Low-Cost Instrumentation for Mechanical Testing of Structural Elements of Soft Systems

Thesis by

Ashley K Turza

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Mechanical Engineering

Tufts University

Medford, Massachusetts, USA

2017

(Defended April 20, 2017)

Advisor Prof. Jason Rife

Reviewer Prof. Eric Tytell

Department of Biology

Reviewer Prof. Robert White

Department of Mechanical Engineering

Acknowledgments

This thesis would not have been possible without the support of a number of outstanding people. First, I would like to acknowledge the National Science Foundation and the IGERT program for funding support under Grant No. 1100452 and DBI-1126382.

I would like to thank my thesis committee. Chief among these is my advisor Professor Jason Rife, who is not only supportive and encouraging in academic matters, but has also deftly managed all of that with a student slightly prone to sticky situations. He successfully managed to get me on a successful research track while I was literally lying on a hospital bed within the first week of my graduate career, and somehow continued to exceed that high bar ever since. I would also like to thank my other department committee member Professor Robert White, who has always been willing to talk me through modeling problems and is one of the best teachers I have ever had. Finally, I would like to thank my cross-department committee member, Professor Eric Tytell, for agreeing to both serve on my committee and be a secondary advisor. His willingness to explain things from a biological background to an engineer has been invaluable in thinking about how interdisciplinary my chosen field is and the critical importance of a common set of tools to work with.

Thank you to the members of the ASAR lab who have graciously put up with me excitedly showing them squishy blocks and then asking the questions I haven't

thought of about them. It has been a pleasure working with all of you and every seemingly trivial question or observation has been invaluable. Thank you to the members of the CRISP lab and other members of the IGERT program as well, for making my graduate experience what it is.

Finally, thank you to my friends, especially to my housemates Ray Speth, Alejandro Sedeño, and TB Shardl for putting up with me throughout all of this, for bouncing ideas off of, helping with Python debugging, explaining best coding practices, and reminding me to drink my coffee and not die. And thank you to my family, to my younger brother for quiet encouragement, and to my parents, who always encouraged and supported my academic pursuits, from the time I was a little kid playing with LEGO to now when I'm a big kid playing with not-quite LEGO. I never would have made it here without you.

Abstract

This thesis describes the design of a low-cost system of tests to characterize mechanical properties in soft structures. More specifically, we focus on testing mesoscale structural components (which we call structural volume elements) from which a larger structure might be constructed. The mechanical properties of a structural volume element are determined both by its material properties and by its geometry. The testkit provides an easy-to-implement means of characterizing force-displacement relationships (e.g. compression, shear, bending) for nonlinear materials in the presence of large deformations and buckling instabilities. The testkit includes both hardware and software components. The purpose of the hardware is to apply loading conditions to a small structural volume element, approximately 1 cm^3 in volume. The purpose of the software is to measure the displacements that occur as loading changes.

Contents

List of Figures

List of Tables

Nomenclature

The following is a list of common abbreviations found in the text:

FEA Finite element analysis

OpenCV Intel Open Computer Vision Performance Library

RGB Red, green, blue

ROI Region of interest

PDMS Polydimethylsiloxane $\mathrm{CH_3[Si(CH_3)_2O]_nSi(CH_3)_3}$

CHAPTER 1

Introduction

Work has been done on making soft robotics more accessible to researchers, STEM students, and the general public, most notably the Soft Robotics Toolkit [3], but analytic models backed by experimental data are still required [4]. Furthermore, if exact solutions to models exist for non-linear continuum soft robotics, they are computationally expensive and thus require some sort of simplification [5]. Combined, these can form a formidable barrier to entry in the field for those who lack the tools or computation power for these methods. This is a particular problem in the soft robotics field because it is incredibly interdisciplinary and the traditional tools and methods used to characterize and simulate behavior might not be readily available within the discipline. For example, a biology-focused research group might not have access to the tools mechanical engineers have for material testing.

1.1 Background

At a fundamental level, a major problem is the lack of tools. Advances in all scientific fields are dependent on the development of innovative methodologies and tools. Soft systems in particular are challenging to work with because of their nature; the morphology and material properties (and sometimes geometric properties) of these structures make their behavior difficult to predict, especially when many of

them undergo large deformations. Due to this, the field of soft robotics in general has to wrestle with problems of a very different nature than their traditional counterparts.

1.1.1 Soft Robotics

The traditional field of robotics focuses on rigid structures and precise control schemes. This limits their ability to adapt to environmental and other situational obstacles, which is a challenge in areas where it would be far safer for robots to be deployed, such as rescue and recovery operations. Consider: in a manufacturing environment, a robot's actions can be pre-determined and programmed exactly. In the outside world, the environments include those that are too hazard for humans to operate safely and in areas where it is too hazardous for a hard robot to operate safely, such as mobility assistant robots in populated areas. Considering developments on self-driving cars by Google and other DARPA projects, the computational requirements for the latter scenario are possibly far greater than what is realistically feasible. The field of soft robotics, on the other hand, chooses to look at these problems in a different way. Often inspired by biology, soft robotics chooses to exploit the form and compliance of softer structures to expand adaptability [6]. Recent trends in this direction include simplifying complicated neurological control through morphological design and materials [7]. This adaptability reduces the danger to both robot and surroundings as well as mitigating the need for absolute predictability or sensor computation.

The field is growing and provides exciting new opportunities and advancements. Like all fields, however, there are also significant challenges, many of which stem from the very materials used to create these systems. Soft robotics are built using a variety of different manufacturing processes, from traditional lithography to 3D printing technologies to shape-memory alloy embedding [6] [8]. The overriding feature of all of these technologies is that the stiffness of the materials remains va-

riable, able to contort based on the environmental factors and loading requirements on each piece. The more adaptive the robot is, the greater the control scheme and kinematics of the robot are, if designed in the traditional manner. The dynamics of soft materials provide a large computational challenge simply due to the number of degrees of freedom and the nonlinear effects of the materials [4]. Indeed, Kim et al have noted that the heterogeneous structures of soft robots often also have complex boundary conditions that make dynamic modeling difficult, leaving the common approach limited to kinematic analysis [9]. Other approaches have been largely theoretical. Previously, morphological computation simulations have proposed a mass-spring system for modeling [10] and voxel lattices [11]. It is from this theoretical basis that we look further into soft material structures that exploit this analysis.

1.1.2 Material Properties

The materials most commonly used in soft robotics are usually polymers, specifically elastomers [12], which are excellent at deformation. Elastomers specifically form nearly-linear chains with only occasional cross-linking. It is these cross-links that act as a “memory” for the material, allowing it to return to its original shape after loading. The problem however is that the properties of these materials are highly dependent on the fabrication process and chemical formulation of the polymer chains. Polymer chains can vary in length, but also the mechanical processing can change the molecular branching and crystallization. These, in turn, have profound impacts on the material properties [13] .

One of these is Young’s modulus. By definition, Young’s modulus concerns the stress-strain relation of a bar of isotropic elastic material under either compression or tension loading. It is necessary for compliance matching [14] and for predicting the behavior of a material, and thus structure, under different loading conditions.

For polymers, the moduli are much lower than the rigid materials traditional robots are made out of, such as metal, often by at least two orders of magnitude [15]. All of these are critical design questions for any type of robot, and soft robots in particular. However, since the properties are so highly dependent on the individual case, accurate modeling can be difficult.

From an engineering perspective, to design while treating material properties as a “black-box” represents a significant challenge. Models and simulations require these properties as inputs in order to predict behavior. How then can materials with such variability be utilized in the most effective manner and in such a way that allows engineers and designers to use the tools and methods they are accustomed to? However, robotics and soft robotics in particular are multi-disciplinary fields and examining the problem from a chemical and material standpoint offers an alternative viewpoint for design purposes.

1.2 Motivation

Our initial forays into designing soft robotic structures were stymied by a simple problem: there is a lack of material property data and analytic models available to fully characterize a system in simulation. While rapid prototyping costs have dropped in recent years and manufacturers sell multi-material-deposition 3D printers off-the-shelf [16], a completely iterative design process is still an expense smaller research labs might find prohibitive. Analysis of soft-structures with nonlinear material properties undergoing large deformations through instabilities (e.g. buckling) remains challenging to implement in simulation, using FEA for example. Modeling these structures often requires assumptions to be made to properly bound the problem such that a solution can be found. While better models could have been used, the question is it more efficient to test a macrostructure experimentally and derive

a model from that, given the accessibility of rapid prototyping?

Soft robotics is an interdisciplinary field and thus cannot assume that all research groups in the field are equal. Expensive commercial engineering tools such as those for material testing, such as those provided by Instron, and FEA software packages such as ANSYS or COMSOL, might not be reasonably accessible, so there has to be a low-cost alternative. To this end, it is helpful to look at data sharing in other fields such as clinical health [17] and in the open-source research paradigm [18]. The rise of not only open-source software but low-cost rapid prototyping 3D printers and online repositories offers an opportunity to provide viable low-cost alternatives to expensive commercial tools.

Not only is there a benefit in providing a low-cost tool alternative (e.g. to researchers who don't have access to an Instron or expertise in complicated FEA simulation), there is a reason why these tools do not exist yet. Traditional engineering materials are rigid and require massive loads to deform. Interest in soft materials is much more recent and these materials deform massively under much smaller loads. Thus, there is a mismatch between the traditional equipment designed for rigid materials and the newer interest in soft structures.

The motivation for this project is thus two-fold: to create a low-cost testing apparatus for non-linear soft structural elements with unknown geometric and material properties and to apply these tests to a particular structural element. These structural elements will form a library of building blocks from which larger soft structures might be synthesized in order to embody desired kinematic or dynamic behaviors. By using these building blocks, we can abstract away the specific material and geometric properties, condensing the block into a "black-box" we can use more easily in a model. We will define these soft elements as structural volume elements (SVE).

1.3 Contribution

This thesis documents two major contributions towards solving this problem:

- Design a low-cost testing apparatus to measure the mechanical properties of a structural volume element undergoing large deformations
- Design a particular structural volume element and characterize its force-displacement properties using the testing apparatus, as a means of demonstrating how a broader library of SVEs might be created in the future.

The contribution of a low-cost apparatus is significant because the space lacks an easily-accessible toolkit for performing such experimental analysis [4]. Additionally, the design of an SVE is necessary to test both the functionality of the device and, more importantly, because it is an example of the potential building blocks that could be created. The SVE we designed for this was a $1cm^3$ block that would be stiff in some directions and soft in others, in order to obtain a broad sense of possible SVEs that could exist in a library.

1.4 Thesis Structure

Chapter 2: Designing a Soft Robot as a Lattice of Structural Volume Elements Chapter 2 specifically focuses on the design and testing of the structural volume element we designed to be the basis of a lattice model. 2.3 discusses the design of the cell.

Chapter 3: A Toolkit for Testing SVE Force-Displacement Here we discuss the design of the hardware and software used in the testkit. 3.1 goes over the hardware design of the test rigs for each of three categories of testing: compression, shear, and bending. 3.2 discusses the software algorithms developed to

extract and process the data taken from photographs of the tests, with consideration for the filter choices in the image processing.

Chapter 4: Experimental Results The experimental results of a representative SVE under compression, shear, and bending are discussed. Compression results show preferred bending directions, as do bending results. Shear results show differences in deformation based on orientation.

Chapter 5: Conclusion

CHAPTER 2

Designing a Soft Robot as a Lattice of Structural Volume Elements

2.1 Introduction

Because of their material properties and tendencies towards non-linear behavior, soft materials are much more difficult to model than their more rigid counterparts. This in turn makes control a difficult problem if the kinematics of the system are undefined. Per Pfeifer [19], there must be a way to take advantage of embodiment present in the natural world for effective biologically-inspired systems that make use of these soft materials and morphology. Previous approaches to control have included paradigms for the inverse kinematics inspired by octopus arms [20] [5], treating the arm as a single unit, and Saunders' work on underactuated feedback systems and passive dynamics inspired by caterpillars [21] [22]. Because these approaches consider the whole it is difficult to create a general model for soft structures outside of an exact geometry.

The proposal here is a reversal of that paradigm: instead of the conventional approach to design a structure and analyze its kinematics, what if one were able to define the desired kinematics and then synthesize a structure that embodies these kinematics? Our approach to this is to break the system down into smaller structures that can be analyzed individually and then assembled into larger structures to obtain the desired properties and behaviors.

The concept of breaking structures into smaller elements is not new. Children's toys have even used modular elements to create robotic systems, such as the Capsela system from the 1980s which provided hard spheres containing actuators and electronics. Unsurprisingly, this concept has resurfaced in the soft robotics field. Lee and Kim [23] developed pneumatically actuated soft robot modules, able to be assembled in order for rapid prototyping. Our goal is a finer-grained extension of this concept, whereby instead of having separate modules for different functions, we create a single element as a building block that can be combined in different configurations.

2.2 Lattices

To define these building blocks for a soft structure, the gap between the smallest scales and the largest needs to be bridged. Our concept is to create mesoscale blocks that can be assembled in repeated patterns (e.g. a lattice); in turn, those lattice structures would be combined into a larger robot (Fig. 2.1). Since we are borrowing the concept of "lattice" from materials science, it is worth taking a brief look at how the term is used at the microscale before adapting it to the mesoscale.

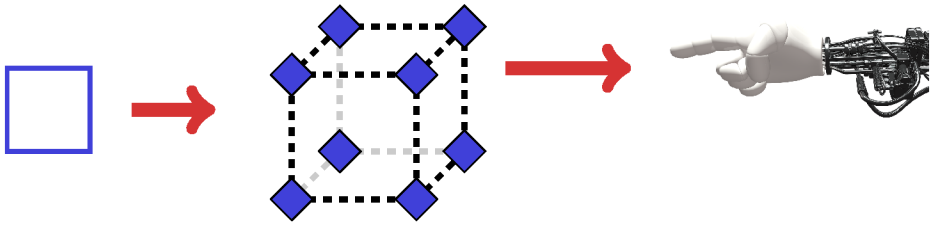


FIGURE 2.1 From SVE to robot

2.2.1 Lattices in Crystallography

Using the methods and nomenclature from crystallography gives us a framework in which to understand using lattice structure on a macro-scale. We start from the atomic level in this metaphor.

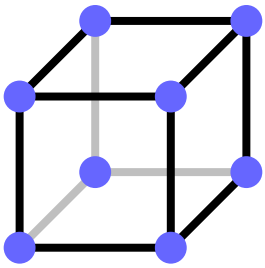
Lattice theory is used both in crystallography and geometry. The most common example of lattice structures are the 14 Bravais lattices, defined as an infinite array of discrete points in 3D space as generated by

$$\mathbf{R} = n_1\hat{\mathbf{a}}_\mathbf{1} + n_2\hat{\mathbf{a}}_\mathbf{2} + n_3\hat{\mathbf{a}}_\mathbf{3} \tag{2.1}$$

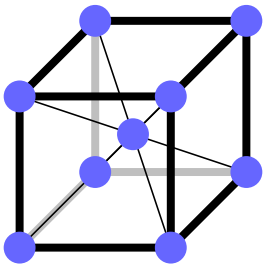
where $\hat{\mathbf{a}}_\mathbf{i}$ are the unit vectors spanning the lattice basis, n_i are integers, and $\mathbf{R}$ is the position vector. For the sake of example, we will look at the cubic Bravais lattices, seen in Fig. 2.2, although the theory expands to further lattice shapes.

Physical properties of materials depend on the atomic structure. Particularly, the Young's modulus is especially dependent on atomic packing and the inter-atomic bonds. The latter can be thought of as small springs between individual atoms (Fig. 2.3), which will be useful in considering structures as a mass-spring matrix, covered further in Chapter 2. At this point, it is useful to consider the implications of treating atomic structures as discretized elements, assembled into lattices to form a solid structure.

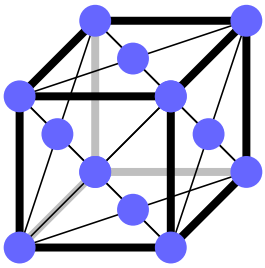
From crystallography, the Cauchy-Born approximation relates the movement



(a) Unit cubic structure



(b) Body-centered cubic (BCC) structure



(c) Face-centered cubic (FCC) structure

FIGURE 2.2 Cubic crystalline structures

of atoms in a crystal to the overall deformation of the solid. Thus, when a crystalline solid undergoes strain, the individual atoms within the crystal lattice also undergo the overall strain. The Cauchy-Born approximation holds for face-centered and body-centered cubic systems, as defined by Bravais lattice nomenclature.

