear modulus is (Ogden, 1972)

$$\mu = \frac{1}{2} \sum_{i=1}^{N} \alpha_i \mu_i \tag{3.6}$$

The Ogden model correlates well with simple tension test data that is elongated up to 700%. The model accommodates for slightly compressible behaviour and a nonconstant shear modulus (Ogden, 1972; Kim, 2015).

3.1.2 Selection

Compliant and elastic materials are commonly used in the construction of soft robots. Three specific silicon-based rubbers that are readily accessible and affordable were characterized by Ellis (2020). These rubbers' Ogden material model parameters are shown in Table 3.1.

Table 3.1: Silicon-based rubber Ogden material model parameters (Ellis, 2020)

Parameter	Parameter Number		
	1	**2**	**3**
Ecoflex 00-30			
μ	-0.0142909	-3.64558e-06	9.59447e-08
α	-5.22444	-0.162804	11.3772
Mold Star 15 SLOW			
μ	-6.50266e-06	0.216863	0.00137158
α	-21.322	1.1797	4.88396
Smooth-Sil 950			
μ	-0.30622	0.0283304	6.5963e-09
α	-3.0594	4.59654	17.6852

Mold Star 15 SLOW is selected as the main material to be digitally modelled and manufactured for the purposes of the project. Mold Star 15's stiffness

lies between EcoFlex and Smooth Sil's. Mold Star 15 is selected because of its availability and physical properties. Mold Star 15 is deliverable to the premises where testing is to be done and available from a registered supplier. It is suitable for inflation while being capable of supporting itself at the relevant scale of construction. Additional relevant properties are listed in Table 3.2.

Table 3.2: Additional Mold Star 15 SLOW material properties (Smooth-On, 2014)

Property	Value
Pot life (min)	50
Cure time (min)	240
Tensile strength (MPa)	2.7579

Mold Star 15's long pot life allows for adequate time to prepare specimens thoroughly. Mold Star 15's relatively short cure life prevents long delays while waiting for specimens to cure. Mold Star 15's relatively high tensile strength prevents tears or ruptures at significant deformations.

3.1.3 Mixing and Casting

Mold Star 15 as provided by the manufacturer consists of two components: Mold Star 15-A and Mold Star 15-B. Mold-Star 15-A is white in colour while Mold Star 15-B is blueish-green. The two components remain liquid when exposed to air or other surfaces. When mixed with each other, the components' pot life begins and the mixed material will begin to set. However, unless mixed adequately in the correct amounts, the material properties will differ from those specified and it may not set entirely.

The two components of the material need to be mixed according to a 1:1 mass ratio. A clean mixing bowl is placed on an electronic scale and the scale is zeroed. Mold-Star 15-A is added to the bowl according to half the desired amount of Mold-Star 15. The mass of Mold Star 15-A is carefully noted and the scale is zeroed again. An equal amount of Mold Star 15-B is added to the mixing bowl.

A clean disposable stirrer such as a tongue depressor is used to mix the two Mold Star components thoroughly. The mixture is considered to be thorough when it is one uniform colour and no lighter or darker streaks are visible.

The mixture must be degassed to remove any air bubbles formed during pouring or mixing. The mixture in the bowl is placed inside a degasser and the pressure is lowered to a vacuum pressure. Bubbles will rise to the top of the mixture and pop. When no bubbles have appeared for a minute, the mixture is considered degassed. It may be removed from the degasser once the pressure has equalised.

The mixture may now be cast into a mould. Care must be taken during casting to prevent the formation of more bubbles. If the mould is movable and fits within the degasser, this may be done to remove any extra bubbles formed during casting.

If the mould is open, the material must be levelled to remove excess material and prevent any menisci from forming, which would affect the specimen's thickness. The material is levelled by applying a scraper across the surface slowly. If the scraper is moved too fast, it may remove too much material, as the mixture is viscous and sticky. If the surface area is too large, a flat cover may be applied by slowly lowering it at an angle.

According to Mold Star 15's pot life, the process detailed above must be completed within 45 min. The cast specimen will be ready 4 h after the pot life has ended.

3.2 Pattern Generation

A component of the generative design process of this project is the construction of internal geometries. The project scope is limited to a 2D grid of squares. Pattern generation methods capable of evolving are implemented in order to generate initial geometries and improve upon them.

3.2.1 L-Systems

L-Systems are implemented in the code pipeline as a pattern generation method. L-Systems are implemented according to a context-free L-System structure as defined by Prusinkiewicz *et al.* (2004). Stochastic L-Systems are not implemented to maintain replicability. Parametric L-Systems are not implemented because a single discrete model state is desirable. Maps are not implemented as the resulting reflections when branches collide with the border wall may negatively impact the scalability of the L-Systems. Reflections inward may quickly fill up the available space or result in geometries dissimilar to ones resulting from the same L-System at higher resolutions. Interactions between elements drawn are also not considered.

The L-System unit generation method is implemented due to the features of efficiency, compactness and scalability. L-Systems are defined in the Python code using class objects containing all necessary information.

3.2.1.1 Parameters

The L-System vocabulary is defined as a class object in the Python code. The vocabulary is predefined. It consists of variables and constants. The vocabulary and its interpretations are outlined in Tables 3.3 and 3.4. The vocabulary is separated according to variables and constants. The vocabulary interpretation differs from traditional L-System interpretations. The interpretation

is required to fill in elements of a grid. Traditional L-System interpretations result in lines of varying lengths and angles being drawn. A unique interpreter was designed and implemented for the purposes of this project.

Table 3.3: L-System variable interpretation

Variable	Interpretation
F	Create an element at the current position and increment the current position in the current direction
f	Increment the current position in the current direction
+	Rotate the current direction by 45° clockwise
-	Rotate the current direction by 45° counterclockwise

Table 3.4: L-System constant interpretation

Constant	Interpretation
[	Push the current position to the position memory stack
]	Pop the latest position on the position memory stack and return to that position
(	Push the current position to the position memory stack. All directional variables, i.e. "+" and "-", in the word following this constant are reversed until the ")" constant is encountered
)	Pop the latest position on the position memory stack and return to that position

Randomly generated L-Systems are manipulated using a random generation seed. Randomly generated L-Systems have requirements and specifications implemented to ensure the validity of the L-System.

At least one L-System rule must be defined. This rule must apply to the variable "F" and must itself contain at least one instance of the letter "F". Up to three additional rules may be defined, one for each of the remaining variables.

Rule components are selected from a predefined list included in Table 3.5. Rule components were selected for various reasons. Rule components containing more than one character allow for variability in rule length. Rule components containing two identical directional variables specify 90° rotations. Rule

components enclosed within square brackets allow for branches to exist within the L-System.

Table 3.5: L-System rule components and their interpretations

Number	Rule Component	Interpretation
1	F	Create an element at the current position and increment the current position in the current direction
2	f	Increment the current position in the current direction
3	+	Rotate the current direction by 45° clockwise
4	-	Rotate the current direction by 45° counterclockwise
5	++	Rotate the current direction by 90° clockwise
6	–	Rotate the current direction by 90° counterclockwise
7	fF	2, then 1
8	Ff	1, then 2
9	[F]	Push the current position to the position memory stack, then 1, then pop the latest position on the position memory stack and return to that position
10	[f]	Push the current position to the position memory stack, then 2, then pop the latest position on the position memory stack and return to that position
11	[+F]	Push the current position to the position memory stack, then 3, then 1, then pop the latest position on the position memory stack and return to that position
12	[+fF]	Push the current position to the position memory stack, then 3, then 7, then pop the latest position on the position memory stack and return to that position
13	[-F]	Push the current position to the position memory stack, then 4, then 1, then pop the latest position on the position memory stack and return to that position
14	[-fF]	Push the current position to the position memory stack, then 4, then 7, then pop the latest position on the position memory stack and return to that position

During initial trials, issues were encountered in allowing rules to be generated with completely random variables and constants. Unmatched brackets did not allow for the correct interpretation of the axioms. Without forcing the inclusion of the "F" character in at least one rule or applying at least one rule to the "F" character, many L-Systems did not result in the creation of any elements. If constants were treated as variables, the brackets could also end up unmatched.

Twelve axioms are predefined and outlined in Table 3.6. Axioms were defined according to desirable symmetry conditions. Symmetry conditions are illustrated in Figure 3.2 according to the reference shape in Figure 3.1.

Table 3.6: L-System axioms

Axis	Rotational Axiom	Reflective Axiom
Horizontal	[F]++++[F]	[F]++++(F)
Vertical	--[F]++++[F]	--[F]++++(F)
Horizontal and vertical	[F]++[F]++[F]++[F]	[F]++(F)++[F]++(F)
Diagonal	+[F]++++[F]	+[F]++++(F)
Negative diagonal	-[F]++++[F]	-[F]++++(F)
Diagonal and negative diagonal	+[F]++[F]++[F]++[F]	+[F]++(F)++[F]++(F)