Thus, it is possible to relate the individual atomic bonds to a simple application of Hooke's law in order to get material properties such as Young's modulus and the stress-strain relation:

$$F = k_0(x - x_0) \tag{2.2}$$

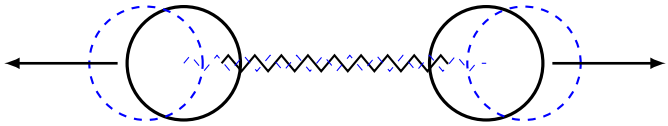


FIGURE 2.3 Atomic bond as spring

If N as the number of units per area, equal to $\frac{1}{x_0^2}$ assuming cubic symmtry, we can convert the displacement to stress σ

$$\sigma = Nk_0(x - x_0)$$

From here, we use strain ϵ_n using the initial spacing x_0 in order to get the Young's modulus E for uniaxial loading

$$\sigma = (\frac{k_0}{x_0})\epsilon_n$$

$$E \approx \frac{\sigma}{\epsilon_n} = (\frac{k_0}{x_0}) \tag{2.3}$$

This is the usual method of determining the property based on material alone.

Eq. 2.2 also applies at a macroscopic level, while also providing the power to condense unknowns into a single variable. It is a more generalized equation that can be used to convey both structure and material into the single variable k_0 which is very useful in the black-box modeling, as it reduces the complexity to a mass-spring system.

2.2.2 Meso-scale Lattices

Physical properties analogous to the Poisson's ratio, Young's modulus, stiffness, and shear modulus can be obtained on a macro-scale networks [11]. Force-displacement properties for macro-scale bodies, such as a full-size robotic limb, can be derived from their microscale properties and their geometry. Although ana-

lytic results can sometimes be obtained for simple geometries, force-displacement relationships for arbitrary bodies are more typically obtained using computational methods, such as finite-element analysis [24].

In order to simplify synthesis of soft structures with desirable force-displacement properties, we propose to introduce a mesoscale structure between the microscale (where the body's material properties are defined) and the macroscale (where the body's full form is defined). In terms of fabrication, a macroscale part is a body that can be fabricated or analyzed in one pass; it is the entire form of the robot. The mesoscale refers to sub-elements of a part, where the full part would be constructed as a lattice of mesoscale structures whose properties can be tuned. (Fig. 2.4). Conceptually, a mesoscale structure is analogous to a LEGO brick (which has variable properties, such as color or studs) and the macroscale structure is the completed set. This would mean that the mechanical properties of the macro part could be tuned locally to define desired patterns of deformation at the macroscale as an accumulation of local deformations of individual mesoscale structures within the part.

We will call this mesoscale structure a Structural Volume Element (SVE). A repeated array (or lattice) of SVEs can be constructed to synthesize a macroscopic geometry with desired force-displacement properties. Moreover, the force-displacement properties at the macroscale become highly tunable if the force-displacement properties of the SVEs can be made tunable.

The model, then, is closer to a finite-difference one. Lattice spring models (LSM) such as these discretized continuum elastic materials, consisting of a 2D or 3D network of one-dimensional springs [25]. Furthermore, these 1D springs can be modeled using simple Hookean linear springs, thereby linearizing a large deformation problem. In these, the spring constant (or the stress-strain relation) are the key factors, modeling the nodes as point masses, where the geometry of the node itself

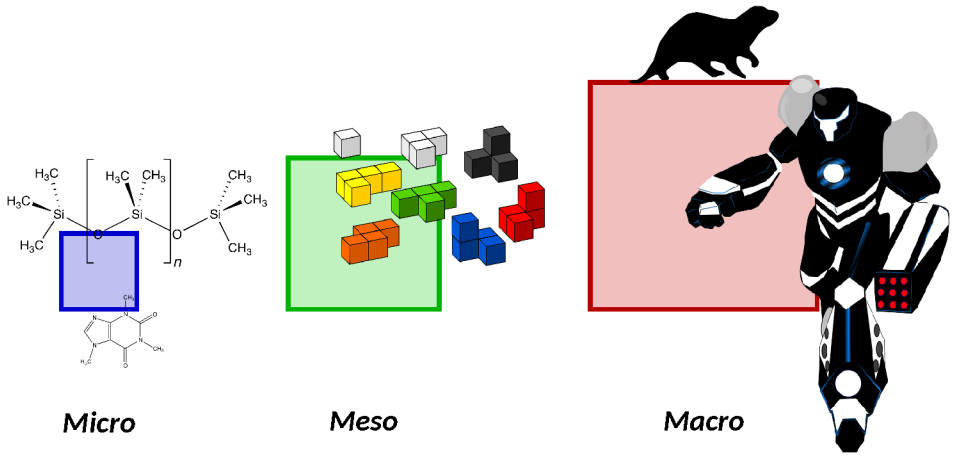
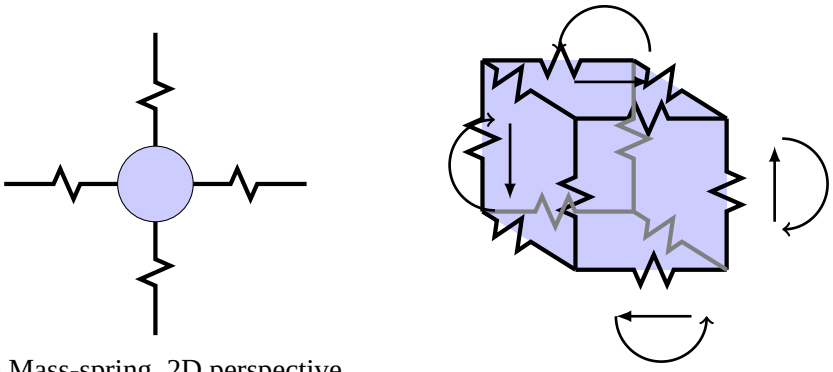


FIGURE 2.4 Scaling domains

plays no real role beyond providing a mass. The spring model maps to continuum equations using the Lamé equations and Taylor approximation [25]. Mathematically, the model works and is used in many applications.

In order to define the stiffness relating nearby nodes (e.g. nearby SVEs), it is important to consider both material and geometric properties of the nodes, which can be leveraged to influence the stiffness and relative hardness of the elements themselves. Jennet et al recently developed a “discrete lattice assembly, in which modular 2D elements mechanically link in 3D” to form lattice structures [26]. This approach uses high-stiffness materials, and the 2D shape allows the relative hardness of the behavior to be a function of the lattice geometry. Furthermore, previous work has been done on flexible honeycomb structures [27], which provides the basis for lattice-based construction. These structures can be black-boxed into 2D or 3D mass-spring units, much like the atomic scale forces.

Our approach takes something of a middle ground between these points. Lipson suggests ”the advantage of using a particle-based approach for simulating soft structures is the ability to easily incorporate new, nonlinear, and active elements such as actuators, contacts, and arbitrary reactive materials” [4]. At the time, we are mostly focused on the creation of a soft structural element that can simulate large-scale



(a) Mass-spring, 2D perspective

(b) Block-spring forces

FIGURE 2.5 Mass-spring vs SVE model diagrams

deformations and has different stiffness coefficients that depend on the orientation of the element in relation to an applied force.

2.3 The Structural Volume Element

2.3.1 SVE Concept

The main design consideration for the SVE is to create a block that has known and controllable deformation along certain axes, while resisting other forms of movement along others. In doing so, the SVE would then be able to be combined with other elements in lattice structures where the force-displacement behavior could be modeled.

To this end, the SVE should be able to be defined as a mass and set of springs, one pair for each axis (Fig 2.5a), neglecting the effects of damping to keep the system linear. One would be able to model complex dynamical systems made of these soft lattices as a system of one-dimensional linear springs (Fig 2.6), using the mathematics of the LSM.

Relying solely on the properties of the material the SVE is formed from to provide the variable stiffness dependent on orientation constants is impractical due to the inherent variations in elastomer material properties (described in more detail in

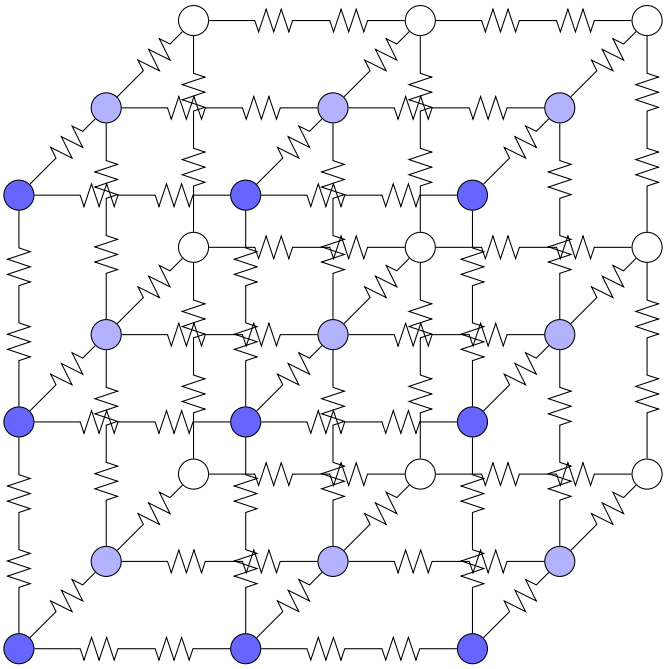


FIGURE 2.6 3D spring lattice structure

Appendix. D. To achieve the requirement, the SVE is dependent on its own geometry in addition to material. Doing so requires using the geometry to control and exploit buckling behavior under loading conditions. The specifications of SVE then are that it cannot suffer a complete collapse under reasonable loads nor can it not provide adequate deformation.

The exploitation of buckling behavior is a critical component. There are cases where it would be desirable to have nonlinear force-displacement relations in structures. By exploiting buckling, it is possible to make structures that have very low stiffness in a certain range of displacements or loads associated with buckling, but have a higher stiffness outside that range. The SVE is then akin to a meso-scale metamaterial, where we can design these structures to be tunable in order to have the loading behavior necessary.

2.3.2 An SVE Example

As a starting point to investigate the concept of designing tunable SVEs, I designed a particular SVE geometry to achieve anisotropic stiffness; specifically, the SVE was designed to have low stiffness along the z-axis, moderate stiffness along the x-axis, and high stiffness along the y-axis. (Figs. 2.9 and 2.10) Additionally, we wanted to examine the resistance to torsion. It is also easily combined into bi-directional lattice structures as seen in Fig. 2.11.

The SVE uses the Cartesian axis in Fig. 2.7. The axes referenced from this point will refer to this.

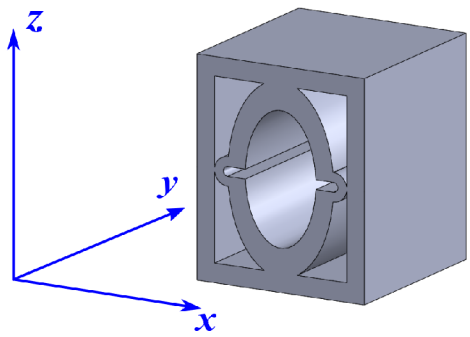


FIGURE 2.7 SVE axis reference

The concept for meeting the design requirements was to achieve anisotropic stiffness by creating an interior structure with an ellipsoidal cross-section. The structure is designed to buckle easily in the plane of the ellipse but to be stiff in the axis normal to that cross-section. The ellipsoid interior bracing of the hollowed out rectangular prism forces the unit cell to deform in the desired ways when under load. The ellipsoid here is in practice two arches mirrored. The notches on the sides of the ellipse cause the structure to favor buckling along the z-axis while retaining stiffness in the y-direction.

Furthermore, the elastic stability and buckling loading paths of arches are well understood. [28] [29] [30] [31] [32] The design can assume a pin-ended parabolic arch structure in one orientation, with a shallower arch in another. The arch design takes advantage of any symmetric prebuckling by centralizing the load. The deep arch can use the classical buckling theory to determine load, but in the case of the shallow arch, the classical theory cannot correctly predict buckling conditions, as determined by Pi and Bradford [33].

The buckling behavior desired is to avoid the classical anti-symmetric bifurcation buckling mode. The structure of the SVE changes the loading condition from point load to the distributed load, seen in Fig. 2.8. Moon et al [30] reported the equilibrium equations for pin-ended shallow parabolic arches under uniform compression, also noting that the thresholds for buckling modes are reliant on the stiffness of the material. Since the classical and modified theories require the Young's modulus, we cannot model the exact behavior of the SVE *a priori*. However, since the intention of the SVE is to create a modular element that relies on both material and geometry for deformation behavior, the arch structure provides a tunable element within the geometry parameter.

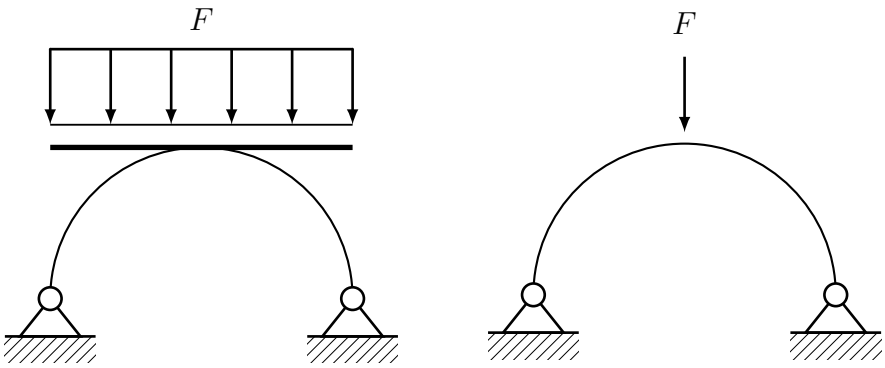


FIGURE 2.8 Arch loading

Our example SVE was printed on a Connex Objet printer using the TangoPlus build material. TangoPlus is a proprietary elastomer from Stratasys (the material safety sheet can be found in Appendix C). This build material is similar to other

elastomers used in soft robotics such as polydimethylsiloxane (PDMS) that have material properties that are difficult to characterize due to the cross-linking behavior during fabrication. Thus, the TangoPlus build material is an excellent choice as a test material if we wish to characterize the properties of SVEs irregardless of material or geometry.

Each element is 10mm $\times$ 10mm $\times$ 15mm and can be seen in Fig. 2.9; the printed version with a ballpoint pen for size reference is shown in Fig. 2.10. These dimensions were chosen as they were within the tolerance of the Objet printer while still remaining meso-scale structural elements.

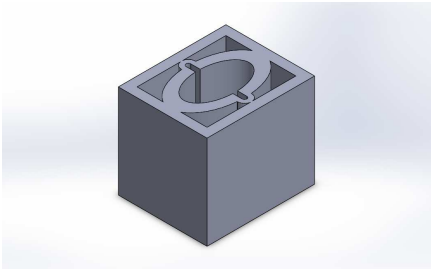


FIGURE 2.9 CAD model of the SVE

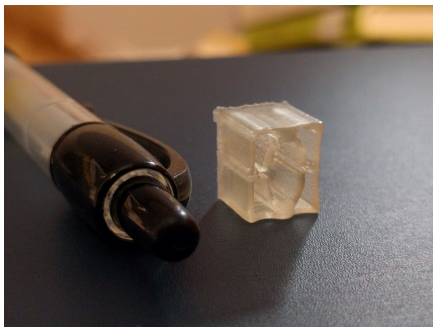


FIGURE 2.10 3D printed SVE

This SVE design also fulfills the requirement that it be able to be combined with others to form a lattice structure that can then be characterized. An example of a bi-directional mesh structure using two rows of six SVEs can be seen in Fig. 2.11. This structure can also be printed out on the same Objet 3D printer, although structures greater than 1 SVE deep proved more difficult to clean after printing.

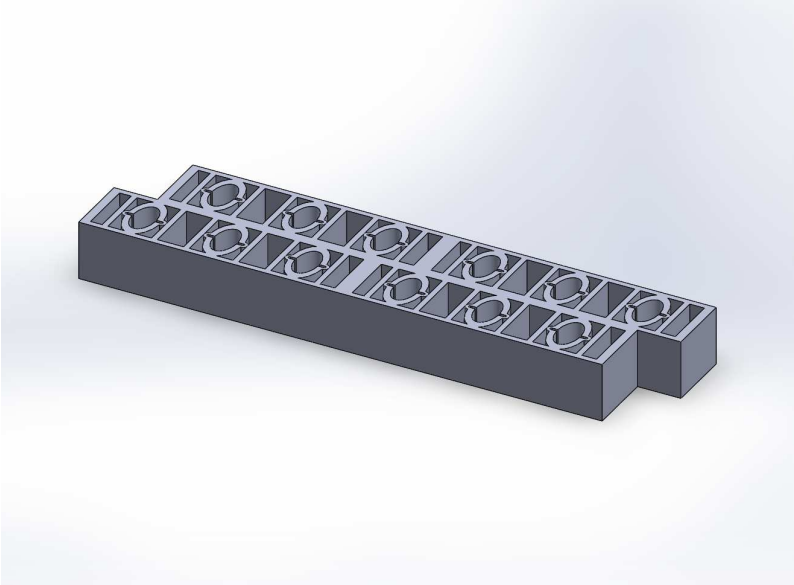


FIGURE 2.11 Example of bi-directional mesh structure using the cells

CHAPTER 3

A Toolkit for Testing SVE Force-Displacement

Measuring the force-displacement of small, soft structures can be expensive, time-consuming, or both. In order to design a macroscale stucture from SVEs at the mesoscale, however, a means of characterizing force-displacement relationships for SVEs is needed. This chapter introduces a low-cost set of test apparatuses for measuring non-linear force-displacement relationships for an SVE. The first section of the chapter will discuss the hardware component of the test apparatus, which applies known forces to an SVE to obtain an unknown displacement. The second section of the chapter will discuss the software component used to record and analyze force-displacement data.

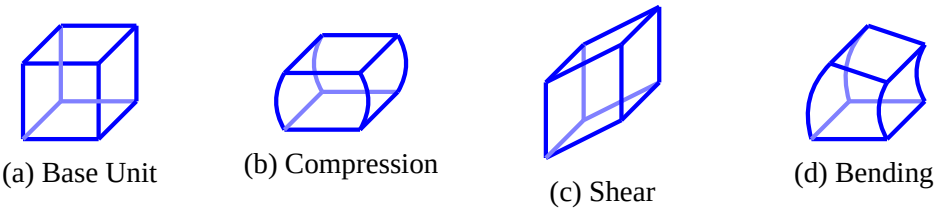
Assessing force-displacement characteristics computationally is difficult for soft-material structures undergoing large deformations. (Some observations about simulation approaches are presented in Appendix D.) By comparison, the proposed experimental approach is inexpensive, quick, and easy to implement with minimal training.

3.1 Hardware Design: Test Fixtures

To obtain force-displacement properties for SVEs via experimentation, I had to design test fixtures appropriate for the design goals. These design goals include:

- For use with small loads on the order of less than 2N and deformations on the order of millimeters
- For measuring relatively large force-displacement ratios
- Create a low-cost method of determining force-displacement characteristics of an SVE

To facilitate the third design goal specifically, the test fixtures were designed to be possible to fabricate on even a benchtop 3D printer. Three categories of properties are tested: compression, shear, and bending, seen in Fig. 3.1. Although other types of testing are relevant (e.g. torsion, tension), development of apparatus to test additional types of loading is left to future work.



(a) Base Unit (b) Compression (c) Shear (d) Bending

FIGURE 3.1 Property tests

The weights used were US pennies minted after 1983. The US Mint requires all pennies minted after mid-1982 to weigh 2.5 grams [34], which provides us with a very accessible approximate standard mass to use in our measurements for very low cost. We can thus increase the load by $2.5\text{g}\times g$ (with $g = 9.81 \times 10^{-3} m/s^2$) per increment. While we could get much more precise measurements with calibrated weights, doing so will increase the overall cost of the hardware; additionally, these approximate weights are a recoverable cost.

All test fixtures were printed out of black ABS plastic on a Stratasys Dimension 3D printer. Pulleys, fishing line, and a 1/8” diameter plastic rod (McMaster-Carr) were cut to size. A reference swatch of 10mm $\times$ 2.5mm was painted in white acrylic (Testors, Item 286128) on each test structure.

In order to spread loads evenly over the SVE surface, the SVE is adhered to a pair of flat plates prior to testing. For our implementation, the plates were two 40 x 40 mm 1/16” acrylic sheets (McMaster-Carr) laser-cut to the appropriate size, with two incident edges painted red with Testors acrylic (Item 286126). Incident edges were chose to eliminate the risk of errors due to refraction in the acrylic, which could cause false-positives in the color-detection algorithm.

3.1.1 Compression Test

The compression test structure consists of a main body that can fit the test piece into the base and a chute that aligns pennies with the center of the test piece. The base of the structure contains a cavity that is 44mm $\times$ 45.3mm $\times$ 16mm; the chute is 21mm $\times$ 33.30mm, which is just wide enough for a US penny which has a diameter of 19.05mm. A series of five sliding latches are included to provide additional support and alignment for the penny loading, as seen in the CAD model in Fig. 3.2.

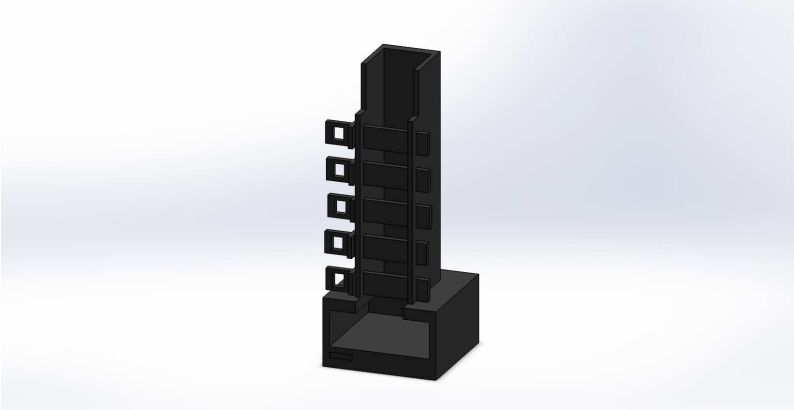


FIGURE 3.2 CAD model of compression test structure

The latches are easily removable and require very little force to slide into place, reducing the amount of misalignment during insertion (Fig. 3.3). The latches are inserted once the height of the penny stack reaches level with the bottom edge of each opening. The apparatus and camera are placed such that the white reference swatch and both parallel plates are clearly within frame while being as close as possible to the lens in order to reduce perspective error and background noise. The positions are then marked on the surface of the table with spike tape so as to minimize alignment changes between data points.

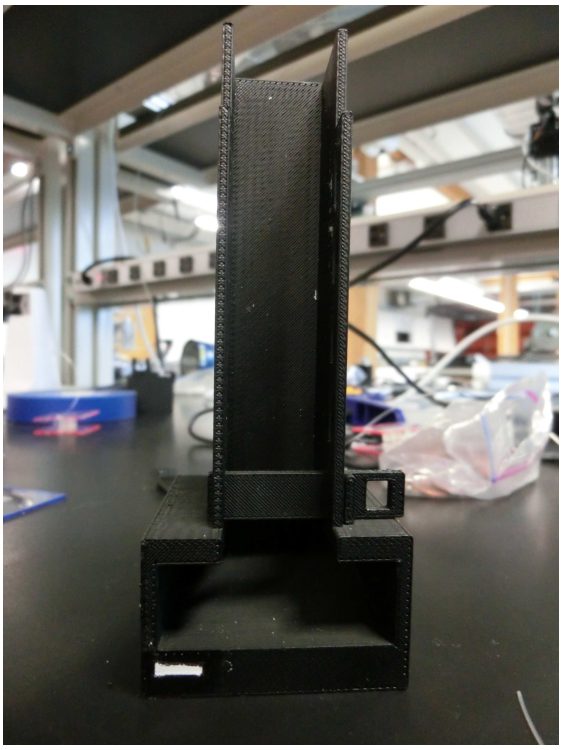


FIGURE 3.3 3D printed compression test structure

In our experimental run, we loaded the test piece with 46 pennies, applied individually. After each increment, we used a Casio EX-ZR400 HD camera to photograph the change in displacement. The results are discussed in Chapter 4.

3.1.2 Shear Test

The shear test fixture is designed to hold the test piece vertically by one of the two plates, leaving the other one with freedom of movement. One plate is locked into place using 3 of the 4 slide locks. The other plate is attached via fishing line (McMaster-Carr) through the top hole that is run over two pulleys and connected to a small bag that can hold the weight. The pulleys are positioned to provide a vertical force upwards on the plate, and by using fishing line, we ignore the effects of friction in the pulleys. The bag used in testing is a small canvas drawstring bag, adding a total of 9.8g to the total penny weight. The bag is left to hang freely.

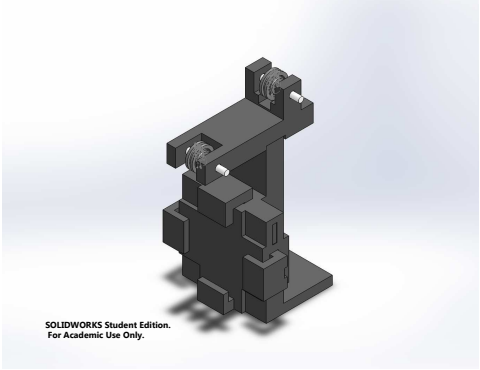


FIGURE 3.4 CAD model of shear test structure

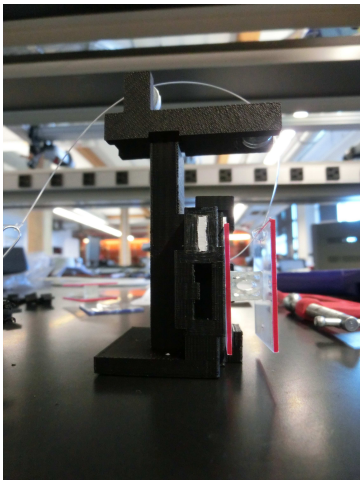


FIGURE 3.5 3D printed shear test structure with piece

The camera is positioned to get the white reference swatch and the two plates in-frame. The camera then remains in place for each test run (Fig. 3.5 and 3.6). Once again, the position of the camera is marked. The shear test fixture is clamped to the bench.

The test units were adhered to the plates using superglue (Scotch Superglue) on opposite parallel sides. Peeling away from the plates was then minimized while minimizing any additional structural support the glue might have provided by only

applying very little adhesive to the faces in question. Three such elements were made in order to conduct tests on all six shear planes. The results for the particular test unit are discussed in Chapter 4.

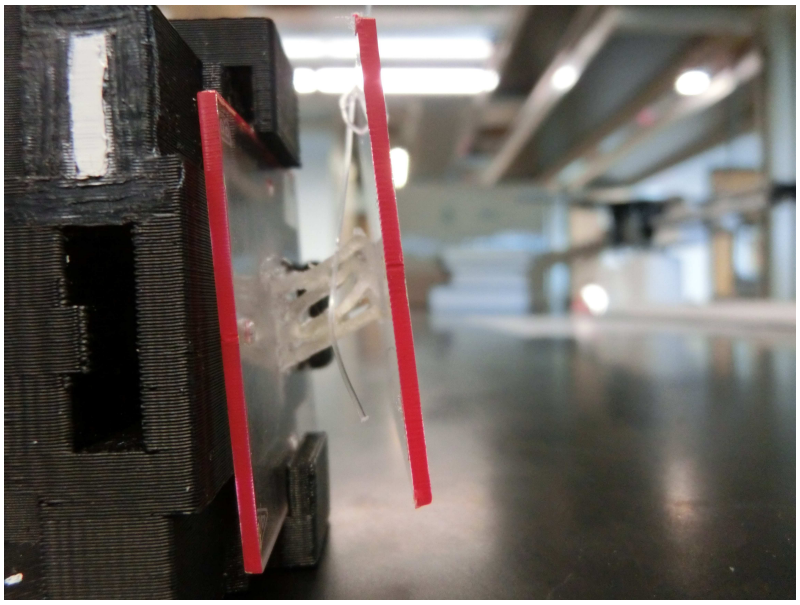


FIGURE 3.6 Shear test in action

3.1.3 Bending Test

Like the shear test, the bending test apparatus requires the test units to be adhered to opposite parallel sides using superglue. Adhesion loss and additional structural support was minimized, leaving the test unit free movement in the alloted directions of shear, bending, and compression.

The bending test apparatus is designed to center the test piece in place by one plate while the other is attached with fishing line to a weight. The bottom plate fits into a well 41mm $\times$ 41mm square and tied down using twine and scotch tape via the through-holes in the base, as seen in Fig. 3.7. The fishing line is attached to the hole in the plate closest to the front edge and run over a pulley before being connected to the weight. The angle was chosen to create a rotation axis as close to the opposite

edge as possible for all three axis orientations, such that angle measurements could be taken regardless of the orientation of the test piece.

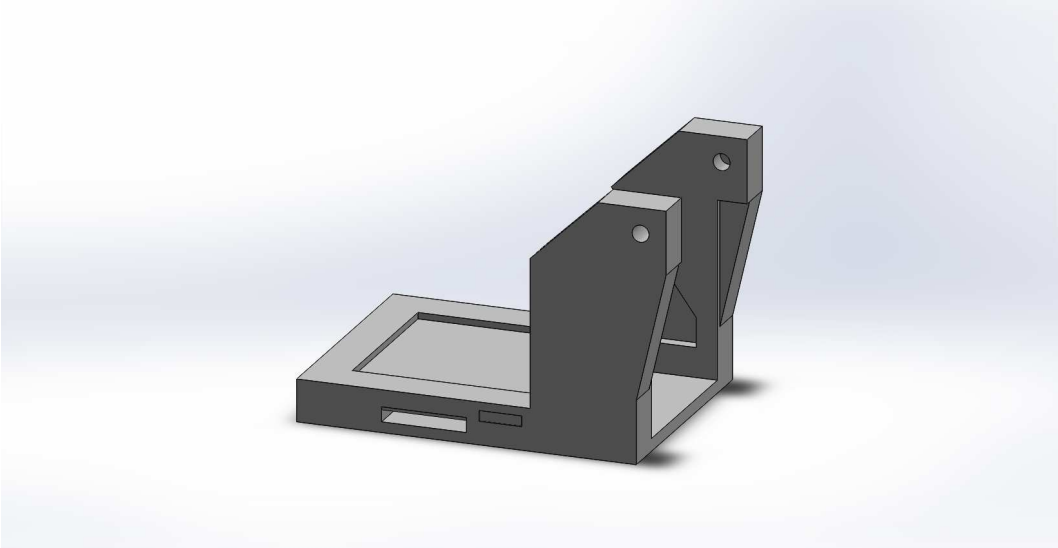


FIGURE 3.7 CAD model of bending test structure

The bending test apparatus is clamped down to the benchtop and the camera positioned so the reference swatch and the top plate are fully within the frame. The camera's position is noted with spike tape. The front of the structure is also taped down so that the effects of the loading are solely on the top plate (Fig. 3.8).

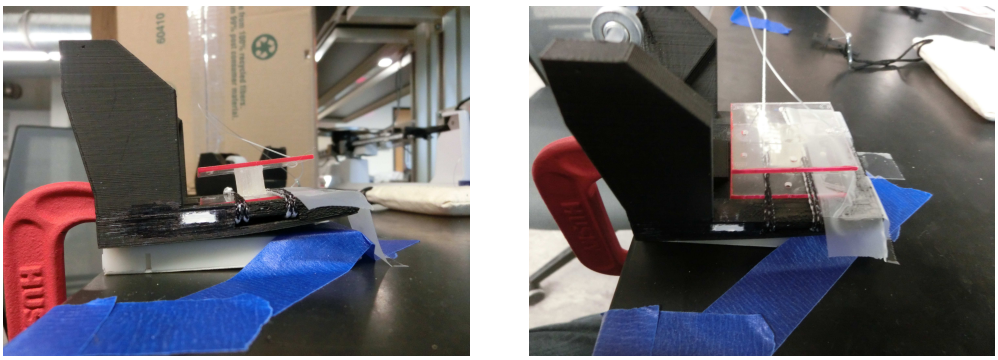


FIGURE 3.8 Bending test apparatus setup

3.2 Software Design

A set of software tools needed to be created to handle all the data from the images taken during the test runs. A single test could include up to 50 images as data points, which makes measuring distances by hand infeasible. To process all this data required robust image processing tools and automation. The task therefore became designing software that fulfilled the fundamental design goals of the entire toolkit. For this, we turned to open-source solutions to create the software solutions.