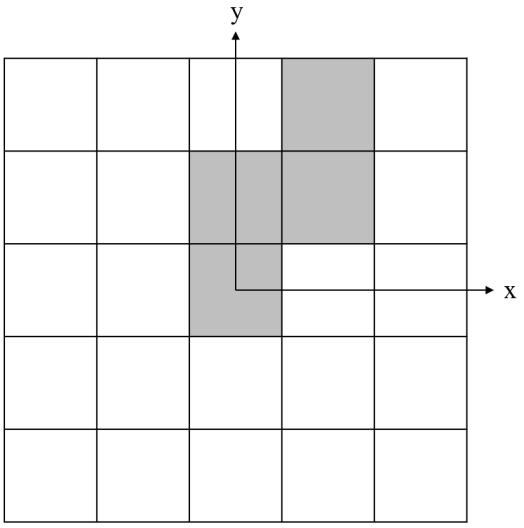
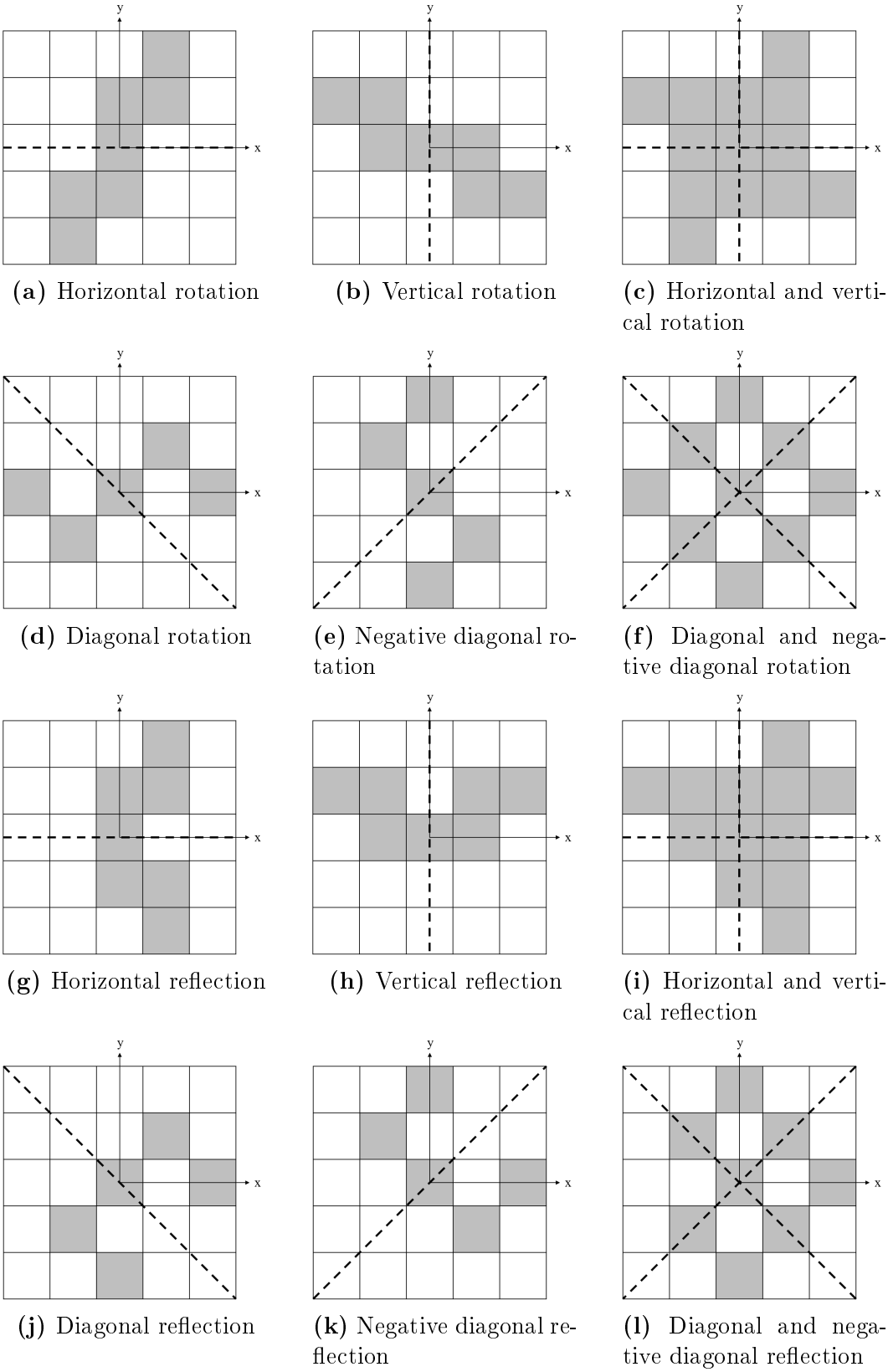


Figure 3.1: L-System symmetry interpretation reference layout



(a) Horizontal rotation

(b) Vertical rotation

(c) Horizontal and vertical rotation

(d) Diagonal rotation

(e) Negative diagonal rotation

(f) Diagonal and negative diagonal rotation

(g) Horizontal reflection

(h) Vertical reflection

(i) Horizontal and vertical reflection

(j) Diagonal reflection

(k) Negative diagonal reflection

(l) Diagonal and negative diagonal reflection

Figure 3.2: L-System interpretation of symmetry axioms according to Figure 3.1. Axes of symmetry are indicated with dotted lines

All L-System rule interpretations are first applied as if the axiom were a single "F". The rules are applied for the specified number of iterations. The resulting word is then placed within the specified axiom of symmetry at all positions occupied by an "F". This allows for the symmetry conditions to be maintained across iterations while retaining "+" and "-" as variables.

3.2.1.2 Interpretation

The final word is interpreted according to the specified internal dimensions of the template. The word is interpreted character by character.

The number of open round brackets in the word is counted. For every open round bracket, the first subsequent closed round bracket is identified. Each string segment starting and ending with an open and closed round bracket is extracted. The round brackets are replaced with the equivalent square brackets. Pluses and minuses are swapped to invert all directional changes within the round brackets. This allows for reflection symmetry conditions to be applied. The original string segment is replaced with the adjusted string segment.

Every character in the word is interpreted from the initial character. The interpretation applies to a grid of squares identified by x- and y-coordinates. The origin square is defined by the coordinates $(0, 0)$. Positive and negative directions are applied as per standard convention. The interpretation starts at the origin with the direction at 0°, i.e. in the positive y-direction.

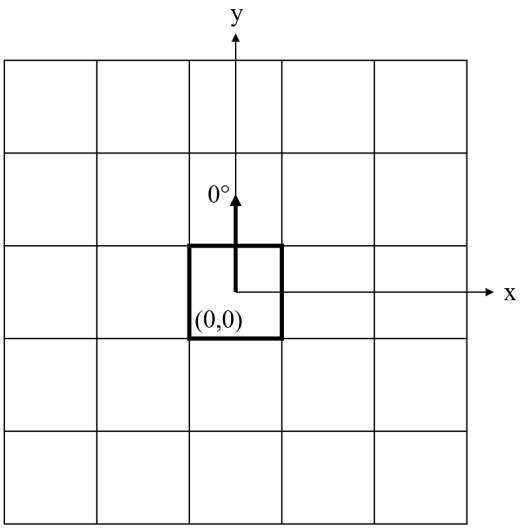
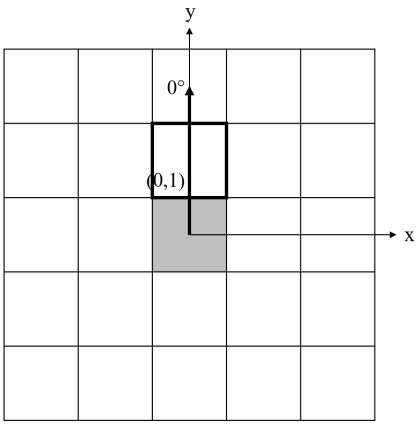


Figure 3.3: Example 5x5 grid for L-System interpretation indicating the origin and orientation. The current position is outlined.

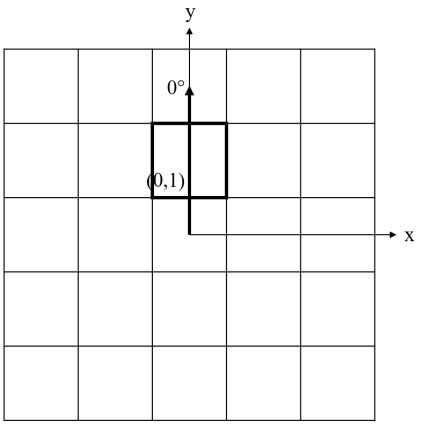
If the character is "F", a check is done to determine if the stack is at its initial value or if the current element is the initial element. If it is either, the element coordinates are set to the origin. If it is neither, the x-

and y-coordinates are incremented in the current direction. The new element coordinates are added to the list of coordinates.

If the character is "f", the procedure is identical to the procedure for the character "F", except that the new element coordinates are not added to the list of coordinates. Thus the position is incremented without an element being added.



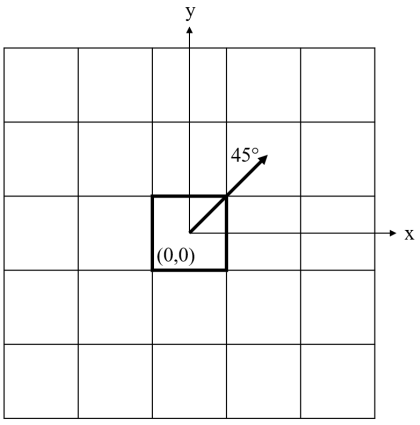
(a) Interpretation of "F" from the origin. A saved element is filled in.



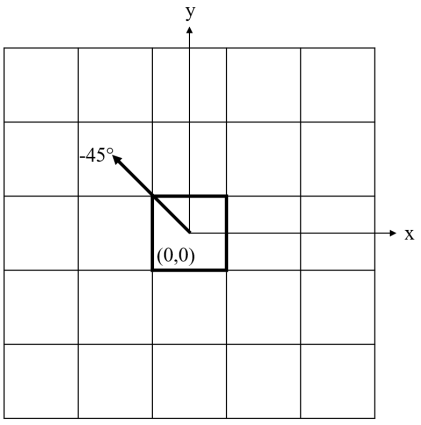
(b) Interpretation of "f" from the origin.

Figure 3.4: L-System interpretation of movement variables

If the character is "+", the current direction is incremented by 45°. If the character is "-", the current direction is decremented by 45°.



(a) Interpretation of "+" from the origin



(b) Interpretation of "-" from the origin

Figure 3.5: L-System interpretation of rotational variables

If the character is "[", the current position and direction is added to the stack. If the character is "]", a check is done to determine if the stack is at its initial value. If it is, the initial coordinates and direction are fetched. If it is not, the latest coordinates and direction are popped from the stack. A check is then done to determine if the stack is at its initial position. If it is, a flag is set to indicate as such.

The complete list of coordinates is sorted in ascending order. All duplicate element coordinates are removed.

3.2.2 CPPNs

A CPPN-like generation method is implemented due to the CPPN features of image generation and scalability. CPPNs are defined using class objects containing all necessary information. A single CPPN may generate a number of models. CPPN models are also defined as class objects containing all necessary information.

A random generation seed is used to determine the CPPN's initial layer, hidden layers, and activation functions. The random generation seed allows for replicability of CPPNs.

A CPPN model may be scaled inwards or outwards. The scale parameter was set at 1, i.e. no scaling, in order to reduce complexity. The functionality of the scaling was left implemented to allow for future tests.

The threshold parameter allows for two approaches to the removal of elements. The CPPN outputs 2D arrays of values ranging from 0 to 1. If the threshold parameter is set from 0 to 1, it is interpreted as a rounding threshold. All values above or equal to the threshold are set to 1 and all values below or equal to the threshold are set to 0. If the threshold parameter is set above 1 to 100, it is interpreted as a percentage of elements to remove. The lowest values making up the specified percentage of all values are set to 0 and the rest of the values are set to 1.

The x- and y-dimensions are specified initially. All hidden layers barring the initial layer have a number of nodes equal to the x-dimension multiplied with the y-dimension. The CPPN will result in a model that fits perfectly within the internal space of the unit.