3.2.1 Design Process

The open-source software movement is designed to be decentralized, participatory, and transparent, allowing for collaborative modifications to software to be shared among users. This ethos has also started moving into research space, which by its very nature is open to collaborative efforts. [18] [17] Furthermore, by definition, open-source software and hardware is non-commercial, which drastically reduces costs.

In order for us to follow this design goal, software had to be written in a non-proprietary language, unlike Matlab. Common languages for scientific and engineering coding are C/C++, FORTRAN, and Python. Of the three, Python and C++ are more supported and have easy-integration with image processing libraries. Python was chosen as it is a high-level language with easy-to-read syntax and interacts well with data structures. With the addition of the free packages of NumPy and SciPy, the language gains functionality in complex data structures to exceed even Matlab even for applications such as neuroimaging research [35]. Using these packages keeps software costs non-existent and allows for further development by future researchers. The extendability by future researchers is in itself a key design factor, as most researchers end up customizing the software they need. Proprietary

software restricts the ability of users to not only understand and adapt program features but can also limit collaborative efforts if everyone must have access to the same version of very expensive software suites.

In terms of functionality, the software is designed to take a directory of images taken during the test, process the JPEG files in order to determine the location of the red plates, calculate the pixel-to-millimeter ratio, and calculate the force-displacement curves for each test.

3.2.2 Image Processing

The image processing is done using the Intel Open Computer Vision Performance Library (OpenCV) and its Pythonic keybindings. The OpenCV library already includes many of the algorithms already optimized for a variety of CPU architectures. We chose to write our own software instead of using existing tools such as ImageJ to integrate all the scripts into a single program and to keep all development in a single language.

The fundamental crux of the image processing done lies in the detection of colors and edges within an image. This requires an understanding both of what a digital image is and the filtering techniques used. A digital image is a numerical array that contains information on RGB and intensity values, known as pixels. [36] Since an image is therefore an array structure, it is possible to treat it as a numerical array for computational purposes.

Like any other digital signal, information can be extracted from it by manipulation via passing the signal through various filters. It is probably the most basic function within image processing. In the general sense, a filter applies a function to a pixel based on the input of the pixels surrounding it [37]. The effects of the filter, therefore, depend on the function.

3.2.2.1 Contour Detection via Bilateral Filter

One filter method is the bilateral filter, and it is used in contour detection when processing images from the compression tests. If we treat a digital image as an $x \times y$ array, then what we need to extract from the image is the difference in the *y* values for the centroids of each plate. To do this, it is necessary to find the two largest continuous contours mapped to the red plate edges.

The bilateral filter is designed to apply smoothing to an image while maintaining some amount of edges. One of the primary applications of this filter is in denoising, but it has also been used for tone mapping, isolating small variations in texture and details in images [38].

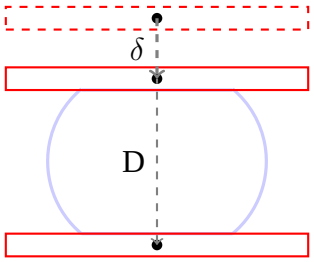


FIGURE 3.9 Compression diagram

The bilateral filter is thus an appropriate tool to determine color similarities within an image. ”Bilateral filters...can operate on the three bands at once, and can be told explicitly, so to speak, which colors are similar and which are not. Only perceptually similar colors are then averaged together, and the artifacts...disappear.” [37] For the application here, where detection of similar colors and their intensities is important for detecting a continuous contour.

To do this, the filter takes a kernel and performs a Gaussian convolution on the image:

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} (\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(|I_p - I_q|) I_q \tag{3.1}$$

$$W_p = \sum_{q \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(|I_p - I_q|) \tag{3.2}$$

where W_p is the normalization factor ensuring pixel weights sum to 1.0, I is the image in question, and σ_s and σ_r specify the amount of filtering. Equation 3.1 is a normalized weighted average with G_{σ_s} being a Gaussian that decreases the influence of pixels as a function of distance and G_{σ_r} being a Gaussian range that decreases the influence of pixels $\mathbf{q}$ that differ from I_p in intensity value. [39]

The functional use of these parameters σ_s and σ_r mean that a bilateral filter reduces the noise in the input image (smoothing) while preserving the edges.

OpenCV provides a bilateral filter function in the form of

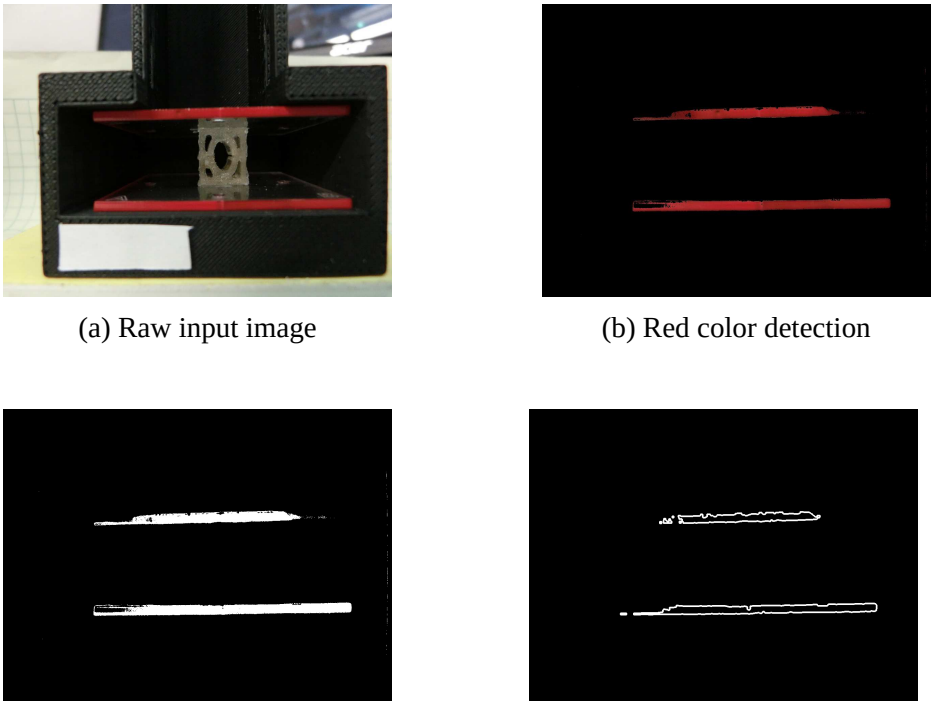
```
dst = cv2.bilateralFilter(src, d, sigmaColor, sigmaSpace)
```

with `d` being the diameter of the pixel neighborhood the filter uses. The documentation recommends the sigma values to be within the range of $(10 < \sigma < 150)$ and `d` to be ≤ 9 for an offline application with heavy noise. [40] However, since it is critical to locate the greatest concentration of pixels that fit the filter criteria, the code takes on a slight hit to computational optimization with a `d` value of 11 in order to eliminate as much noise as possible.

The image is subjected to a minor erosion and morphology transform to clean up any stray artifacts from the bilateral filtering. From the cleaned binary image, we find the outer contours of the areas remaining. Any contour that exceeds a threshold area is identified as one of the two plates; any contour that does not match the threshold criteria is removed. Fig. 3.10 shows the step-by-step process each image undergoes.

Once the contours have been identified in the binary image, we use the built-in OpenCV function `moments` to find the image moments[1] and thus image center of

[1] OpenCV uses Green's theorem to compute the moments, or weighted average of pixel intensities, for a vector or rasterized shape up to the 3rd order.



(a) Raw input image

(b) Red color detection

(c) Binary image mask

(d) Contoured image

FIGURE 3.10 The color detection image processing

mass for each contour.

From the two mass points, we can then determine the distance between them in units of pixels. The next step is converting that pixel distance to millimeters. One method is taking an outside imaging program and measuring a reference distance by hand, such as by use of MATLAB's distance tool. Doing this method might be acceptable for a single image, but quickly encounters two problems for multiple images: either the time required per image has increased to the point where this automation is pointless or it becomes a source of possible error if only a single image from a test run is measured.

To counter both of these problems, other image processing techniques were to determine the conversion ratio of pixels to millimeters. Instead of the bilateral filter, this submodule uses a Canny algorithm in specific region of interest (ROI) of

the image where the reference swatch is known to be. Reducing the image to as much of a white-on-black area as possible allows for quite generous RGB and area thresholding, mitigating much of the effects of lighting and shadows so as to get as accurate a reference as possible. The detection algorithms for the shear and bending tests also use Canny edge detection.

3.2.2.2 Edge Detection with Canny Algorithm

The Canny edge detector is designed to optimize the detection of true edges while minimizing false-positives [41] [42]. It is also a standard for edge detection algorithms since its inception in Canny's 1986 paper. OpenCV provides a CPU assembly optimized version.

The Canny edge detection first requires use of a Gaussian blur filter, with a kernel size of [3,3]. The kernel is an $n \times n$ matrix representing the number of pixels around to get a single pixel value which is then convoluted with the image. A [1,1] kernel would just be a single pixel. The kernel size must therefore be odd and positive. The size of the kernel determines the spread of the Gaussian, which specifies the amount of blurring; larger values include more pixels, which means more blur. Once again from this a binary mask is created, and a Gaussian blur with kernel size [3,3] is applied to eliminate noise before the Canny edge detector is used.

The reasoning for a Canny filter rather than bilateral filtering is based on the process by which OpenCV finds contours. In the case of the compression test, the image moments needed to be extracted, since distance calculations could be performed on simply those two points. The shear and bending algorithms are dependent on extracting the approximation of lines from each image. Therefore, it is critical that the absolute edges of the plates are found.

In the case of the shear test, the algorithm constrains the detected contour with a bounding box that captures all points. The bending test, however, constrains the

contour in the *minimum* bounding box, which preserves any rotation. The reason for the difference is due to the desired information to be extracted from the image. In the case of the shear test, the algorithm is looking for the uppermost or bottommost edges of each plate. Once it finds that point, it assumes that as the top edge and constructs the bounding box. In the case of Fig. 3.11, the bottom edge was used due to lighting conditions and to prevent any risk of clipping from the top edge.

From there, it determines the midpoint of each top edge and draws a line between those two points (Fig. 3.12). Angle α is determined as the offset from the adjacent A and, by trigonometry, is used to find displacement δ, which thus ignores any photo offset from the horizontal.

(a) Raw image with bounding box (b) Canny filter mask

FIGURE 3.11 Shear test detection

In contrast, the bending test is more concerned with the angular displacement during loading (Fig. 3.13). Angle θ is determined by finding the midpoints on the minor axis of the top plate and taking that line as a vector, seen in Fig. 3.14. From this vector, θ is derived in Eq. 3.3 as referenced to the horizontal. Force-displacement can be determined as the $\Delta\theta$ relative to each loading condition, starting from 0N.

$$\hat{\mathbf{u}} = \frac{x\hat{\mathbf{i}} + y\hat{\mathbf{j}}}{\sqrt{x^2 + y^2}}$$

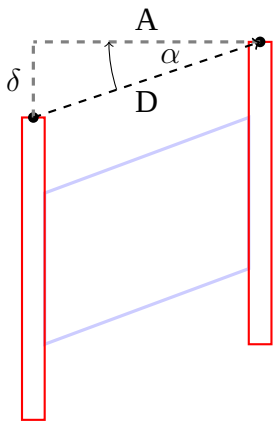


FIGURE 3.12 Shear diagram

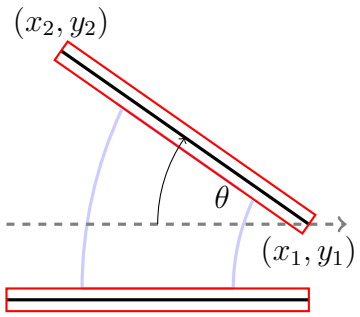


FIGURE 3.13 Bending diagram

$$\hat{\mathbf{u}} \cdot \hat{\mathbf{i}} = \frac{x}{\sqrt{x^2 + y^2}} = \cos\theta$$

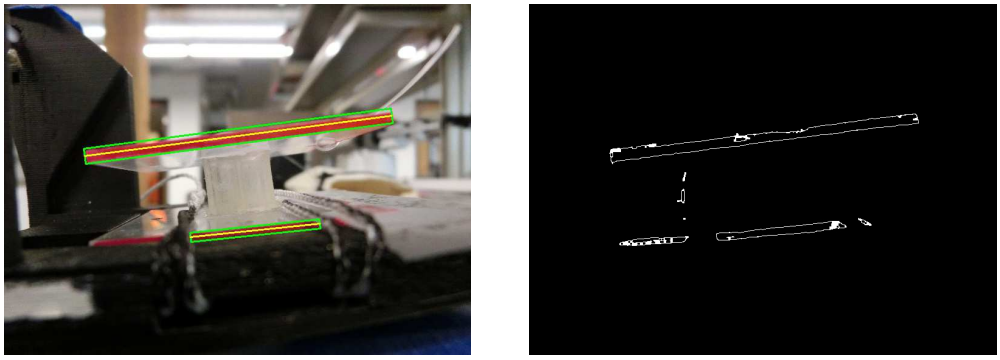
$$\theta = \arccos\left(\frac{x}{\sqrt{x^2 + y^2}}\right) \tag{3.3}$$

Taking consideration of the lower plate, in order to eliminate any rotation around the horizontal axis of the image:

$$\theta = \arccos\left(\frac{\hat{\mathbf{u}} \cdot \hat{\mathbf{v}}}{\|\hat{\mathbf{u}}\|\|\hat{\mathbf{v}}\|}\right) \tag{3.4}$$

with $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$ being the unitized vectors relating to the top and bottom plates.

All pixel-to-millimeter ratios were computed from the reference swatch in the



(a) Raw image with bounding box (b) Canny filter mask

FIGURE 3.14 Shear test detection

first image in the test directory file structure. We use a Canny filter to determine the length of the major axis of the swatch and relate that to the axis' real measurement of 10mm.

3.2.3 Data Fidelity

Discussion of the qualitative results of the compression, shear, and bending tests are described in detail in Chapter 4. Here, we only discuss the fidelity of the image processing.

Considering one set of compression testing, out of 13 tests with 46 images per test, the algorithm failed to identify the contours in 8 of the images, giving the algorithm a failure rate of 1.34%. In 8 of the 13 tests, the algorithm failed on 1 image out of the 46, for a failure rate of 2.17%. None of the tests had more than one failed image detection.

It is necessary to determine and evaluate the possible sources of error caused by the camera itself and image processing techniques.

3.2.3.1 Camera Calibration

The easy accessibility to cameras makes the step of calibrating the system to the camera used essential. Every camera causes some distortion when it creates an

image. To account for this, one needs to take into consideration the intrinsic and extrinsic parameters of a camera. Each camera is unique but once these distortion coefficients and parameters, the correction can be applied to any image captured by it. The following equations assume a pinhole camera model.

$$DC = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{3.5}$$

DC are the distortion coefficients, with *k* values referring to radial distortion and *p* values referring to tangential distortion.

$$CM = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

CM is the camera matrix, (f_x, f_y) are the focal length, and (c_x, c_y) are the optical centers. The camera matrix is used in a perspective transformation to project 3D points into an image plane.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.7}$$

OpenCV provides built-in functions to do this calibration, as well as test patterns. [43]

This calibration is included, but we found that the intrinsic properties of the camera were not the major sources of error in the image processing, seen in the simple image test in Fig. 3.15.

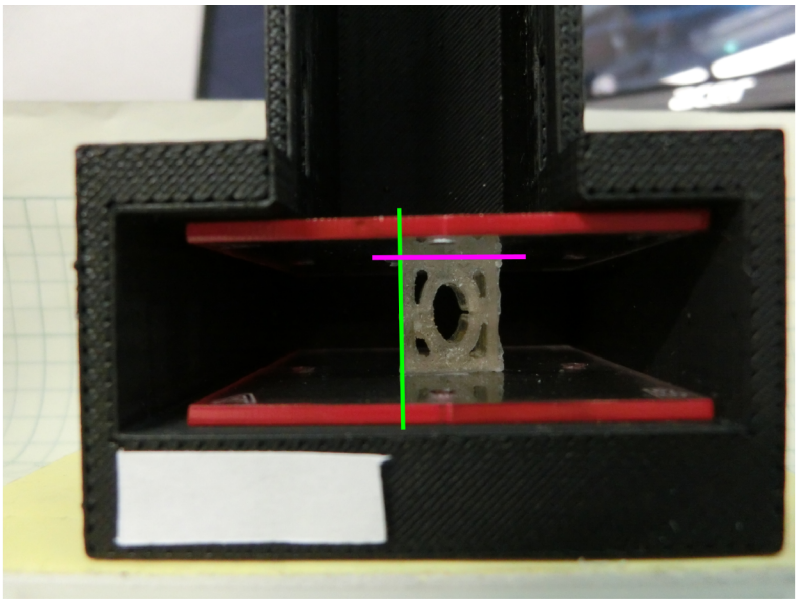


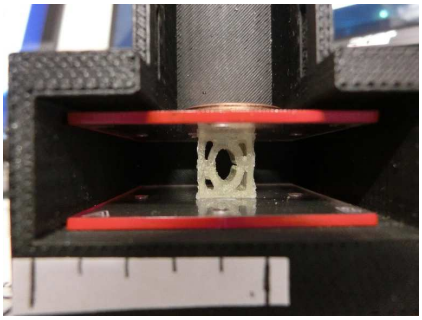
FIGURE 3.15 Testing the radial and tangential distortion of the Casio Exlim HS Ex-ZR400

3.2.3.2 Errors from Image Processing

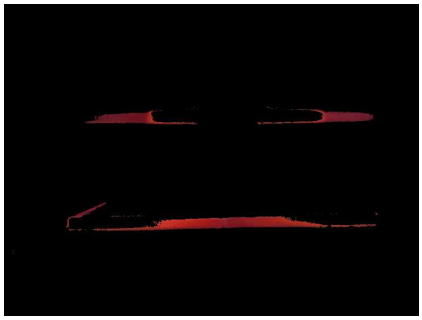
The source of the errors was likely a flaw in the color detection algorithm. The process takes a range of RGB values aimed at a broad red spectrum, but catching everything could lead to more false-positive identifications. To account for this, the contour selection relies on a threshold value to determine the largest contours, which are then assumed to be the top and bottom plates.

The source of error is when the largest contours are of the same plate, only broken into two parts due to failed color detection, as seen in Fig. 3.16. In this particular instance, the brighter areas, although they look red to a human eye, have a higher instance of green than the rest of boundary conditions. However, if we increase the allowed green value from 56 to 90, this has a significant impact on contoured image, as seen in Fig. 3.17. Increasing the RGB value in this way would not decrease the possibility of errors in this method of color detection.

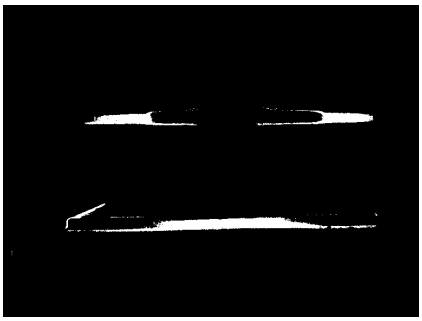
As it is, the errors produced by this method are fairly easy to catch. If the algorithm detects the two largest contours as being on the same plate, the distance bet-



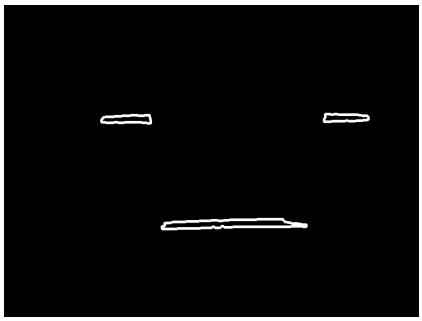
(a) Raw input image



(b) Red color detection



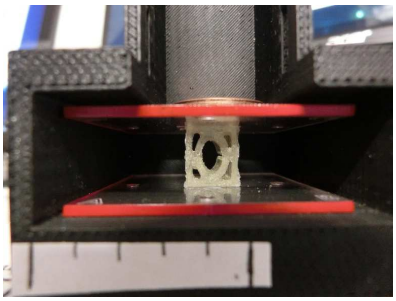
(c) Binary image mask



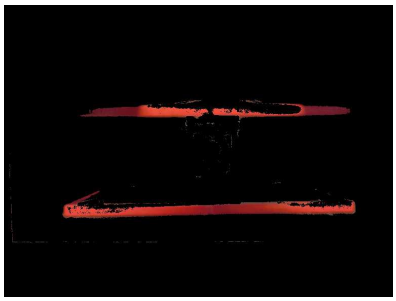
(d) Contoured image

FIGURE 3.16 Errors in color detection

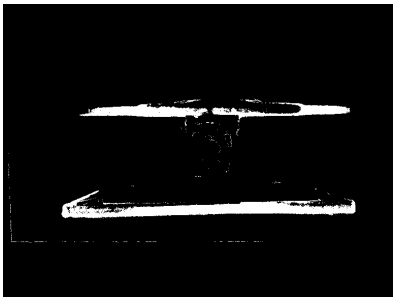
ween the center moments will be less than 1mm. Since the dimensions of the unit cell themselves state this is impossible, we can safely disregard those data points.



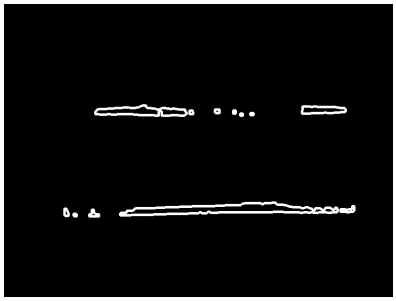
(a) Raw input image



(b) Red color detection



(c) Binary image mask



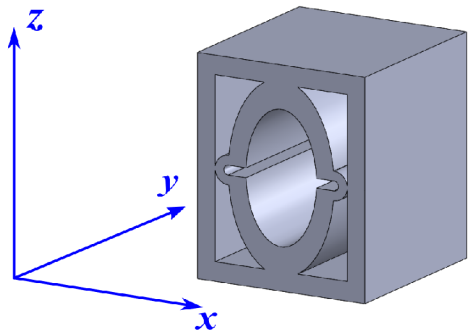
(d) Contoured image

FIGURE 3.17 Errors in color detection from altered RGB values

CHAPTER 4

Experimental Results

The SVE was tested using the hardware and software testkit to measure a subset of its force-displacement properties. The axes of the SVE are defined in Fig. 4.1 and will be referred to in this chapter.



;

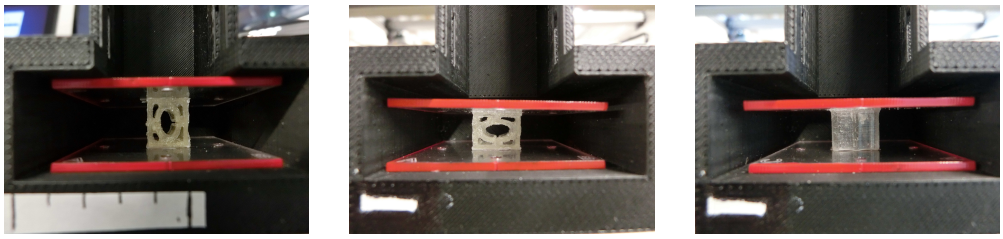
FIGURE 4.1 SVE axis reference

The subset of force-displacement properties were derived from compression, shear, and bending behavior.

4.1 Compression

Compression tests were conducted on three loading axes, shown in Fig. 4.2. A test is defined as the loading and orientation combination for an SVE. Each test

consists of multiple trials, defined as loading the structure from 0N to a maximum value incrementally. Every data point in a trial consists of a force value and an associated displacement value for an integer number of pennies N, where N varies from 0 to 46. All the trials use the same SVE.



(a) Z-axis compression (b) X-axis compression (c) Y-axis compression

FIGURE 4.2 Compression axis orientation

Initial compression tests for z-axis loading (4.2a) used the MATLAB distance tool to determine a pixel-to-millimeter ratio. For the x- and y-axis, the px-to-mm ratio was determined via the software described in Chap. 3. For the z-axis test, 12 trials were run, and the results are given in Fig. 4.3. The data is plotted with force in Newtons on the horizontal axis and distance in mm (referring to the height of the SVE) on the vertical.

Fig. 4.4a shows the mean of the compression trials before and after smoothing. The mean values for the test were computed and that data was smoothed using a Savitzky-Golay filter with a window size of 13 and a polynomial order of 5 for linear regression. The Savitzky-Golay filter function in SciPy is based on the data smoothing least-squares calculations that preserve the original shape of the data as much as possible [44].

Error bars were calculated as a standard deviation from the mean value. Both the mean and error bars and the filtered data are plotted (Fig. 4.4), with the raw mean in blue and the filtered mean in orange. For z-axis loading, force is plotted on the horizontal, distance on the vertical. For x- and y-axis loading, displacement (change in height of SVE) is plotted on the vertical instead due to the value range

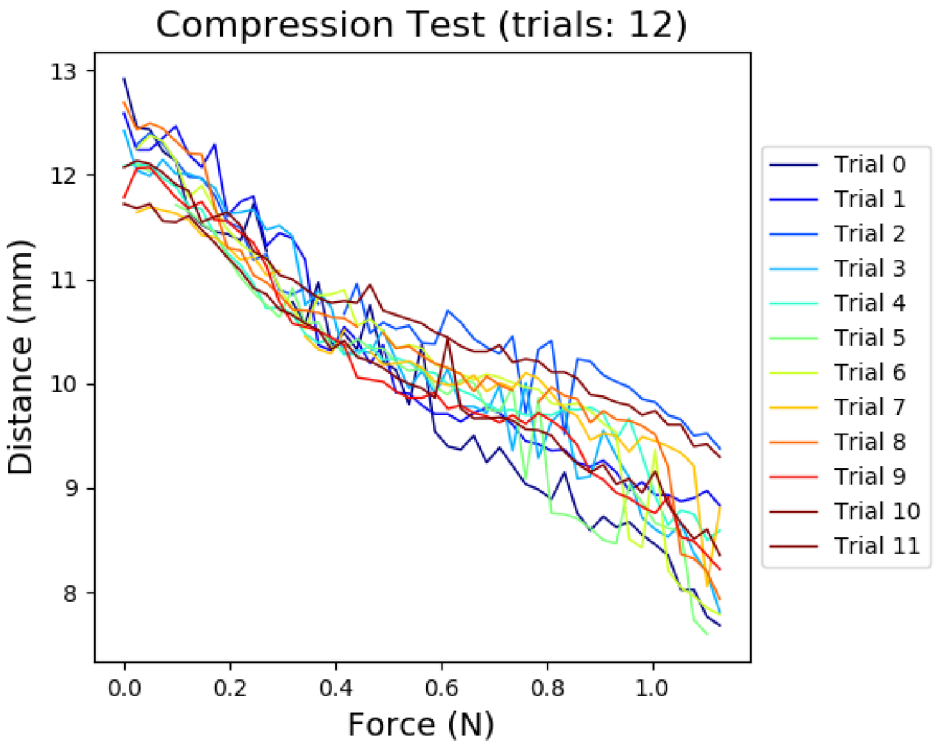


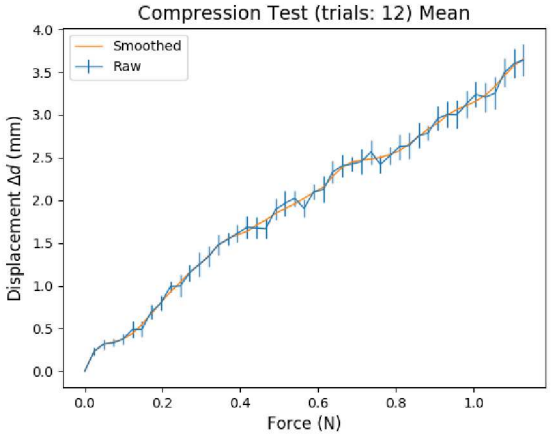
FIGURE 4.3 Force-displacement data, z-loading

being smaller.

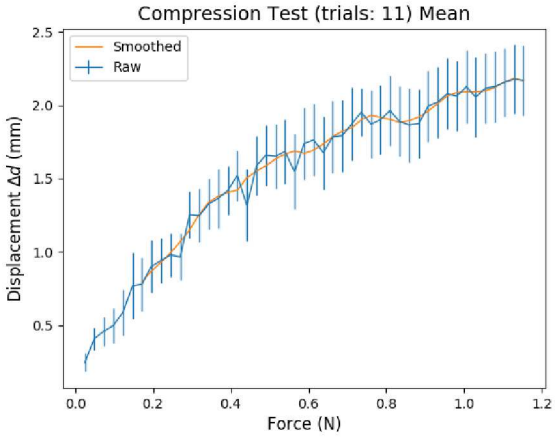
Using force-displacement curves, it is possible to extract mechanical parameters such as an effective spring variable *k*. Here, the slope of the curve is the effective *k* for the SVE in that particular orientation, and as can be seen in the figures, that slope changes with loading:

$$k = \frac{dF}{dL} \tag{4.1}$$

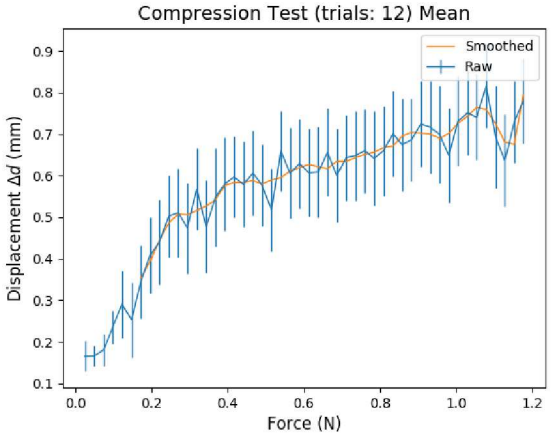
As we can see from Fig. 4.4a, for z-axis loading derived from 12 trials the gradient *k* is a non-linear function with two knees in the curve at approximately 0.4N and at 0.8N. The SVE shows greater deformation between loading conditions of 0 and 0.4N than between 0.4 and 0.8N. After 0.8N, the slope is similar to the 0 - 0.4N region.



(a) Z-loading



(b) X-loading



(c) Y-loading

FIGURE 4.4 Averaged force-displacement curves, compression

From the 11 trials in the x-axis loading, *k* follows a fairly clean, non-linear *k* function. The Savitzky-Golay 5th order polynomial cannot solve at the beginning of the curve. The slope does show greater deformation from 0 to 0.6N.

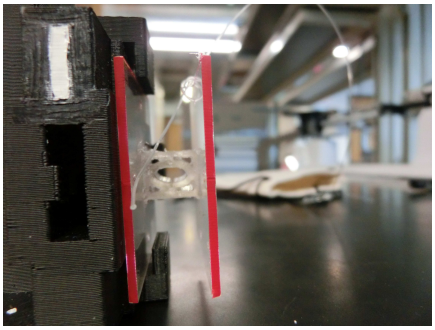
Like the x-axis loading, the y-axis also holds a non-nonlinear *k* function, determined from 12 trials. There is an obvious knee in the curve at approximately 0.25N, where the deformation becomes much more gradual. The local minimum to the left of 1.2N is due to an outlier in the data. The y-axis loading shows the least overall deformation given the loading conditions, with the deformations staying within the range of 0-1mm, which is a sharp contrast to the 0-4mm range of the z-axis loading.

4.2 Shear

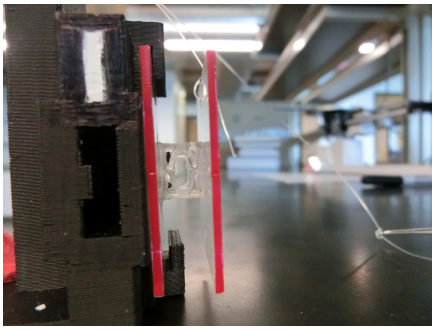
The shear tests follow the same methodology as the compression tests, with the exception being that *N* is defined from 0 to 48. The tests were conducted over six shear planes, shown in Fig. 4.5.

The shear tests measured the force-distance for each orientation, with distance defined as the vertical offset between the two plates δ shown in Fig. 4.6 (h is the initial height of SVE and D is change). All of the shear tests underwent 11 trials, with the exception of the *zy*-plane, which underwent 10. (The discrepancy is due to the file size of the images.) $N = 49$, with an initial loading of 0N before iterating on $F = ((2.5N + 9.8) \times 10^{-3}kg) \times 9.81\frac{m}{s^2}$. The results of the tests are found in Fig. 4.7, with force in N plotted on the horizontal and distance in mm plotted on the vertical.