CPPN models obtained for the purposes of this this thesis are at much lower resolutions than traditional CPPN models. A reduction in complexity was deemed appropriate. Traditional CPPNs allow for unique activation functions to be applied to each node. This was not implemented in order to reduce complexity. Five activation functions are available for the hidden layers of the CPPN. Activation functions are applied to an entire layer. The activation functions included in this project are

- sin

- cos
- tanh
- sigmoid
- smooth ReLu

Only the last two activation functions are available for the output layer of the CPPN. They result in values ranging only from 0 to 1. The values are rounded according to the specified threshold parameter.

3.3 Generative Design Approaches

Generative design methods are required to assist in the determination of well-performing units generated by the pattern generation methods. Large populations of unique units must be created. Each individual's performance should be ranked according to a predefined objective function. Better-performing units should be identified and favoured by the generative design methods.

3.3.1 Monte Carlo Simulations

Monte Carlo simulations operate on the principle that a large enough population of calculated solutions should be fairly representative of the entire population of possible solutions (Metropolis and Ulam, 1949). A Monte Carlo simulation of units is thus implemented as a generative design method.

A population of units is randomly generated according to the specified unit generation method. The unit generation process is outlined in Section 3.7.4. The entire unit population is simulated and ranked according to an objective function specified by the template case. The ranking process is discussed in more detail in Section 3.7.7. Further evaluation of unit performance may be carried out by manual inspection of high-ranking units.

3.3.2 Evolutionary Algorithms

Evolutionary algorithms involve the initial generation of a random population and the improvement of the population members' fitness according to some objective function over a number of generations. An evolutionary algorithm is thus implemented as a generative design method.

An initial random population is generated, run, and evaluated. Two parent population members are selected for evolution. Parents are ranked in descending order according to their fitness. The ranking procedure is discussed in detail in Section 3.7.7. The parent population is iterated through from the most fit to the least fit parent until a parent has been selected. A random

number between 0 and 1 is generated for selection. A fitness weighting function is used to determine the selection of a parent. The fitness weighting function is defined as

$$y = \frac{2\,(p + 1 - r)^c}{p^2 + p} \tag{3.7}$$

Where

- y is the fitness weighting
- p is the population size
- r is the rank of the selected member
- c is a user-defined fitness constant set to 1

The cumulative fitness weighting of all units before and after the current population member is calculated. If the random number is between the two cumulative fitness weightings, the member is selected as the parent. Population members may evolve in three ways.

Crossover may occur between two parents. For every crossover point, a random number between 0 and 1 is generated. If the number is below the specified probability, crossover occurs. A random index in the list of parent parameters is selected. The parameters are swapped between parents from this index onwards. The two resulting children are potentially evolved further.

Random mutation of a child parameter may occur. For every random mutation point, a random number between 0 and 1 is generated. If the number is below the specified probability, random mutation occurs. A random parameter of the child is selected. The parameter is randomly changed to any allowed value for that parameter.

Biased mutation of a child parameter may occur. For every biased mutation point, a random number between 0 and 1 is generated. If the number is below the specified probability, biased mutation occurs. A random parameter of the child is selected. Another random number between 0 and 1 is generated. If the number is below 0.5, the parameter is decreased by 1. If the number is above or equal to 0.5, the parameter is increased by 1. The parameter is bounded by the allowed ranges.

All children are added to the population list. The new population list is used to generate the next population of units.

3.4 FEM

A FEM analysis of material behaviour is implemented. Licensed and maintained FEM software is readily available. As per Section 1.3, original FEM code will not be implemented to limit the scope of the project.

3.4.1 Template

The generative design approach as discussed in Section 3.3 requires large populations of similar units to be created and run. The units need to be similar to meaningfully draw comparisons between them and select the best performing units. A template unit is created containing all necessary and non-unique specifications. The template can be altered according to the specified unit generation method.

The template has all appropriate FEM settings applied. The template is a 2D grid of square elements. 2D quad shell elements, each having four nodes, are used. Plane strain geometrical properties are applied to all elements. A single contact body is defined containing all elements. Material properties are added to all elements according to the Ogden material model. Mold Star 15's Ogden material model parameters as per Table 3.1 are applied.

The template has boundary conditions applied causing deformation according to a specified case. Cases are outlined in Section 3.5. Forced displacement boundary conditions are applied to relevant nodes. Displacement magnitudes are either 0 or a desired magnitude applied according to a table. The table used is defined as a simple $y = x$ function across the interval 0 s to 1 s.

Certain template parameters need to be specified by the user. These parameters are detailed in Section 3.7.2. Once the user-specified parameters have been applied, the template may be created. Template creation is described in Section 3.7.3.

3.4.2 Unit Simulations

Units are altered templates. Units have some internal elements removed. It is assumed that units are actuated by an internal pressure. Internal pressures are applied in the FEM models by edge loads perpendicular to all internal edges which have no neighbouring elements. Figure 3.6 illustrates the internal pressure boundary conditions as applied by the software pipeline.

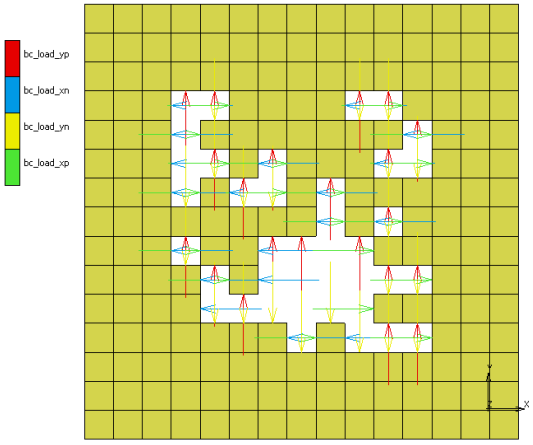


Figure 3.6: Unit internal pressure boundary conditions. The unit ID is grid_39_1_973_1_8_28_42.

Units are not expected to be utilised independently. It is assumed that multiple units will be arranged together in grid-like patterns to construct actuators with desirable behaviours. Units are thus not modelled in isolation. Units are copied to create a 5×5 grid of identical units. Figure 3.7 illustrates the 5×5 grid created from the unit in Figure 3.6. Only the central unit is considered during analysis of the results. The central unit is assumed to be representative of a typical unit in a large arrangement.

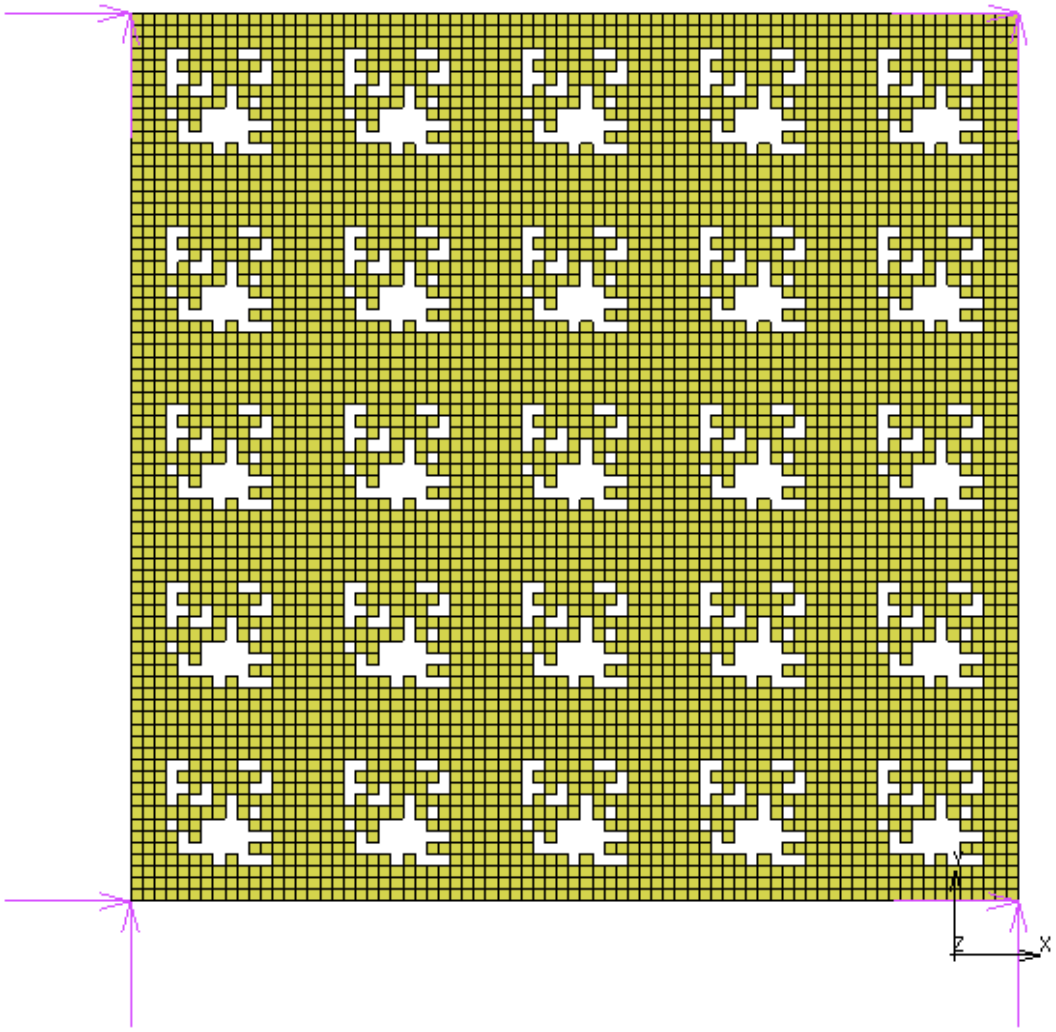


Figure 3.7: 5×5 grid layout of the unit in Figure 3.6 with fixed boundary conditions indicated

Fixed boundary conditions are applied to prevent rigid body modes. The four outermost nodes of the 5×5 grid arrangement are fixed in the x- and y-direction. These boundary conditions are visible in Figure 3.7.

3.5 Unit Deformation Cases

Three unit deformation cases were defined, implemented and evaluated with the software pipeline. The pipeline allows for easy implementation of new unit deformation cases and methods of evaluation. The unit deformation cases were selected due to their robustness, potential applications and simplicity.