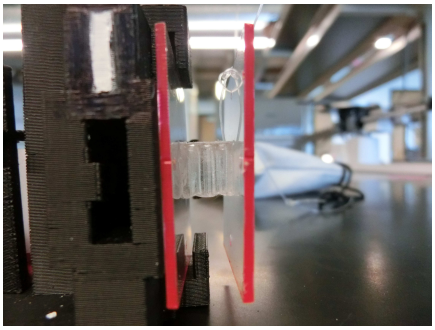
The *xz*-plane force-displacement results are plotted in Fig. 4.7a. The plot only goes up to 1.0N; there was significant noise in the data after this point and was omitted. From the rest of the data, the nonlinear *k* function has a knee around 0.8N loading force, with *k* being much steeper after this point. The slope in the region



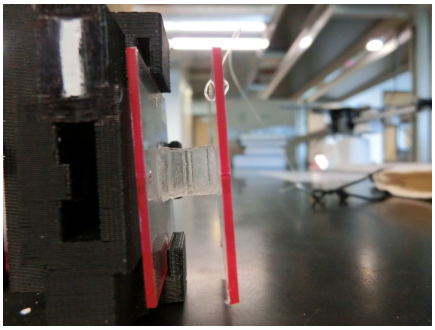
(a) *xz*-plane shear



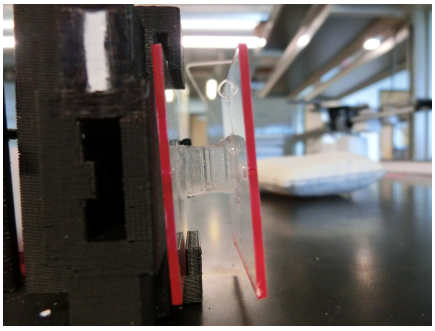
(b) *zx*-plane shear



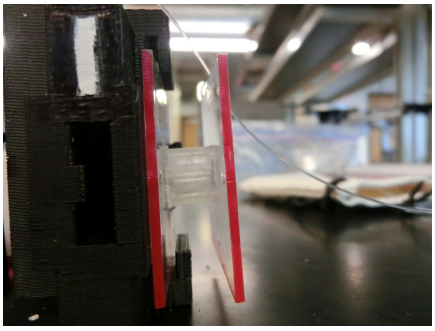
(c) *xy*-plane shear



(d) *yx*-plane shear



(e) *yz*-plane shear



(f) *zy*-plane shear

FIGURE 4.5 Shear orientations

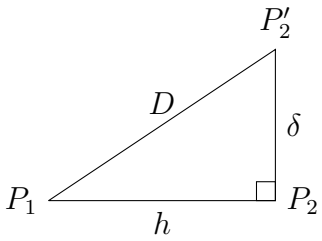
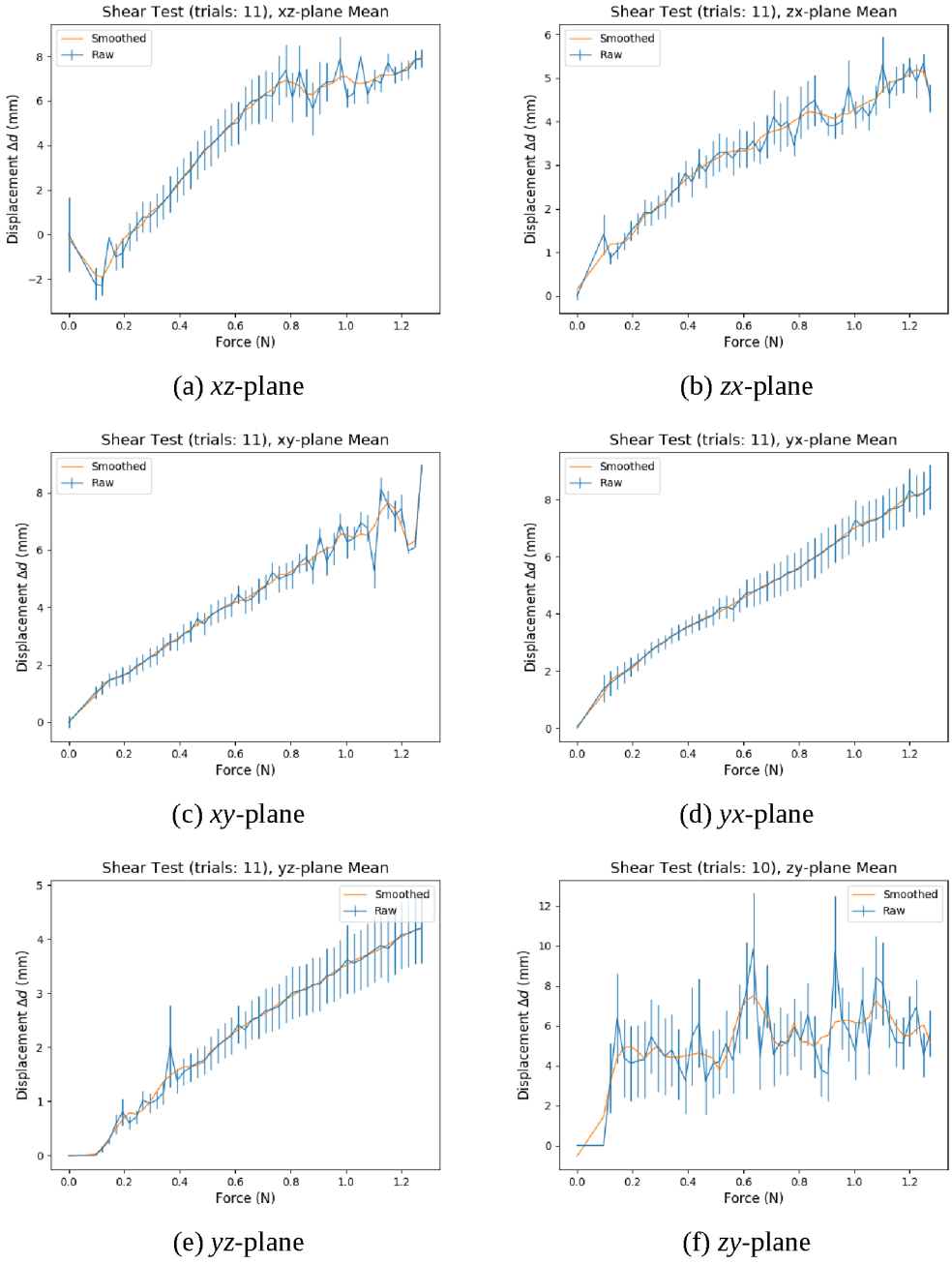


FIGURE 4.6 Shear geometry

between 0.4N and 0.8N is nearly flat. The force-displacement results for the *zx*-plane (Fig. 4.7b) show that the *k* function is nonlinear with



(a) *xz*-plane (b) *zx*-plane

(c) *xy*-plane (d) *yx*-plane

(e) *yz*-plane (f) *zy*-plane

FIGURE 4.7 Force-displacement for shear

The force-displacement results for the *xy*-plane and *yx*-plane are plotted in Fig. 4.7c and 4.7d respectively. The *xy*-plane shows a steady k between a change in slope $\frac{dF}{dx}$ at $F \approx 0.2N$, after which it undergoes a much slower displacement. The *yx*-plane shows a very similar behavior.

The *yz*-plane force-displacement results are plotted in Fig. 4.7e. The initial

loading of the bag (from 0N to 0.096138N) caused a spuriously large displacement in some trials due to errors in signal processing. Subsequent loading had a much more gradual force-displacement slope. Therefore, the outliers were removed, showing a non-linear slope with a knee at 0.4N.

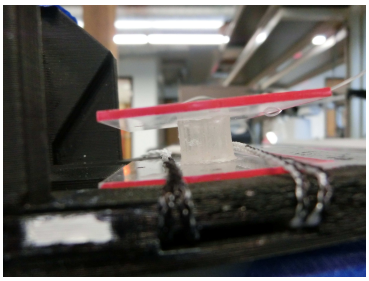
The *zy*-plane force-displacement results are plotted in Fig. 4.7f. The later trials in this test show significant noise, perhaps suggesting fatigue within the SVE or increased errors in the image processing. Because of this, it is somewhat hard to determine the locations of the nonlinearities in the k function.

Comparing the results to an inspection of the photo record, there was some difficulty in accurately detecting the correct two edges. The solution required resizing the images to 1/6 of height and width to remove false-positives resulting from incomplete contours. This is likely due to lighting conditions decreasing the contrast enough in the larger image, such that false positive identifications would not reach the threshold value in the resized ones. Since the original images are 4608×3456 pixels at 72 dpi, the resize leaves turns the image to 768×579 pixels at 96 dpi, reducing the pixel-to-millimeter ratio by a factor of 6. The alteration is accounted for in determining the relation to the reference swatch. The data from the tests suggests that this is an acceptable compromise between pixel granularity and accurate edge detection.

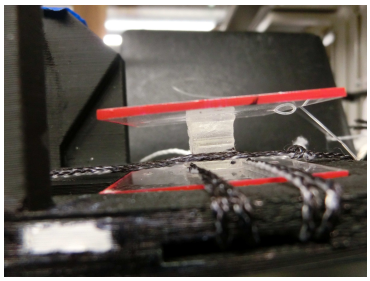
4.3 Bending

The results of the bending tests on 5 axis are detailed in this section. Five were conducted because of the differences on planar loading in addition to the bending axis needed to be taken into consideration. For example, bending along the x-axis requires testing with loading on the y-plane and on the z-plane, or *xy*-bending and *xz*-bending.

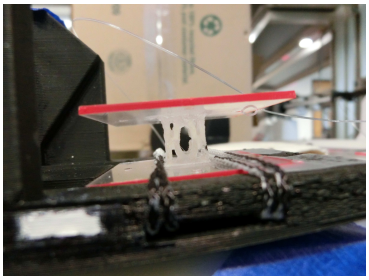
Due to time constraints, only testing of *yz*-bending was completed; the *yx*-bending test is left as future work.



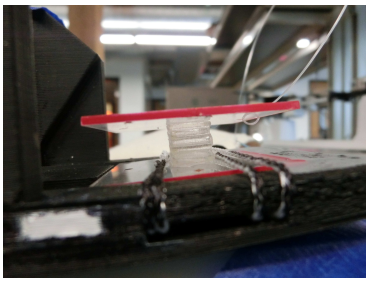
(a) xy-bending



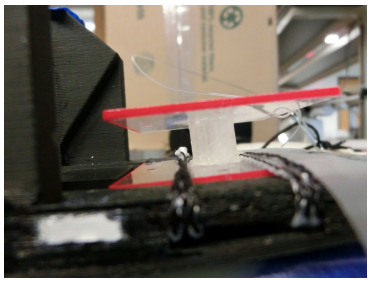
(b) xz-bending



(c) yz-bending



(d) zx-bending



(e) zy-bending

FIGURE 4.8 Bending orientations

The bending tests measure the angular displacement-force for each bending axis. The angle θ is defined as the change in angle between the top plate and bottom plate. Unlike the compression and shear tests, the number of trials varies between each test. This is because the value of N varies for each test, as loading halted once the edges of the top and bottom plates contacted. This point required a different integer number of pennies per orientation. N remained constant per trial within each test, as did

the SVE used. Once again accounting for the weight of the bag, F was calculated starting at 0N and then as $F = ((2.5N + 9.8) \times 10^{-3}kg) \times 9.81\frac{m}{s}$. The N for each tensor is given in Table 4.1.

Dyadic	Number of Pennies N	Max Load (N)
xy	48	1.27204
xz	24	0.68404
yz	19	0.56154
zx	12	0.39004
zy	42	1.12504

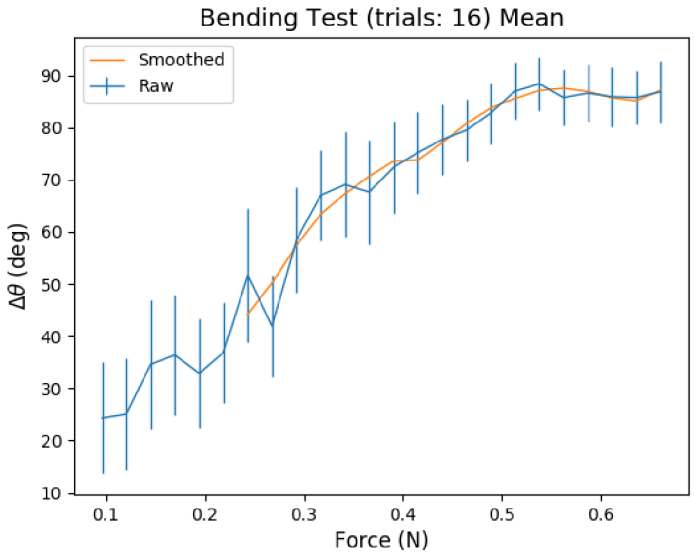
TABLE 4.1 Maximum loads for dyadics

The data results are plotted in Fig. 4.9, with force in N on the horizontal and the change in angle ($\Delta\theta$) on the vertical axis. In all tests, the mean and error bars are plotted in blue along with the Savitzky-Golay 5th order polynomial linear regression in orange. The results show that the Savitzky-Golay filter had significant difficulty in fitting the data from these tests at lower loadings. Only data for the *xz*-bending and *zy*-bending are shown, due to the other tests showing spurious results.

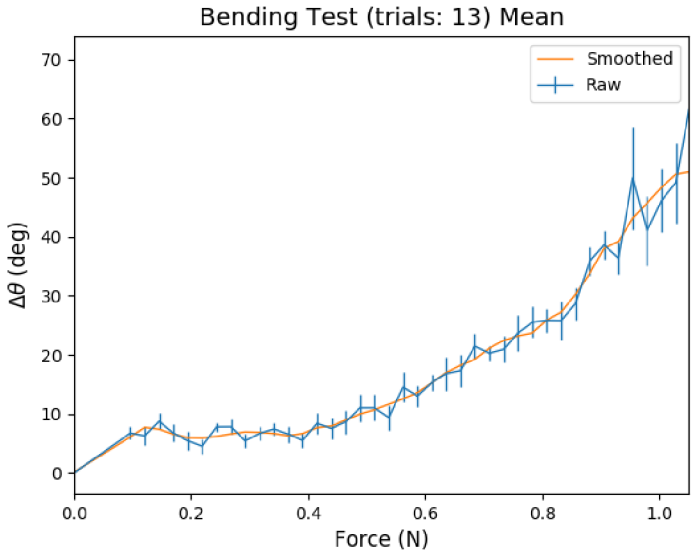
The force-angular displacement results for the *xz*-bending are shown in Fig. 4.9a. $N = 24$ for this test, suggesting this tensor is more susceptible to bending than the *zy*-tensor and the *xy*-tensor. The data follows a nonlinear function, flattening at approximately 0.5N when the two edges of the plates approach.

The force-angular displacement results for the *zy*-bending are plotted in Fig. 4.9a. $N = 42$ for this test, and the data shows that this axis is fairly resistant to loading at lower loads. The behavior obviously changes around 0.8N loading, with the SVE becoming remarkably more pliant after that point until the edges of the plates meet. The *zy*-tensor plot showed a sharp drop at the end due to processing error and the outlier was omitted.

As mentioned earlier, the tests for the *xz*, *yz*, and *zx*-tensors resulted in spurious



(a) xz-tensor



(b) zy-tensor

FIGURE 4.9 Anglular displacement-force for bending

data after going through the image processing. The image record and the loading N data show that there is a difference in the force-deformation behavior for each tensor, with the *yz*-tensor being most susceptible to bending and the *xy*-tensor the least. These curves could be found by hand from the image record; the computer image processing failed to detect the correct plate edges, causing extremely spurious

results. These errors are likely due to the imperfections that occurred during 3D printing the fixture, causing the test pieces to rest at an angle in the experiments. The code attempts to account for this possible source of error by taking the dot product between the centerlines of the plates. However, there was difficulty in calibrating the RGB color range and Canny filter to detect the small piece of the bottom plate unobscured by the apparatus and twine. Future versions would lift the base of the well, providing more red area to be detected.