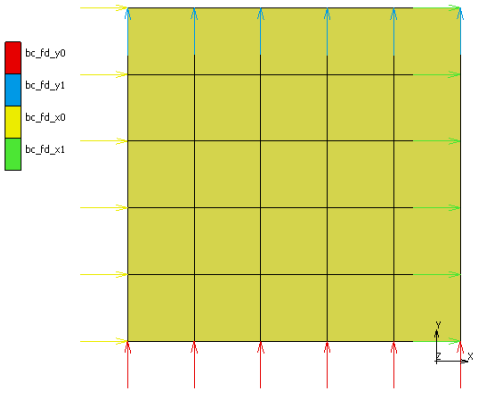
3.5.1 Uniaxial Elongation

Case 1 is a case of uniaxial elongation as defined by Kim (Kim, 2015). This case has no rigid body modes. Four boundary conditions are applied. They are outlined in Table 3.7.

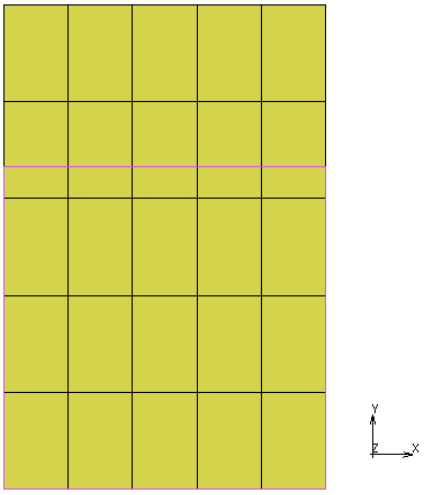
Table 3.7: Uniaxial elongation boundary conditions (Kim, 2015)

Label	Boundary	Constraint	Direction
bc_fd_yy1	Bottom edge	Fixed	y
bc_fd_yy2	Top edge	Forced displacement	y
bc_fd_xx1	Left edge	Fixed	x
bc_fd_xx2	Right edge	Fixed	x

The boundary conditions as applied in Marc Mentat to a template of 5×5 elements are illustrated in Figure 3.8a. The resulting deformation is illustrated in Figure 3.8b.



(a) Uniaxial elongation boundary conditions as applied in Marc Mentat



(b) Resulting deformation of uniaxial elongation. The original shape is outlined in pink.

Figure 3.8: Uniaxial elongation boundary conditions and resulting deformation.

This case has applications in causing extension. Multiple units combined and oriented in the same direction would result in a linearly extending actuator.

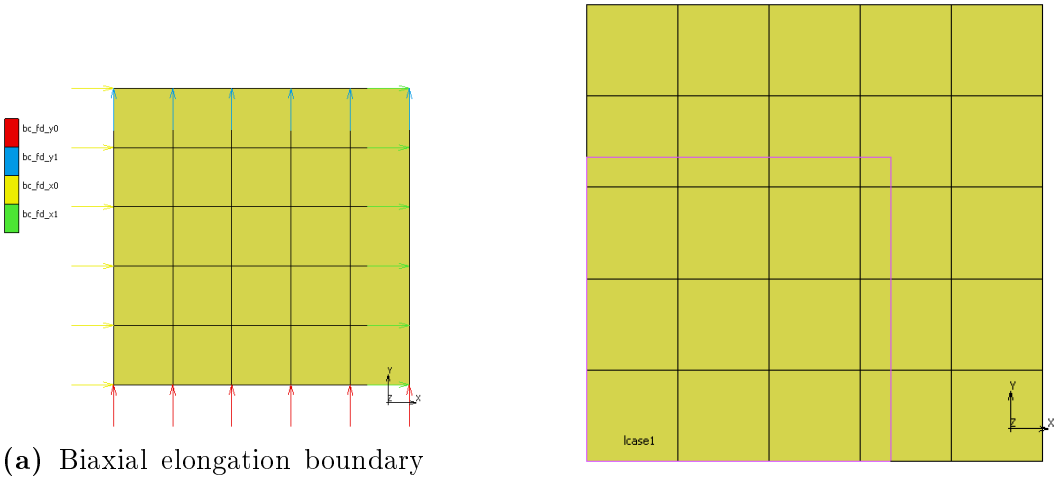
3.5.2 Biaxial Elongation

Case 2 is a case of biaxial elongation. Four boundary conditions are applied. They are outlined in Table 3.8.

Table 3.8: Biaxial elongation boundary conditions

Label	Boundary	Constraint	Direction
bc_fd_yy1	Bottom edge	Fixed	y
bc_fd_yy2	Top edge	Forced displacement	y
bc_fd_xx1	Left edge	Fixed	x
bc_fd_xx2	Right edge	Forced displacement	x

The boundary conditions as applied in Marc Mentat to a template of 5×5 elements are illustrated in Figure 3.9a. The boundary conditions are visually identical to Figure 3.8a. The resulting deformation is illustrated in Figure 3.9b.



(a) Biaxial elongation boundary conditions as applied in Marc Mentat

(b) Resulting deformation of biaxial elongation

Figure 3.9: Biaxial elongation boundary conditions and resulting deformation

This case has applications in causing expansion. Multiple units combined and arranged in a grid would result in an inflating actuator that retains its shape.

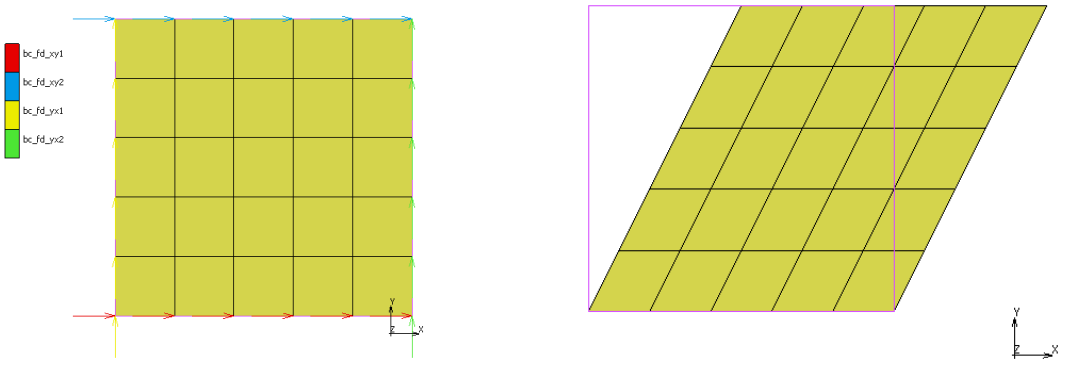
3.5.3 Pure Shear

Case 3 is a case of pure shear as defined by Kim (Kim, 2015). This case has no rigid body modes. Four boundary conditions are applied. They are outlined in Table 3.9.

Table 3.9: Pure shear boundary conditions (Kim, 2015)

Label	Boundary	Constraint	Direction
bc_fd_xy1	Bottom edge	Fixed	x
bc_fd_xy2	Top edge	Forced displacement	x
bc_fd_yx1	Left edge	Fixed	y
bc_fd_yx2	Right edge	Fixed	y

The boundary conditions as applied in Marc Mentat to a template of 5×5 elements are illustrated in Figure 3.10a. The resulting deformation is illustrated in Figure 3.10b.



(a) Pure shear boundary conditions as applied in Marc Mentat

(b) Resulting deformation of pure shear

Figure 3.10: Pure shear boundary conditions and resulting deformation

This case has applications in causing angular deformation. Multiple units combined and oriented in the same direction would result in an angular actuator. Combining multiple actuators could result in curling actuators or actuators with more complex behaviours.

3.5.4 Objective Functions

To enable the software pipeline to identify well-performing units according to the given template deformation case, objective functions unique to each case must be defined. If cases are added to the software pipeline, objective functions must be added for them as well.

For the uniaxial elongation case, it is desired that the unit elongates while its width remains constant. Two objective functions are identified. The objective function defining elongation is

$$f = \frac{h}{w} \tag{3.8}$$

Where

- f is the objective function value
- h is the height of the unit
- w is the width of the unit

The objective function defining constant width is

$$f = \max\left(\frac{w}{w_t}, \frac{w_t}{w}\right) \tag{3.9}$$

Where

- w_t is the width of the template

It is desired to maximise both of these functions.

For the biaxial elongation case, it is desired that the unit retains its shape while enlarging. Retaining its shape implies that the height to width ratio remains as close to 1 as possible and that the outer sides remain as straight as possible. Two objective functions are identified. The objective function defining the retention of the height to width ratio is

$$f = |h - w| \tag{3.10}$$

The objective function defining the outer sides remaining straight is

$$f = \sum_{i=1}^{4} H_d\left(s_{t,i}, s_i\right) \tag{3.11}$$

Where

- s_t is the set of node coordinates of a side of the template
- s is the set of node coordinates of a side of the unit
- $H_d\left(x, y\right)$ is the Hausdorff distance as defined in Section 3.5.4.1

It is desired to minimize both of these functions.

An objective function regarding the enlargement is not defined. The same pressure is applied to all units regardless of the number of elements removed. Units with more elements removed have more edge loads applied. Each edge load is a pressure load of the same magnitude. Elements with more internal edges will have higher total pressures applied and will enlarge more. This would be an unfair objective function compared to the other two selected objective functions. A unit with all internal elements removed may enlarge by

a significant factor while another element retains its shape better. The empty unit may rank higher due to a singular high fitness value.

For the pure shear case, it is desired that the top left and right corners shift horizontally to the right with respect to the bottom left and right corners respectively. Three objective functions are identified. The objective functions defining the rightward shift of the top corners are defined as

$$f = x_{tl} - x_{bl} \tag{3.12}$$

And

$$f = x_{tr} - x_{br} \tag{3.13}$$

Where

- x_{tl} is the x-coordinate of the top left corner
- x_{bl} is the x-coordinate of the bottom left corner
- x_{tr} is the x-coordinate of the top right corner
- x_{br} is the x-coordinate of the bottom right corner

The objective function defining constant height is

$$f = \max\left(\frac{h}{h_t}, \frac{h_t}{h}\right) \tag{3.14}$$

Where

- h_t is the height of the template

It is desired to maximise both of these functions.

3.5.4.1 Hausdorff Distance

3.6 Model Validation

Validation of the FEM models is preferable in order to verify the accuracy of the experimentally obtained Ogden material model parameters and justify the code pipeline's identification of better-performing units. An experimental setup is designed and constructed to validate the FEM models.

3.6.1 Physical Components

A modular mould is designed for the casting of multiple units. Modularity is desirable as all units are derived from a single template, but individual units may vary greatly. Manufacturing times, costs and complexity would be too high if a mould had to be constructed for every unit that was desired to be tested.

The modular mould consists of a mould frame and two types of mould cells. The mould frame is square with dimensions indicated in Figure 3.11. The cells are square with thicknesses of either 2 mm or 4 mm, as illustrated in Figure 3.12. The mould frame is placed on a smooth, level granite tabletop. The mould cells are arranged within the frame according to the layout of the unit to be cast. 2 mm cells are placed where elements exist according to the unit layout. 4 mm cells are placed where elements have been removed. Figure illustrates an example unit layout and the arrangement of the mould before the material has been cast.

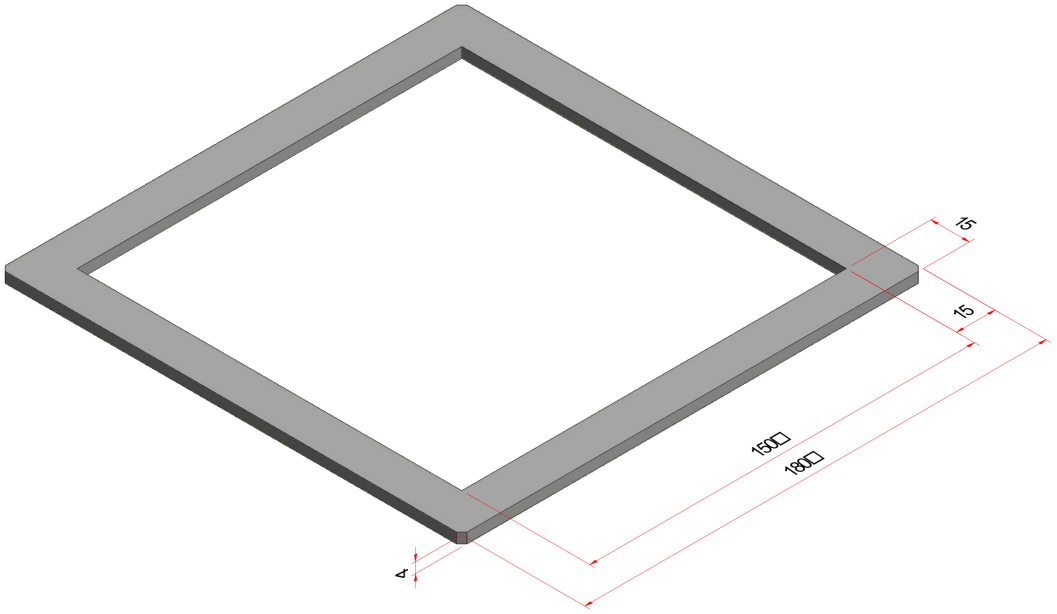


Figure 3.11: Dimensioned modular mould base

(a) Small mould cell **(b)** Large mould cell

Figure 3.12: Dimensioned mould cells

The experimental setup is comprised of two perspex plates separated by 4 2 mm thick washers kept in place by 4 screws, bolts and footpieces on a level surface. The bottom perspex plate has a hole in its centre with a pressure supply tube attached to it. Figure 3.13 illustrates the experimental setup. A camera is affixed to a tripod, positioned over the experimental setup and aimed directly downward at it.

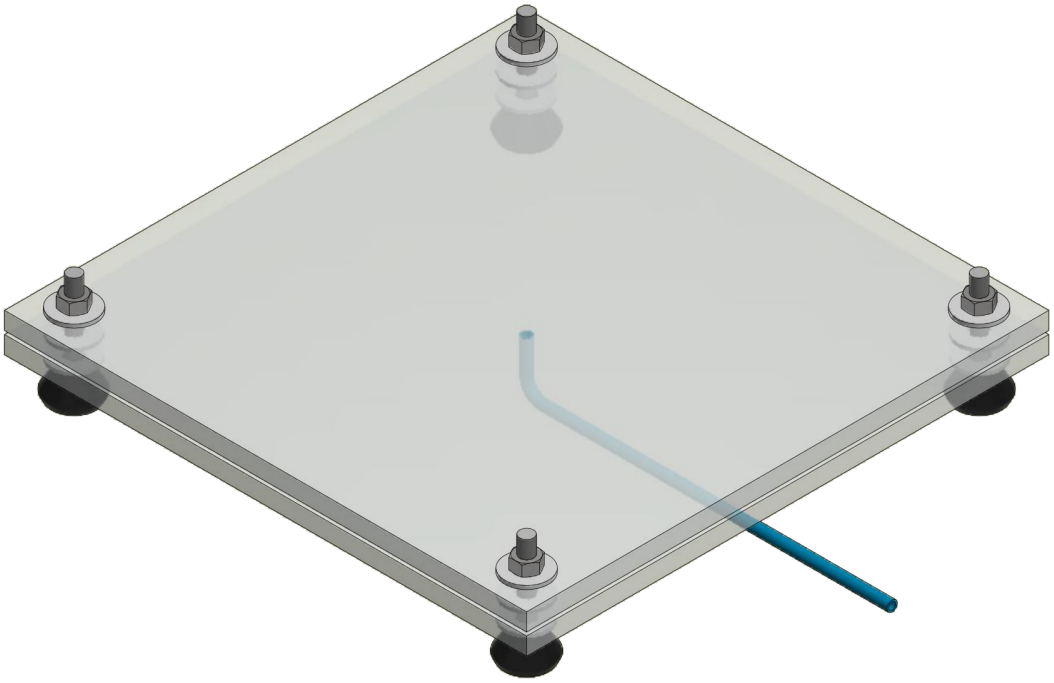


Figure 3.13: Experimental setup model

Perspex is selected as the testing surface for various reasons. It has a low frictional coefficient, decreasing the effects of friction on the experiment. It is transparent, allowing for the effects of the applied internal pressure to be viewed and recorded. It is durable enough at the thickness used to resist the internal pressures applied and be manufacturable. It is flexible enough to allow for some deformation without breaking while not deforming due to the internal pressure.

3.6.2 Experimental Procedure

The material is cast in the mould after having been prepared as outlined in Section 3.1.3. The physical unit is removed from the mould once it has set completely. The mould may then be rearranged according to a new unit configuration after having been appropriately cleaned.

The FEM models applied a linearly ramping internal pressure to the units. The internal pressure was only applied to a predetermined maximum value. The experimental setup will replicate applied pressures at specific points along the ramp and potentially beyond the maximum simulated pressure.

The experimental setup is put in place with the top perspex plate and bolts having not yet been placed. The surfaces of the perspex plates which will come into contact with the physical unit are lubricated with a silicon-based spray. The spray will react with the physical unit over time, causing it to degrade. Tests must be done in a timely manner to prevent the effects of the spray from impacting results.

The physical unit is placed on the bottom plate with its cavity positioned over the pressure inlet hole. The top plate is affixed to the bottom plate separated by the 2 mm washers. The camera is switched on and placed into focus. The pressure valve is slowly opened to desirable pressure intervals. At each interval, photographs are taken of the deformation and the pressure applied is noted.

3.7 Software Pipeline

The software pipeline is complex, and many components are interrelated. Details considered trivial or irrelevant are not discussed. The entirety of the code is available to the reader if they would wish to investigate or make use of it. The code is extensively commented for any clarification needed.

3.7.1 Software Selection

The Finite Element Method (FEM) is an approach to numerically solving field problems. Field problems require the determination of one or more dependent variables' distribution in space. Field problems are mathematically described with differential equations or integral expressions. Finite elements can be expressed as small parts of a larger body. A field quantity within an element may only have a simple spatial variation, such as being described by polynomial terms no higher than the second order. FEM differs from calculus as calculus uses infinitesimal elements. FEM thus delivers approximate solutions. (Cook *et al.*, 2002)

Several commercial FEM software packages capable of realistically modelling soft bodies are available. LSDyna is a FEM software package widely used in industry. It is owned by ANSYS and maintained by LSTC. The software's

code is based on highly non-linear and transient dynamic FEM with explicit time integration (LSTC). Siemens NX 12 is an integrated software package capable of performing FEM analysis. The software package has a user-friendly interface for graphical design of components (Siemens). MSC.Marc Mentat is a pre- and postprocessing software for the MSC.Marc FEM solver. It is focused on nonlinear material modelling and analysis. It has an extensive set of options available for post-processing (MSC, 2003).

MSC.Marc Mentat 2019 is selected for this project. Marc Mentat is actively maintained and updated. Tutorials and reference manuals are built into the software. Support is available if issues are encountered or for any other reason. Limitations include the necessity of having continuous access to a license server. An official VPN configuration as well as remote desktop tools allowed for access to the software as long as a network connection could be established.

Python 3.6 is used to construct a modelling pipeline. Packages are available that allow for the integration of Python and Marc. Many advanced numerical analysis packages are also available for Python. The Python code is accessible, installable and editable. Additional information on how to install and edit the code is available in Appendix .

3.7.2 Initialisation

Before a set of units is created, the user needs to specify certain parameters for the template, analysis approach and unit generation method.

Template parameters that need to be specified are outlined in Table 3.10. Prerequisites for the parameters are included where applicable.

Table 3.10: Template class object initialisation parameters and prerequisites

Parameter	Description	Prerequisite
case	The template case identifier	1, 2, 3
x_ e	The number of elements in the x-direction	> 0, int
y_ e	The number of elements in the y-direction	> 0, int
e_ s	The side length of the element in mm	> 0
b	The number of elements reserved for the unit boundary width	> 0, int
ogd_ mat	The Ogden material model parameters	*mold_ star_ 15* *ecoflex_ 0030* *smooth_ sil_ 950*
n_ steps	The number of analysis steps in 1 s of analysis	> 0, int
table_ name	The name of the function applied to the template	str
d_ mag	The applied displacement in mm	
p_ mag	The applied internal pressure in MPa	

The template case identifier refers to the unit deformation cases in order as outlined in Section 3.5. Additional unit deformation cases may be defined by adding the appropriate code, as well as additional Ogden material models.

A unit boundary is always included. The unit boundary is a layer of elements along the side of the unit which are not allowed to be removed by the unit generation process. The unit boundary ensures that the internal geometry remains enclosed to allow for inflation. The unit boundary also ensures that units will always be capable of being packed into a grid alongside one another.

The name of the function applied to the template is a simple string label to identify the graph type applied to the deformation and internal pressure boundary conditions. The graph may be modified from the currently implemented ramp input to any graphical function by editing the Python code appropriately.

Analysis approach parameters that need to be specified are outlined in Table 3.11. Prerequisites for the parameters are included where applicable.

Table 3.11: Analysis approach initialisation parameters and prerequisites

Parameter	***Description***	***Prerequisite***
a_meth	The analysis approach method	m, g
g_meth	The unit generation method	r, l, c
n_u	The number of units to generate	> 0, int

The analysis approach method is specified by a single character. An "m" indicates the Monte Carlo method should be used. A "g" indicates the evolutionary algorithm method should be used. The unit generation method is specified by a single character. An "r" indicates random unit generation should be used. An "l" indicates L-Systems should be used. A "c" indicates CPPNs should be used. Additional analysis approach methods and unit generation methods may be defined by adding the appropriate code.