Additionally, examination of the raw data suggests that the algorithm is also having difficulty at times determining the correct quadrant for the angle, exhibiting off-by-90°errors, even when the image identification would otherwise be correct. Future versions would have to correct this error.

The results for bending along the xz- and zy-tensors appear promising, suggesting that the test protocol and methodology has promise, but requires significant work to increase accuracy.

4.4 Conclusions

By using the toolkit, we have characterized the force-displacement relations on one SVE in compression and shear, and have a work-in-progress method for bending. The results are not enough to fully characterize this SVE, but are enough to show the possible validity of the structure as a variable-stiffness modeling element. The results from the compression and shear tests are enough to extract the 9 independent material properties for a linear elastic orthopedic block, as long as the loading conditions of the test are the same as the loading conditions in use.

Furthermore, the results show that the methodology for designing test structures and the software to identify the behavioral change is sound enough to be applied to designing tests for other properties such as torsion.

Through this testing, we have demonstrated the utility of the toolkit and thus now have a process that could be used to characterize SVEs. These SVEs might include intentionally designed structures like the one tested here and also self-organized structures such as foams.

CHAPTER 5

Conclusion

The intention of this project was to fill a need within the soft robotics community and create a tool other researchers can use to characterize the soft structures they use in their own research. Review of the literature and examination of common materials used such as PDMS shows a lack in the robust computational design tools necessary to create accurate models, which results in a slower pace for progress. The need for a very-accessible tool to do quantitative modeling on small samples was obvious.

5.1 Summary

Testing using the SVE shows that the toolkit shows promise as a useful research tool. The results of the compression tests were processed at a rate of about 1 image/-sec processing time, as taken by logging timestamps. Furthermore, the Canny edge detection proved to be significantly faster than the bilateral filter, being able to process 550 image files within 123 - 132 sec. Taken into account along with the time it takes to run the experiment procedure, it is still faster than a complex computational model.

Compression tests show a reasonable level of granularity to displacements, down to 0.1mm, and show non-linear force-displacement behavior. These results

show that we did achieve the design goals of orientation-dependent stiffness, with the y-axis is much stiffer than the z-axis, as intended. The results also show that each orientation has a non-linear force-displacement slope, with distinct regions of stiffness correlated to loading conditions.

Shear tests results were acceptable, although calibration for accurate edge detection was a significant issue. Many of these could possibly be mitigated with better lighting conditions and instrument placement, so as to assist the software in correctly identifying the edges in question by increasing contrast.

The bending test results, as mentioned, have the problem of the image processing algorithms not being robust enough to handle the full range of tests. Significant work needs to be done in fine-tuning the algorithm, the test-fixture, or both. Section 4.3 discussed the particular issues and likely sources of the errors for this test in particular. When the software was successful, the data results looked good.

Despite these flaws, the toolkit has the potential to be used to create approximate quantitative models of materials used in soft structures.

The SVE was used to validate the effectiveness of the toolkit, but it also proved highly effective in some of its own design goals. In compression in particular, the SVE achieves the goal of variable stiffnesses depending on orientation, with the z-axis being the most ductile and the y-axis the stiffest.

Shear tests also revealed some variation in deformation based on orientation. For example, the *zx*-plane shear had a deformation range of 0-6mm while the *xz*-plane orientation had a range of 4-16mm.

The main result from the bending tests is that the loads required to provide significant deformation are very dependent on the orientation.

Overall, the concept of a 3-dimensional soft structural element to use as a modeling tool appears validated, although the SVE in particular requires further design work.

5.2 Contribution

This thesis detailed the major contributions of solving the problems of characterizing meso-scale soft structures agnostic of material and geometry by:

- Design a low-cost testing apparatus to measure the mechanical properties of a structural volume element undergoing large deformations
- Design a particular structural volume element and characterize its force-displacement properties using the testing apparatus, as a means of demonstrating how a broader library of SVEs might be created in the future.

Both of these contributions are important, due to the problems within the soft robotics field that they are aimed at solving. The materials used most often in soft robotic applications are elastomers that have non-linear deformations under different loading conditions. This characteristic makes them extremely difficult to model in simulation, requiring assumptions that decrease the accuracy of such models, large computational power, or both. Many research works take a structure-first design paradigm, relying on the iterative design process to characterize and refine their soft structures. Even with 3D printing as a prototype manufacturing process, this approach can be expensive and slow.

Towards point 1, the toolkit was designed to make use of lower-cost technologies such as plastic 3D printers as well as developing open-source software solutions to analyze the structures we called structural volume elements. Using tools such as OpenCV and other Python packages, we created software tools that could analyze the force-displacement behavior of an SVE to a sub-millimeter granularity in the case of compression. By making these software and hardware tools open-source or otherwise freely-available, it fills a niche for such tools that are low-cost and do not require great force loads as larger, commercially-available tools do.

We also defined the idea of the structural volume element, proposing a meso-scale deformable structure that can be assembled in lattices and used to characterize a full system. This approach flips the current structure-first design paradigm, where we instead define the desired properties first and build structures to match. By using the meso-scale SVE, we intend for a greater flexibility in design of soft robots by being able to collapse many parameters such as material and sub-structural geometry into a single lumped parameter that can then be fit into a model.

For point 2, we used our own designed SVE to test our toolkit in the realms of compression, shear, and bending. The results from this show that the compression and shear tests (fixtures and methodology) do fairly well in characterizing the force-displacement properties in various orientations. The bending tests, while less robust, still show that with alterations to methodology, force-angular displacement properties can be equally characterized. Overall, the principles of the toolkit are sound, and show that the designed SVE successfully matches our desired design parameters of variable stiffness based on orientation and also resistance to shear forces dependent on orientation. From this, we can extrapolate how this toolkit can be used to create other SVEs to gain other desirable behaviors and so build a collection of building blocks for soft structures.

5.3 Future Work

There is much work needed to be done to fully turn this into an accepted design tool. For example, while the tests were validated using the SVE, a full validation of the usefulness of the kit would require testing on other structural elements. These would include different materials, such as foams or other polymers beyond TangoPlus, and different geometries. Comparing the results from the kit with the properties of a known element of specific material and geometry is also necessary to fully

validate the system.

Furthermore, there should be testing apparatuses for tests in other loading conditions, such as tensile loading and torsion. The current generation of the toolkit provides no method for determining properties based on these loads, which are dynamically imperative for complex motion. Additionally, this lack means that it is impossible to validate the SVE itself under these loading conditions, which limits the possibilities for its effectiveness. Future models of the testing apparatuses could also possibly be integrated into a single unit, rather than requiring one per loading condition.

While the literature proves that it is possible to combine structural elements, testing still needs to be done to prove the assertion that combining simple volume elements into a matrix gives a predictive power of behavior. Furthermore, work also needs to be done to determine the method by which the SVEs are combined, taking into account areas that might interfere with each other during loading. Once this has been tested, theoretical models such as lattice spring models can be reasonably used as a very powerful design tools.

The software designed for the toolkit is far from optimized, and while the rate of approximately 1 image/sec processed using the bilateral filter is reasonable at smaller scales, it represents a significant time commitment when dealing with thousands of images. The Canny filter is a much faster algorithm, however, the data results had much larger standard deviations for repeatability. There is room for code optimization at the very least, as well as extending the codebase to more accurately detect the plates and extending functionality for further loading tests. In terms of usability, the software is command-line only, which is slightly unfriendly to more-casual users. An enterprising coder would solve common user-input errors by creating a GUI interface that not only denotes what kind of test is being analyzed, but also gives a more dynamic control over RGB filter values. The latter would also help in

reducing the probability for error introduced by lighting conditions and by different camera models.

Overall, the current system is not perfect by any means, but it provides a reasonable foundation as a potential design tool that can ultimately be used to synthesize new robots and soft structures.

5.4 Broader Impacts

One of the important things about any avenue of research is that the work done advances the field in some way. Thus, the hope is that the contributions of this thesis provide a forward step in the field of soft robotics.

While not an obvious advancement, the toolkit and SVE are tools designed to fill an identified need. By creating tools that facilitate further advances, the field as a whole advances. Additionally, by creating these tools to be low-cost, we lower the barrier to entry and can encourage further interdisciplinary collaboration as more people have access to manufacturing and design capabilities. Thus, this work achieves that fundamental goal.

Furthermore, by proposing a property-first design paradigm, where the desired behaviors are specified prior to design of a structure, the SVE is a step in creating more general and robust models for soft robotic systems. In the future there could be a library of these SVEs, from which different SVEs could be selected to synthesize a new design. This library of building blocks would give rise to models that would bring soft structure analysis closer to more traditional methods of simulation and modeling.

With such tools, one can imagine future soft robots built using small meso-scale structures, as if they are soft LEGOs. We can walk forward into a world where soft robots crawl, swim, or fly into areas impossible for humans to survive, from search

and rescue operations to deep-space planetary exploration where the robots need the morphological advantages soft structures provide. Such a future is only possible if we have the tools to take us there.

Bibliography

[1] I. D. Johnston, D. K. McCluskey, C. K. L. Tan, and M. C. Tracey, “Mechanical characterization of bulk Sylgard 184 for microfluidics and microengineering,” *Journal of Micromechanics and Microengineering*, vol. 24, no. 3, p. 035017, Mar. 2014. [Online]. Available: http://stacks.iop.org/0960-1317/24/i=3/a=035017?key=crossref.cf83b17be3210f20943ae0e9169fc9b8

[2] “6.777j/2.751j Material Property Database: Material: PDMS (polydimethylsiloxane).” [Online]. Available: http://www.mit.edu/~6.777/matprops/pdms.htm

[3] D. P. Holland, E. J. Park, P. Polygerinos, G. J. Bennett, and C. J. Walsh, “The Soft Robotics Toolkit: Shared Resources for Research and Design,” *Soft Robotics*, vol. 1, no. 3, pp. 224–230, Sep. 2014. [Online]. Available: http://online.liebertpub.com/doi/10.1089/soro.2014.0010

[4] H. Lipson, “Challenges and Opportunities for Design, Simulation, and Fabrication of Soft Robots,” *Soft Robotics*, vol. 1, no. 1, pp. 21–27, Mar. 2014. [Online]. Available: http://online.liebertpub.com/doi/abs/10.1089/soro.2013.0007

[5] F. Renda, M. Cianchetti, M. Giorelli, A. Arienti, and C. Laschi, “A 3d steady-state model of a tendon-driven continuum soft manipulator inspired by the octopus arm,” *Bioinspiration & Biomimetics*, vol. 7, no. 2, p. 025006, Jun. 2012. [Online]. Available: http://stacks.iop.org/1748-3190/7/i=2/a=025006?key=crossref.649b1a3b6f7094c828206726c4acd945

[6] D. Rus and M. T. Tolley, “Design, fabrication and control of soft robots,” *Nature*, vol. 521, no. 7553, pp. 467–475, May 2015. [Online]. Available: http://www.nature.com/doifinder/10.1038/nature14543

[7] R. Pfeifer, M. Lungarella, and F. Iida, “The challenges ahead for bio-inspired ’soft’ robotics,” *Communications of the ACM*, vol. 55, no. 11, p. 76, Nov. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2366316.2366335

[8] K.-J. Cho, J.-S. Koh, S. Kim, W.-S. Chu, Y. Hong, and S.-H. Ahn, “Review of manufacturing processes for soft biomimetic robots,” *International Journal of Precision Engineering and Manufacturing*, vol. 10, no. 3, pp. 171–181, Jul. 2009. [Online]. Available: http://link.springer.com/10.1007/s12541-009-0064-6

[9] S. Kim, C. Laschi, and B. Trimmer, “Soft robotics: a bioinspired evolution in robotics,” *Trends in biotechnology*, vol. 31, no. 5, pp. 287–294, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167779913000632

[10] H. Hauser, A. J. Ijspeert, R. M. Füchslin, R. Pfeifer, and W. Maass, “Towards a theoretical foundation for morphological computation with compliant bodies,” *Biological Cybernetics*, vol. 105, no. 5-6, pp. 355–370, Dec. 2011. [Online]. Available: http://link.springer.com/10.1007/s00422-012-0471-0

[11] J. Hiller and H. Lipson, “Dynamic Simulation of Soft Multimaterial 3d-Printed Objects,” *Soft Robotics*, vol. 1, no. 1, pp. 88–101, Mar. 2014. [Online]. Available: http://online.liebertpub.com/doi/abs/10.1089/soro.2013.0010

[12] F. Ilievski, A. D. Mazzeo, R. F. Shepherd, X. Chen, and G. M. Whitesides, “Soft Robotics for Chemists,” *Angewandte Chemie*, vol. 123, no. 8, pp. 1930–1935, Feb. 2011. [Online]. Available: http://onlinelibrary.wiley.com.ezproxy.library.tufts.edu/doi/10.1002/ange.201006464/abstract

[13] M. F. Ashby and D. R. H. Jones, *Engineering Materials 2: An introduction to microstructures, processing and design*, 3rd ed. Amsterdam: Elsevier, 2006, oCLC: 552234177.

[14] C. Majidi, “Soft Robotics: A Perspective—Current Trends and Prospects for the Future,” *Soft Robotics*, vol. 1, no. 1, pp. 5–11, Mar. 2014. [Online]. Available: http://online.liebertpub.com/doi/abs/10.1089/soro.2013.0001

[15] M. F. Ashby and D. R. H. Jones, *Engineering Materials 1: An introduction to properties, applications and design*, 3rd ed. Amsterdam: Elsevier, Butterworth-Heinemann, 2005, oCLC: 552234154.

[16] J. Rossiter and H. Hauser, “Soft Robotics - The Next Industrial Revolution? [Industrial Activities],” *IEEE Robotics Automation Magazine*, vol. 23, no. 3, pp. 17–20, Sep. 2016.

[17] T. Lang, “Advancing Global Health Research Through Digital Technology and Sharing Data,” *Science*, vol. 331, no. 6018, pp. 714–717, Feb. 2011. [Online]. Available: http://www.sciencemag.org/cgi/doi/10.1126/science.1199349

[18] J. M. Pearce, “Building Research Equipment with Free, Open-Source Hardware,” *Science*, vol. 337, no. 6100, pp. 1303–1304, Sep. 2012. [Online]. Available: http://www.sciencemag.org/cgi/doi/10.1126/science.1228183

[19] R. Pfeifer, M. Lungarella, and F. Iida, “Self-organization, embodiment, and biologically inspired robotics,” *science*, vol. 318, no. 5853, pp. 1088–1093, 2007. [Online]. Available: http://science.sciencemag.org/content/318/5853/1088.short

[20] M. Giorelli, F. Renda, M. Calisti, A. Arienti, G. Ferri, and C. Laschi, “A two dimensional inverse kinetics model of a cable driven manipulator inspired by the octopus arm.” IEEE, May 2012, pp. 3819–3824. [Online]. Available: http://ieeexplore.ieee.org/document/6225254/

[21] F. Saunders, B. A. Trimmer, and J. Rife, “Modeling locomotion of a soft-bodied arthropod using inverse dynamics,” *Bioinspiration & Biomimetics*, vol. 6, no. 1, p. 016001, Mar. 2011. [Online]. Available: http://stacks.iop.org/1748-3190/6/i=1/a=016001?key=crossref.7a355cf99e3a0a89130dd889002f4bae

[22] F. Saunders and J. Rife, “Embedding Desired Eigenstates into Active and Passive Dynamics of a Linear, Underactuated Feedback System,” *Journal of Mechanical Design*, vol. 136, no. 7, p. 071005, Apr. 2014. [Online]. Available: http://mechanicaldesign.asmedigitalcollection.asme.org/article.aspx?doi=10.1115/1.4025295

[23] J.-Y. Lee, W.-B. Kim, W.-Y. Choi, and K.-J. Cho, “Soft Robotic Blocks: Introducing SoBL, a Fast-Build Modularized Design Block,” *IEEE Robotics & Automation Magazine*, vol. 23, no. 3, pp. 30–41, Sep. 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7549069/

[24] B. Jenett, D. Cellucci, C. Gregg, and K. Cheung, “Meso-scale digital materials: modular, reconfigurable, lattice-based structures,” in *ASME 2016 11th International Manufacturing Science and Engineering Conference*.