The number of units to generate applies to the entire Monte Carlo method, or a single generation's population if the evolutionary algorithm is used.

3.7.3 Template Creation

The user-specified template parameters are used to define a template class object. The template class object calculates more parameters used in the construction of the template. The relevant calculated parameters are outlined in Table 3.12.

Table 3.12: Template class object internally calculated parameters

Variable	Description	\textbf{Formula}
x_s	The side length of the template in mm in the x-direction	$e\_s \times x\_e$
y_s	The side length of the template in mm in the y-direction	$e\_s \times y\_e$
x_n	The number of nodes in the x-direction	$x\_e + 1$
y_n	The number of nodes in the y-direction	$y\_e + 1$
n_e	The total number of elements in the template	$x\_e \times y\_e$
n_n	The total number of nodes in the template	$x\_n \times y\_n$
$e\_internal$	The list of internal element IDs that are allowed to be removed	Function
$n\_external$	The list of external node IDs	Function
t_id	The template ID	Function
grid	A representative grid of ones	Function
fp_t	The file paths of all relevant files	Function

The list of internal element IDs refers to the elements contained within the boundary of the unit. Only these elements are allowed to be removed by the unit generation method. The IDs are obtained by an internal function. The function algorithm is detailed in Figure .

The template ID is a formatted string. The ID is generated using unique template class object parameters. The ID is formatted as

$$\text{grid}\_<case>\_<x\_e>\text{x}<y\_e>\_<x\_s>\text{x}<y\_s>$$

A grid of ones is generated to represent the template in coordinate calculations. The grid is generated according to the element resolution of the template. Ones are representative of elements in place. When elements are removed during unit generation, they are replaced with zeroes.

The file paths are generated for all relevant files. The template ID is used as the base name of every file. Additional identifiers and file formats are appended as necessary. A more detailed description of the file management system may be found in Section .

The template is created in Marc Mentat. The nodes are created starting at the global origin on the XY-plane. The nodes are incrementally added in the positive x-direction. The nodes are spaced apart as defined by e_s. Once a row of nodes is completed as defined by x_n, the y-coordinate is positively incremented as defined by e_s. A new row of nodes is created. This process is repeated until completed as defined by y_n.

Four nodes are used to make square 2D elements. Starting at the global origin, elements are incrementally added in the x-direction until completed as defined by x_e. All rows are added until completed as defined by y_e.

The graph used to apply the boundary conditions is defined. The boundary conditions are applied according to the case identifier. The boundary conditions related to each case are detailed in Section 3.5.

Mechanical planar strain geometric properties are added to all elements. The Ogden material model for Mold-Star 15 is applied to all elements. A single contact body is defined containing all elements. The loadcase containing the fixed and forced displacement boundary conditions is created. The job for the loadcase is created.

The template is saved at this point. All units created during a simulation are built from this template. The template job is run and its success evaluated. The process of running a simulation and evaluating its success is outlined in Section 3.7.5.

The template model is saved again. Meaningful template parameters and data obtained from the template is written in a human-readable manner to a log file. The template class object is saved to a file that can be accessed later.

3.7.4 Unit and Population Generation

Unit generation refers to the process of automatically generating internal geometry of a unit based on the predefined template. Three methods of unit generation are implemented and investigated. All three methods provide a list of internal element IDs provided to MarcMentat as elements to be removed from the template file.

Random generation is implemented as a baseline to compare with the other two unit generation methods. A random generation seed is used to allow for replicability. A number of elements to be removed is first selected. A list of unique internal element IDs is then selected and sorted.

L-Systems and CPPNs as outlined in Sections 3.2.1 and 3.2.2 respectively are also available for unit generation. A list of maximum and minimum parameter values is defined. The parameter values are appropriate to the unit generation method specified. The different parameters, ranges and motivations are outlined in Table 3.13. The range values are inclusive.

Table 3.13: Unit generation method parameter ranges and motivations

Parameter	Range		Motivation
	Min	Max	
Random			
Seed	1	Number of units to be generated	The seed is limited to the number of units requested by the simulation
Number of elements removed	0	Number of internal elements	The complete range of elements which may potentially be removed is included
L-System			
Seed	1	Number of units to be generated	The seed is limited to the number of units requested by the simulation
Axiom ID	1	Number of predefined axioms	The complete list of axioms is included
Number of rules	1	Number of L-System variables	At least one rule is required for a valid L-System, and the number of rules is limited to the number of L-System variables
Rule length	2	5	A rule must be longer than one character, and potential complexity is limited
Number of iterations	1	5	At least one iteration must be applied for a valid L-System, and potential complexity is limited
CPPN			
Seed	1	Number of units to be generated	The seed is limited to the number of units requested by the simulation
Model ID	1	N/A	Potential complexity is limited
Scale	1	N/A	Potential complexity is limited
Number of hidden layers	2	10	At least two layers are required to have a valid network, and potential complexity is limited
Size of the initial hidden layer	2	32	
Element removal threshold	0	100	The complete range of elements which may potentially be removed is included

A population of units is generated according to the specified unit generation method. Each population member is a list of parameters within the bounds specified in Table 3.13. If the unit generation method is random, the two parameters are used to generate a list of elements to be removed directly.

If the unit generation method is specified as L-Systems, the parameters are used to generate a random L-System class object. Specified and internally generated L-System class object parameters are outlined in Table 3.14.

Table 3.14: L-System class object parameters and descriptions

Parameter	Description
seed	The random generation seed used to generate the L-System
vocab	The L-System vocabulary
gramm	The L-System grammar
axiom	The initial axiom of the L-System
n	The number of iterations to apply to the L-System
word	The resulting word of the L-System

The L-System word is interpreted to obtain a list of elements to be removed. L-System word characters are interpreted according to Tables 3.3 and 3.4. The interpretation is done from the first character in the word to the last. The interpretation results in a set of coordinates centred around $(0, 0)$. The coordinates are shifted from being centred around the origin to being centred around the central element of the unit. All coordinates outside of the internal space of the unit are dropped. The remaining coordinates are interpreted as element indices. These element indices are compared to the list of all internal element indices to obtain the list of elements to be removed.

If the unit generation method is specified as CPPNs, the parameters are used to generate a random CPPN class object. Relevant CPPN class object parameters are outlined in Table 3.15. The CPPN is set to generate the maximum number of models specified by the parameter. A CPPN model class object is defined with the model ID specified by the current parameters. CPPN model class object parameters are outlined in Table 3.16.

Table 3.15: CPPN class object parameters and descriptions

Parameter	Description
seed	The random generation seed used to generate the CPPN
mod_n	The number of models to be generated by the CPPN
scale	The scale of the focus on the model
hl_n	The number of hidden layers in the network
hl_s	The size of the initial hidden layer
thresh	The rounding or element removal threshold
x	The number of elements in the x-direction
y	The number of elements in the y-direction
res	The resolution of the resulting models
grid	The resulting models

Table 3.16: CPPN model class object parameters and descriptions

Parameter	Description
cppn	The CPPN class object containing all defined models
mod_id	The ID of the CPPN model
grid	The binary model

The resulting grid from the CPPN model class object is a binary 2D grid dimensioned according to the internal space from which elements may be removed. The binary 2D grid is interpreted to obtain element indices for every zero. This list of element indices is the list of elements to be removed.

Each list of elements to be removed is paired with its unit generation method class object or list of parameters. Each pair is added to the population list. For each population member, the template is opened and the relevant elements are removed. Further settings are applied as outlined in Section 3.4.2.

It was observed during initial testing that many units generated with L-Systems had either all or no internal elements removed. Two units, one with all elements removed and one with no elements removed, are generated and evaluated before a population is generated. If any population member is detected to have all or no internal elements removed, it is not evaluated again. Results from the relevant initial units generated are attributed to it. This reduces the overall runtime of the simulations.

3.7.5 Running a Job

The command to run the job is sent to Marc Mentat. Jobs may take anywhere from 0.01 s to 300 s to complete. This depends on the complexity of the model and the number of cut-backs during calculation required to accurately solve for the model behavior.

Marc Mentat creates several files during the process of running a job. The log file specifies the exit condition of the job. The log file is not created at the start of the job.

A model is always saved just before a job is run. The time stamp of this saved model is used for evaluation of the log file. It is first determined if the log file exists. If it does not exist, the code waits for 1 second before checking again. This repeats until the log file is found to exist. If the log file is found, its time stamp is compared to the model file's time stamp. If the log file is older than the model, i.e. it is a log file of a previous run of the model, the code waits 1 second before checking if it has been updated. If the log file is newer than the model, it is inspected for the exit number string or the access violation string.

If the exit number string is found, the exit number is evaluated. Two exit numbers and an error case are identified and defined in Table 3.17.

Table 3.17: Marc Mentat exit number descriptions

Exit Number	Description
3004	A successful run
67	A license server connection timeout or failure
Other	An unsuccessful run

If exit number 3004 is found, the model is recognized as having run successfully. The model output file is opened. All relevant data is read from the model output file and written to clearly labeled CSV files. Any relevant data that must be calculated externally from Marc Mentat is calculated and also written to clearly labeled CSV files. Relevant data is outlined and motivated in Section .

If exit number 67 is found, the job is rerun and the entire process as detailed above is repeated until the exit number can be rerun again. If an access violation string is found, it is treated identically to exit number 67.

If any other exit number is found, an error message with the number is displayed. No results are obtained from the model. The model ID is logged appropriately.

The model is saved and the template model is reopened. The code continues on to the next model.

3.7.6 Analysis Approach

If the analysis approach was specified as the Monte Carlo method, a single population of units is generated and ranked. Further evaluation of unit performance may be carried out by manual inspection.

If the analysis approach was specified as evolutionary algorithms, multiple populations of units are generated, ranked, and evolved using the evolutionary algorithm outlined in Section 3.3.2.

3.7.7 Ranking

Unit objective function evaluations, or performance measures, are weighted equally. To compare performance measures equivalently, values are studentized according to

$$f = \frac{x - mean}{std} \tag{3.15}$$

Where

- f is the fitness value
- x is the performance measure
- *mean* is the mean of all units' performance measures
- *std* is the standard deviation of all units' performance measures

All fitness values applicable to a unit are summed together. Units are ranked in either descending or ascending order, depending on whether objective functions were desired to be maximised or minimised respectively.

A list of the ranked population members is generated. All empty or full units are inserted in the list at the position occupied by the initially generated empty or full units. All failed units are appended to the end of the list.

3.7.8 File Management

A massive amount of FEM and log files are generated by the software pipeline. A single successful FEM model simulation results in five files being generated by Marc Mentat. An additional log file is generated by the Python code. Thousands of units may be generated during a simulation. Multiple log and graph files are generated for every population.

File names and paths are necessarily generated by the Python code. A hierarchical file naming system was set up to generate unique file names and paths. The file naming system efficiently sorts files according to the template parameters, unit parameters, and time of the simulation. The file naming system generates unique names for every unit generated based on the unit parameters. Files are easily sorted through manually by a user looking to inspect a specific unit, or by the code attempting to access it. Desired alterations to the default file path need only be made once because of the hierarchical nature of the system and the implemented code methodology.

Figure 3.14 illustrates the file hierarchy. Parameters are outlined in Table 3.18.

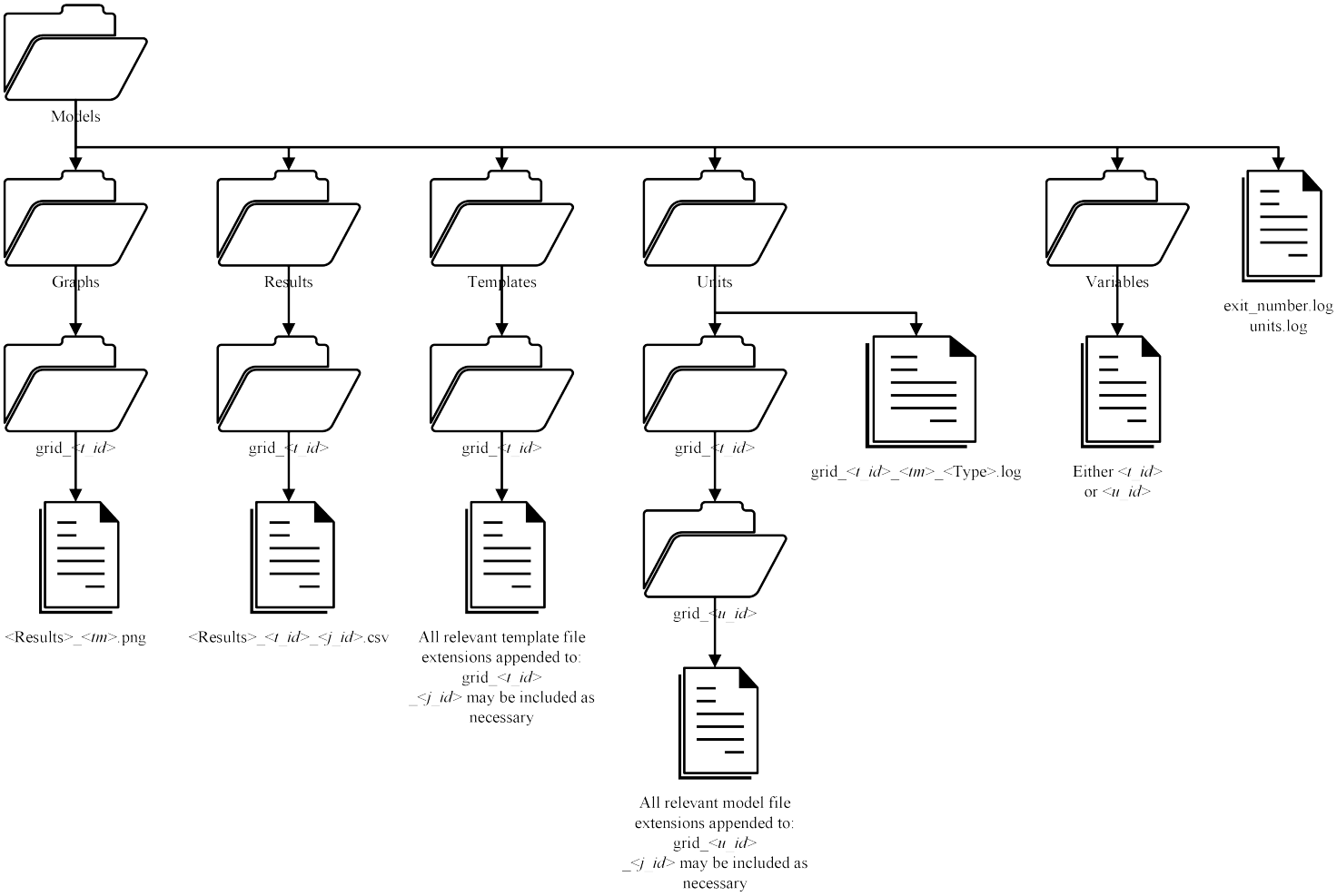


Figure 3.14: File hierarchy flow diagram

Table 3.18: File hierarchy parameters and descriptions

Parameter	Description
tm	The timestamp of the simulation formatted as a string
t_id	The template ID
u_id	The unit ID
j_id	The job ID
Results	The type of simulation results stored
Type	The type of units logged

The template ID is described in Section 3.7.3. The job ID refers to one of two jobs. The first job applies the unit deformation boundary conditions as described in Section 3.5. The second job applies the internal pressure boundary conditions as described in Section 3.4.2.

The unit ID is determined by the unit generation method. All unit IDs start with the number of elements removed followed by an underscore. If the unit generation method is random, a unique hash code is generated according to the list of element IDs that have been removed. If the unit generation method is an L-System, a unique hash code is generated according to the L-System rules. If the unit generation method is a CPPN, the unit ID is formatted as

$$<mod\_id>\_<seed>\_<scale>\_<hl\_n>\_<hl\_s>\_<thresh>$$

The type of units logged in the log files are one of six possible types:

- No Type - All units
- success - All units that have run successfully
- failed - All units that have run unsuccessfully
- empty - All units that have no internal elements
- full - All units that have all internal elements
- ranked - All units ranked according to their fitness values

Chapter 4

Results

This chapter illustrates results obtained from sample simulations of large populations of units according to the outlined unit generation methods in Section 3.7.4 and the analysis approaches outlined in Section 3.7.6. The software pipeline described in Section 3.7 was used to generate the units and obtain the results.

Table 4.1 contains the default parameter values. Unless specified otherwise, parameters were set to the values in Table 4.1. Additional parameters for evolutionary algorithms are included in Table 4.2.

Table 4.1: Default simulation parameter values

Parameter	**Value**
n_u	1000
y_e	15
e_s	10
b	3
n_steps	5
d_mag	$\frac{y\_e \times e\_s}{2} = 75$
p_mag	0.025
Monte Carlo Analysis	
n_u	1000
Evolutionary Algorithm	
n_u	50
gen	50

Table 4.2: Evolutionary algorithm event parameter values

Event	Probability (%)	Potential Number of Occurences
Crossover	50	1
Random mutation	10	2
Biased mutation	50	2

4.1 Random Unit Generation

4.1.1 Monte Carlo Analysis

Parameter ranges are outlined in Table 4.3.

Table 4.3: Random unit generation parameters for a Monte Carlo analysis

Parameter	Minimum	Maximum
Seed	1	1000
Number of elements removed	0	81

Figures and represent the relationships between the parameters in Table 4.3 and the score as calculated in Section .

4.2 L-System Unit Generation

4.2.1 Monte Carlo Analysis

Parameter ranges are outlined in Table 4.4.

Table 4.4: L-System unit generation parameters for a Monte Carlo analysis

Parameter	Minimum	Maximum
Seed	1	1000
Axiom ID	1	12
Number of rules	1	4
Rule length	2	5
Number of iterations	1	5

Figures to represent the relationships between the parameters in Table 4.4 and the score as calculated in Section .

4.3 CPPN Unit Generation

4.3.1 Monte Carlo Analysis

Parameter ranges are outlined in Table 4.5.

Table 4.5: CPPN unit generation parameters for a Monte Carlo analysis

Parameter	Minimum	Maximum
Seed	1	1000
Model ID	1	N/A
Scale	1	N/A
Number of hidden layers	2	10
Size of the initial hidden layer	2	32
Element removal threshold	0	100

Figures to represent the relationships between the parameters in Table 4.5 and the score as calculated in Section .

Chapter 5

Model Validation

Physical replicas of selected units are constructed and inflated. The deformation is observed and compared with the modelled behaviour as calculated by the FEM software.

5.1 Mixing and Casting

The modular mould as described in Section 3.6.1 is assembled according to the desired unit layout. Mold Star 15 is mixed and cast as per Section 3.1.3. After the cure time has passed the unit is removed from the mould and excess material is carefully cut off. Figure shows a filled mould.

5.2 Experimental Setup

The experimental setup as described in Section 3.6.1 is constructed. Figure shows the experimental setup with no unit in place. A sealant is used to prevent air leakage at the point of entry of the pressure supply tube.

5.3 Results

Three units were selected for model validation.

5.3.1 Unit 1

The first unit selected is grid_ as illustrated in Figure . This unit was selected as a random control unit. This unit does not perform well according to any specific unit deformation case. This unit has complex internal geometry that may deform in unusual ways. Cavities exist which are potentially inaccessible from the undeformed state, but become accessible as the unit inflates.

5.3.2 Unit 2

The second unit selected is grid_ as illustrated in Figure . This unit was selected as a linearly extending unit. This unit performs well according to the uniaxial deformation case. Cavities exist which are potentially inaccessible from the undeformed state, but become accessible as the unit inflates.

5.3.3 Unit 3

The third unit selected is grid_ as illustrated in Figure . This unit was selected as a linearly extending unit. This unit performs well according to the uniaxial deformation case when surrounded by identical units, but performs poorly when on its own.

5.4 Issues Encountered

Issues were encountered during model validation. Some were attempted to account for beforehand and some were unforeseen.

Mould cells were not manufactured exactly to tolerances.

Chapter 6

Conclusions

Appendices

Appendix A

Test

List of References

Aiguier, M., Le Gall, P. and Mabrouki, M. (2008). Emergent properties in reactive systems. In: *2008 15th Asia-Pacific Software Engineering Conference*, January 2008, pp. 273–280. Beijing. ISSN 14416638.

Behl, M. and Lendlein, A. (2007). Shape-memory polymers. *Materials Today*, vol. 10, no. 4, pp. 20–28. ISSN 13697021.
Available at: `http://dx.doi.org/10.1016/S1369-7021(07)70047-0`

Boyraz, P., Runge, G. and Raatz, A. (2018). An Overview of Novel Actuators for Soft Robotics. *Actuators*, vol. 7, no. 48, pp. 1–21. ISSN 2076-0825.

Brose, P. (1993). *Compositional Model-Based Design: A Generative Approach To The Conceptual Design Of Physical Systems.* Ph.D. thesis, University Of Southern California.

Buchanan, C. and Gardner, L. (2019). Metal 3D printing in construction: A review of methods, research, applications, opportunities and challenges. *Engineering Structures*, vol. 180, no. January, pp. 332–348. ISSN 18737323.

Cheney, N., Bongard, J. and Lipson, H. (2015). Evolving Soft Robots in Tight Spaces. In: *GECCO '15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 935–942. ACM, Madrid. ISBN 9781450334723.

Cheney, N., MacCurdy, R., Clune, J. and Lipson, H. (2013). Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding. In: *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference*, pp. 167–174. ISBN 9781450319638.

Cook, R.D., Malkus, D.S., Plesha, M.E. and Witt, R.J. (2002). *Concepts and Applications of Finite Element Analysis.* 4th edn. Wiley, Madison. ISBN 978-0-471-35605-9.

Damper, R.I. (2000). Emergence and levels of abstraction. *International Journal of Systems Science*, vol. 31, no. 7, pp. 811–818. ISSN 14645319.

Do, T.N., Phan, H., Nguyen, T.-Q.Q. and Visell, Y. (2018). Miniature Soft Electromagnetic Actuators for Robotic Applications. *Advanced Functional Materials*, vol. 28, no. 18, p. 1800244. ISSN 16163028.

Ellis, D.R. (2020). *Generative Design Procedure for Embedding Complex Behaviour in Pneumatic Soft Robots.* Ph.D. thesis, Stellenbosch University.

Elsayed, Y., Vincensi, A., Lekakou, C., Geng, T., Saaj, C.M., Ranzani, T., Cianchetti, M. and Menciassi, A. (2014). Finite Element Analysis and Design Optimization of a Pneumatically Actuating Silicone Module for Robotic Surgery Applications. *Soft Robotics*, vol. 1, no. 4, pp. 255–262. ISSN 21695172.

Groenwold, A.A., Stander, N. and Snyman, J.A. (1999). A regional genetic algorithm for the discrete optimal design of truss structures. *International Journal for Numerical Methods in Engineering*, vol. 44, no. 6, pp. 749–766. ISSN 00295981.

Hiller, J. and Lipson, H. (2012). Automatic design and manufacture of soft robots. *IEEE Transactions on Robotics*, vol. 28, no. 2, pp. 457–466. ISSN 15523098.

Hornby, G.S. and Pollack, J.B. (2001*a*). Evolving L-systems to generate virtual creatures. *Computers and Graphics (Pergamon)*, vol. 25, no. 6, pp. 1041–1048. ISSN 00978493.

Hornby, G.S. and Pollack, J.B. (2001*b*). The advantages of generative grammatical encodings for physical design. In: *Proceedings of the IEEE Conference on Evolutionary Computation, ICEC*, pp. 600–607.

Hsiao, J.-H., Chang, J.-Y. and Cheng, C.-M. (2019). Soft medical robotics: clinical and biomedical applications, challenges, and future directions. *Advanced Robotics*, vol. 33, no. 21, pp. 1099–1111. ISSN 1568-5535.
Available at: `https://www.tandfonline.com/action/journalInformation?journalCode=tadr20`

Huang, W.M., Yang, B., An, L., Li, C. and Chan, Y.S. (2005). Water-driven programmable polyurethane shape memory polymer: Demonstration and mechanism. *Applied Physics Letters*, vol. 86, no. 11, pp. 1–3. ISSN 00036951.

Ilievski, F., Mazzeo, A.D., Shepherd, R.F., Chen, X. and Whitesides, G.M. (2011). Soft robotics for chemists. *Angewandte Chemie - International Edition*, vol. 50, no. 8, pp. 1890–1895. ISSN 14337851.

Kim, N.H. (2015). *Introduction to nonlinear finite element analysis.* Springer, New York. ISBN 9781441917461.

Kolodziej, J. (2002). Modeling hierarchical genetic strategy as a Lindenmayer system. In: *Proceedings. International Conference on Parallel Computing in Electrical Engineering*, pp. 409–414. ISBN 0-7695-1730-7.
Available at: `http://ieeexplore.ieee.org/document/1115312/`

LSTC (). LS-DYNA | Livermore Software Technology Corp.
Available at: `https://www.lstc.com/products/ls-dyna`

Luis, E., Pan, H.M., Bastola, A.K., Bajpai, R., Sing, S.L., Song, J. and Yeong, W.Y. (2020). 3D printed silicone meniscus implants: Influence of the 3D printing process on properties of silicone implants. *Polymers*, vol. 12, no. 9, pp. 2136–2154. ISSN 20734360.

Martinez, R.V., Branch, J.L., Fish, C.R., Jin, L., Shepherd, R.F., Nunes, R.M.D., Suo, Z. and Whitesides, G.M. (2013). Robotic Tentacles with Three-Dimensional Mobility Based on Flexible Elastomers. *Advanced Materials*, vol. 25, no. 2, pp. 205–212.
Available at: `http://nrs.harvard.edu/urn-3:HUL.InstRepos:12388816`

Martinez, R.V., Fish, C.R., Chen, X. and Whitesides, G.M. (2012). Elastomeric origami: Programmable paper-elastomer composites as pneumatic actuators. *Advanced Functional Materials*, vol. 22, no. 7, pp. 1376–1384. ISSN 1616301X.

Metropolis, N. and Ulam, S. (1949). The Monte Carlo Method. *Journal of the American Statistical Association*, vol. 44, no. 247, pp. 335–341.

MSC (2003). MSC.Marc Mentat Datasheet.

Mutlu, R., Alici, G., Xiang, X. and Li, W. (2014). Electro-mechanical modelling and identification of electroactive polymer actuators as smart robotic manipulators. *Mechatronics*, vol. 24, no. 3, pp. 241–251. ISSN 09574158.

Niroomandi, S., Alfaro, I., Cueto, E. and Chinesta, F. (2010). Model order reduction for hyperelastic materials. *Internation Journal for Numerical Methods in Engineering*, vol. 81, pp. 1180–1206.
Available at: `http://onlinelibrary.wiley.com/doi/10.1002/nme.3279/full`

Ogden, R.W. (1972). Large deformation isotropic elasticity - on the correlation of theory and experiment for incompressible rubberlike solids. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 326, no. 1567, pp. 565–584. ISSN 2053-9169.

Onal, C.D., Chen, X., Whitesides, G.M. and Rus, D. (2017). Soft mobile robots with on-board chemical pressure generation. *Springer Tracts in Advanced Robotics*, vol. 100, pp. 525–540. ISSN 1610742X.

Pernkopf, F. and Bouchaffra, D. (2005). Genetic-Based EM Algorithm for Learning Gaussian Mixture Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1344–1348.

Prusinkiewicz, P., Lindenmayer, A., Hanan, J.S., Fracchia, F.D., Fowler, D., de Boer, M.J.M. and Mercer, L. (2004). *The algorithmic beauty of plants*. 1. Springer-Verlag, New York.

Rieffel, J., Knox, D., Smith, S. and Trimmer, B. (2013). Growing and Evolving Soft Robots. *Artificial Life*, vol. 20, no. 1, pp. 143–162. ISSN 1064-5462.

Rodriguez, J.N., Zhu, C., Duoss, E.B., Wilson, T.S., Spadaccini, C.M. and Lewicki, J.P. (2016). Shape-morphing composites with designed micro-architectures. *Scientific Reports*, vol. 6, pp. 1–10. ISSN 20452322.
Available at: www.nature.com/scientificreports/http://dx.doi.org/10.1038/srep27933

Runge, G., Wiese, M., Gunther, L. and Raatz, A. (2017). A framework for the kinematic modeling of soft material robots combining finite element analysis and piecewise constant curvature kinematics. In: *2017 3rd International Conference on Control, Automation and Robotics, ICCAR 2017*, pp. 7–14. ISBN 9781509060870.

Sekhar, P. and Uwizeye, V. (2012 jan). Review of sensor and actuator mechanisms for bioMEMS. *MEMS for Biomedical Applications*, pp. 46–77.
Available at: https://www.sciencedirect.com/science/article/pii/B978085709129150002X

Shea, K., Aish, R. and Gourtovaia, M. (2005). Towards integrated performance-driven generative design tools. *Automation in Construction*, vol. 14, pp. 253–264. ISSN 09265805.

Shepherd, R.F., Ilievski, F., Choi, W., Morin, S.A., Stokes, A.A., Mazzeo, A.D., Chen, X., Wang, M. and Whitesides, G.M. (2011). Multigait soft robot. *Proceedings of the National Academy of Sciences of the United States of America*, vol. 108, no. 51, pp. 20400–20403. ISSN 00278424.

Siemens (). Announcing the Next Generation Design Platform: NX 12 | NX Design.
Available at: https://blogs.sw.siemens.com/nx-design/announcing-the-next-generation-design-platform-nx-12/

Sims, K. (1994*a*). Evolving 3D Morphology and Behavior by Competition. In: *Artificial Life IV Proceedings*, pp. 28—-39.

Sims, K. (1994*b*). Evolving virtual creatures. In: *SIGGRAPH '94: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pp. 15–22. ISBN 0897916670.
Available at: http://www.karlsims.com/papers/siggraph94.pdf

Smooth-On (2014). Mold Star 15, 16 and 30.
Available at: http://www.smooth-on.com/tb/files/MOLD{_}STAR{_}15{_}16{_}30{_}TB.pdf

Stanley, K.O. (2006). Exploiting regularity without development. In: *Proceedings of the 2006 AAAI Fall Symposium on Developmental Systems*, pp. 49–56. ISBN 1577353013.

Villoslada, A., Flores, A., Copaci, D., Blanco, D. and Moreno, L. (2014). High-displacement flexible Shape Memory Alloy actuator for soft wearable robots. *Robotics and Autonomous Systems*, vol. 73, pp. 91–101. ISSN 09218890.

Wang, Z., Chen, M.Z.Q. and Yi, J. (2015). Soft robotics for engineers. *HKIE Transactions*, vol. 22, no. 2, pp. 88–97. ISSN 2326-3733.
Available at: `http://dx.doi.org/10.1080/1023697X.2015.1038321`

Whitesides, G.M. (2018). Soft Robotics. *Angewandte Chemie - International Edition*, vol. 57, no. 16, pp. 4258–4273. ISSN 15213773.

Yan, J., Zhang, X., Xu, B. and Zhao, J. (2018). A New Spiral-Type Inflatable Pure Torsional Soft Actuator. *Soft Robotics*, vol. 5, no. 5, pp. 527–540. ISSN 21695180.

Zakinthinos, A. and Lee, E.S. (1998). Composing secure systems that have emergent properties. In: *Proceedings of the Computer Security Foundations Workshop*, pp. 117–122. ISBN 0818684887. ISSN 10636900.