American Society of Mechanical Engineers, 2016, pp. V002T01A018–V002T01A018. [Online]. Available: http://proceedings.asmedigitalcollection.asme.org/proceeding.aspx?articleid=2558703

[25] G. A. Buxton, C. M. Care, and D. J. Cleaver, “A lattice spring model of heterogeneous materials with plasticity,” *Modelling and simulation in materials science and engineering*, vol. 9, no. 6, p. 485, 2001. [Online]. Available: http://iopscience.iop.org/article/10.1088/0965-0393/9/6/302/meta

[26] B. Jenett, S. Calisch, D. Cellucci, N. Cramer, N. Gershenfeld, S. Swei, and K. C. Cheung, “Digital Morphing Wing: Active Wing Shaping Concept Using Composite Lattice-Based Cellular Structures,” *Soft Robotics*, vol. 4, no. 1, pp. 33–48, Mar. 2017. [Online]. Available: http://online.liebertpub.com/doi/10.1089/soro.2016.0032

[27] I. G. Masters and K. E. Evans, “Models for the elastic deformation of honeycombs,” *Composite structures*, vol. 35, no. 4, pp. 403–422, 1996. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0263822396000542

[28] D. A. DaDeppo and R. Schmidt, “Sidesway Buckling of Deep Circular Arches Under a Concentrated Load,” *Journal of Applied Mechanics*, vol. 36, no. 2, p. 325, 1969. [Online]. Available: http://AppliedMechanics.asmedigitalcollection.asme.org/article.aspx?articleid=1398937

[29] R. P. Feynman, R. B. Leighton, M. L. Sands, and R. P. Feynman, *Mainly electromagnetism and matter*, nachdr. ed., ser. The Feynman lectures on physics. Reading/Mass.: Addison-Wesley, 2007, no. Richard P. Feynman; Robert B. Leighton; Matthew Sands ; 2, oCLC: 254170662.

[30] J. Moon, K.-Y. Yoon, T.-H. Lee, and H.-E. Lee, “In-plane elastic buckling of pin-ended shallow parabolic arches,” *Engineering Structures*, vol. 29, no. 10, pp. 2611–2617, Oct. 2007. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S014102960700034X

[31] J. E. Tetazoo, J. Florey, R. Q. Putz, and J. A. Random, “Rapid Temporary Structural Engineering in Impractical Locations.”

[32] M. A. Bradford, B. Uy, and Y.-L. Pi, “In-Plane Elastic Stability of Arches under a Central Concentrated Load,” *Journal of Engineering Mechanics*, vol. 128, no. 7, pp. 710–719, Jul. 2002. [Online]. Available: http://ascelibrary.org/doi/10.1061/%28ASCE%290733-9399%282002%29128%3A7%28710%29

[33] Y.-L. Pi, M. Bradford, and B. Uy, “In-plane stability of arches,” *International Journal of Solids and Structures*, vol. 39, no. 1, pp. 105–125, Jan. 2002. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0020768301002098

[34] “The United States Mint About Us.” [Online]. Available: https://www.usmint.gov/about_the_mint/index583f.html?action=coin_specifications

[35] “nipy/why.rst at master · nipy/nipy · GitHub.” [Online]. Available: https://github.com/nipy/nipy/blob/master/doc/faq/why.rst

[36] A. R. Smith, “A Pixel Is Not A Little Square, A Pixel Is Not A Little Square, A Pixel Is Not A Little Square!” *Microsoft Computer Graphics, Technical Memo*, vol. 6, 1995. [Online]. Available: http://ftp.alvyray.com/Memos/CG/Microsoft/6_pixel.pdf

[37] C. Tomasi and R. Manduchi, “Bilateral filtering for gray and color images,” in *Computer Vision, 1998. Sixth International Conference on*. IEEE, 1998, pp.

839–846. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/710815/

[38] S. Paris, P. Kornprobst, J. Tumblin, and F. Durand, “A Gentle Introduction to Bilateral Filtering and its Applications.” [Online]. Available: https://people.csail.mit.edu/sparis/bf_course/

[39] ——, “Bilateral Filtering: Theory and Applications,” *Foundations and Trends® in Computer Graphics and Vision*, vol. 4, no. 1, pp. 1–75, 2008. [Online]. Available: http://www.nowpublishers.com/article/Details/CGV-020

[40] “OpenCV: OpenCV modules.” [Online]. Available: http://docs.opencv.org/trunk/index.html

[41] J. Canny, “A Computational Approach to Edge Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4767851

[42] Y. Luo and R. Duraiswami, “Canny edge detection on NVIDIA CUDA,” in *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, Jun. 2008, pp. 1–8.

[43] “OpenCV: Camera Calibration.” [Online]. Available: http://docs.opencv.org/3.1.0/dc/dbb/tutorial_py_calibration.html

[44] A. Savitzky and M. J. Golay, “Smoothing and differentiation of data by simplified least squares procedures.” *Analytical chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.

[45] R. C. Huang and L. Anand, “Non-linear mechanical behavior of the elastomer polydimethylsiloxane (PDMS) used in the manufacture of microfluidic

devices," 2005. [Online]. Available: https://dspace.mit.edu/handle/1721.1/7456

[46] S. R. Quake and A. Scherer, "From micro-to nanofabrication with soft materials," *Science*, vol. 290, no. 5496, pp. 1536–1540, 2000. [Online]. Available: http://science.sciencemag.org/content/290/5496/1536.short

[47] A. Nasto, M. Regli, P.-T. Brun, J. Alvarado, C. Clanet, and A. E. Hosoi, "Air entrainment in hairy surfaces," *Physical Review Fluids*, vol. 1, no. 3, Jul. 2016. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevFluids.1.033905

CHAPTER A

Python Code

The Python code listed here has several package dependencies:

- numpy
- scipy
- cv2
- matplotlib
- imutils

The author suggests using Anaconda for installation.

Code current as of 2017-04-27. The most up-to-date version of the code can be found at: `https://github.com/kaazrut/softsys-toolkit`

code/imgproc.py

```
# Image processing for Toolkit
# The software suite is released under the CCO License, with
# acceptable parts compatable with the GNU Public License.  If you
# use the software as is or as a basis for expansion, the original
# author (Ash Turza) would appreciate acknowledgement but it is not
# required.
#
# Ash Turza can be found at kaazrut@gmail.com.
#
# This is the toplevel script for the software toolkit.  Check the
# README for detailed instructions on use.


import os
import glob
import color_detect
import size_calibration
import numpy as np
import matplotlib.pyplot as plt
import sys
import logging
from scipy.signal import savgol_filter, filtfilt
from scipy.spatial import distance
import time
import tkinter as tk
from tkinter import filedialog
import ipdb

#change per test run
#PX2IN = 2460 #px to inches based on image reference, taken from
    matlab's distance tool
bag = 9.8 #weight of bag in g
```

```
32
33 #initialize variables; don't touch these.
34 #PX2MM = PX2IN/25.4  #px to mm conversion
35 fileList = []
36 forceVal = []
37 distVal = []
38 FORCE_LABEL = 'Force (N)'
39 prop_cycle = plt.rcParams['axes.prop_cycle']
40
41 fileCycle = [0,0] #num, size
42 NUM_COLORS = 20
43
44 #creating logger; especially do not touch this unless you know what
       you're doing
45 logger = logging.getLogger(__name__)
46 handler = logging.FileHandler('temp/imgproc.log')
47 logger.setLevel(logging.INFO)
48 handler.setLevel(logging.INFO)
49 formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s
       - %(message)s')
50 handler.setFormatter(formatter)
51 logger.addHandler(handler)
52
53 #create plots for single test data
54 def singresplot(xvar, yvar, xax, yax, plotTitle, testName, path):
55     print('Generating plots...')
56     plt.figure()
57     plt.plot(xvar, yvar, linewidth=2.0)
58     plt.xlabel(xax, fontsize=14)
59     plt.ylabel(yax, fontsize=14)
60     plt.title(plotTitle, fontsize=16)
61     plt.savefig(os.path.join(path, testName + '_raw.png'))
62
```

```
63      plt.show()
64
65  #create plots with multiple test data
66  def multiplot(xvar, yvar, xax, yax, plotTitle, testName, path):
67      print('Generating single plot...')
68      plt.figure()
69      ax = plt.subplot(111)
70      cmap = plt.get_cmap('jet')
71      ax.set_color_cycle(cmap(2*i/NUM_COLORS) for i in range(NUM_COLORS
        ))
72      for j, k in enumerate(yvar):
73          ax.plot(xvar, yvar[j], linewidth=1.0, label='Trial %s' % j)
74
75      ax.set_xlabel(xax, fontsize=14)
76      ax.set_ylabel(yax, fontsize=14)
77      ax.set_title(plotTitle, fontsize=16)
78      box = ax.get_position()
79      ax.set_position([box.x0, box.y0, box.width*0.8, box.height])
80      ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
81      plt.savefig(os.path.join(path, testName + '-all.png'))
82      plt.close()
83
84  #create plots of both the mean for multiple tests and filtered linear
        -regression of multiple tests
85  def meanplot(xvar, yvar, stdError, xax, yax, plotTitle, testName,
        sgFiltered, flag, path):
86      if flag =='dist':
87          testName = testName + 'Dist'
88      else:
89          testName = testName + 'DeltaD'
90
91      print('Generating error plots...')
92      plt.figure()
```

```
93      plt.errorbar(xvar, yvar, yerr=stdError, linewidth=1.0)
94      plt.xlabel(xax, fontsize=12)
95      plt.ylabel(yax, fontsize=12)
96      plt.title(plotTitle + ' Mean, Raw', fontsize=14)
97      plt.savefig(os.path.join(path, testName + '-errorbar.png'))
98      plt.clf()
99
100     plt.errorbar(xvar, sgFiltered, yerr=stdError, linewidth=1.0)
101     plt.xlabel(xax, fontsize=12)
102     plt.ylabel(yax, fontsize=12)
103     plt.title(plotTitle, fontsize=14)
104     plt.savefig(os.path.join(path, testName + '-sg-errorbar.png'))
105     plt.clf()
106
107     plt.errorbar(xvar, yvar, yerr=stdError, linewidth=1.0, label='Raw
        ')
108     plt.plot(xvar, sgFiltered, linewidth=1.0, label='Smoothed')
109     plt.xlabel(xax, fontsize=12)
110     plt.ylabel(yax, fontsize=12)
111     #plt.xlim((0 ,1.05)) #for axis manipulation
112     plt.title(plotTitle + ' Mean', fontsize=14)
113     plt.legend(loc=0)
114     plt.savefig(os.path.join(path, testName +'-means.png'))
115     plt.close()
116
117 #code for the compression tests
118 def compression(path, px2mm, flag):
119     distVal = []
120     forceVal= []
121     deltaVal=[]
122     i = 0
123
124     for infile in glob.glob (os.path.join(path, '*.jpg')):
```

```
125         currentImage = os.path.basename(infile)
126         basepath = os.path.dirname(infile)
127         pixelVec = color_detect.cdetect(basepath, currentImage, flag)
128         fileList.append(infile)
129         fsize = os.path.getsize(infile)
130         com = pixelVec[1]
131         #calculate vertical displacement
132         diff = abs(com[0][1] - com[1][1])/px2mm
133         distVal.append(diff)
134         if i == 0:
135             delta = 0
136         else:
137             delta = abs(distVal[0] - diff)
138         force = i*2.5*10**-3*9.81 # # of pennies * weight * g
139         forceVal.append(force)
140         deltaVal.append(delta)
141         i = i+1 #thank you python for being 0-index
142         fileCycle[0] = fileCycle[0] + 1
143         fileCycle[1] = fileCycle[1] + fsize
144     return(distVal, forceVal, deltaVal)
145
146 #tests that require using an angle: shear and bending; use flag to
        determine which one
147 def thetadep(path, px2mm, flag):
148     distVal = []
149     forceVal= []
150     deltaVal= []
151     i = 0
152     j = 0
153
154     for infile in glob.glob (os.path.join(path, '*.jpg')):
155         currentImage = os.path.basename(infile)
156         basepath = os.path.dirname(infile)
```

```
157         pts = color_detect.cdetect(basepath, currentImage, flag)
158         fileList.append(infile)
159         fsize = os.path.getsize(infile)
160 
161         if flag == 2:
162             if len(pts) < 2:
163                 diff = np.nan
164             else:
165                 x1 = pts[1][0] - pts[0][0]
166                 y1 = pts[1][1] - pts[0][1]
167                 theta = np.arctan2(y1, x1)
168                 hyp = distance.euclidean(pts[0], pts[1])
169                 diff = abs(hyp*np.sin(theta))/px2mm
170         else:
171             u1 = (pts[0][0][1] - pts[0][1][0])
172             u2 = (pts[0][1][1] - pts[0][0][1])
173             vector1 = (u1, u2)
174             v1 = (pts[1][0][1] - pts[1][1][0])
175             v2 = (pts[1][1][1] - pts[1][0][1])
176             vector2 = (v1, v2)
177             dotV = np.dot(vector1, vector2)
178             mag1 = np.sqrt(np.dot(vector1, vector1))
179             mag2 = np.sqrt(np.dot(vector2, vector2))
180             #why it's measuring from 180 I have no idea
181             diff = np.rad2deg(np.arccos(dotV/(mag1*mag2)))
182             if diff-180 >= 0:
183                 diff = 360 - diff
184 
185         distVal.append(np.float64(diff))
186 
187         if j == 0:
188             force = 0
189             delta = 0
```

```
190         else:
191             delta = abs(distVal[0] - diff)
192             force = ((i*2.5)+bag)*10**-3*9.81
193             i = i+1
194 
195         forceVal.append(force)
196         deltaVal.append(delta)
197         j = j+1
198         fileCycle[0] = fileCycle[0] + 1
199         fileCycle[1] = fileCycle[1] + fsize
200     return(distVal, forceVal, deltaVal)
201 
202 def statanalysis(subDist, forceVal, testNum, studyID, testName,
    distLabel, path, flag):
203     #and here we do the actual statistical analysis of the aggretate
    data
204     subDist = np.array(subDist)
205     for x in np.nditer(subDist, op_flags=['readwrite']):
206         if flag == 1:
207             if x < 1.:
208                 x[...] = np.nan
209         elif flag == 2:
210             if x > 20:
211                 x[...] = np.nan
212         else:
213             continue
214 
215     distMean = np.nanmean(subDist, axis=0, dtype=np.float64)
216     std = np.nanstd(subDist, axis=0, dtype=np.float64)
217     stdError = std/np.sqrt(testNum)
218 
219     #Savitzky-Golay filtering (window size: 13, polyorder: 5)
220     sgDistFilt = savgol_filter(distMean, 13, 5, axis=0)
```

```
221     #manipulate arrays to export raw data to .csv files for
        importation into other software
222     if flag == 1:
223         forV = np.array(forceVal)
224         multiplot(forceVal, subDist, FORCE_LABEL, distLabel, studyID,
         testName, path)
225         meanplot(forceVal, distMean, stdError, FORCE_LABEL, distLabel
        , studyID, testName, sgDistFilt, 'dist', path)
226     else:
227         forV = np.array(forceVal[0])
228         multiplot(forceVal[0], subDist, FORCE_LABEL, distLabel,
        studyID, testName, path)
229         meanplot(forceVal[0], distMean, stdError, FORCE_LABEL,
        distLabel, studyID, testName, sgDistFilt, 'dist', path)
230     dataCombo = np.c_[forV, subDist.T]
231     np.savetxt(os.path.join(path,testName +'.csv'), dataCombo,
        delimiter=',')
232     np.savetxt(testName +'.csv', dataCombo, delimiter=',')
233
234
235 #the difference here is that this section takes the delta distance (
        displacement) and plots it
236 def statdisplacement(sDisplace, forceVal, testNum, studyID, testName,
         displabel, path):
237     sDisplace = np.array(sDisplace)
238     for x in np.nditer(sDisplace, op_flags=['readwrite']):
239         if x > 100:
240             x[...] = np.nan
241     displaceMean = np.nanmean(sDisplace, axis=0, dtype=np.float64)
242     dispSTD = np.nanstd(sDisplace, axis=0, dtype=np.float64)
243     dispSTDError = dispSTD/np.sqrt(testNum)
244     sgDisplaceFilt = savgol_filter(displaceMean, 13, 5, axis=0)
245     meanplot(forceVal, displaceMean, dispSTDError, FORCE_LABEL,
```

```
    displLabel, studyID, testName, sgDisplaceFilt, 'disp', path)

def statdisplacementalt(sDisplace, subDist, forceVal, testNum,
    studyID, testName, displLabel, path, flag):
    subDist = np.array(subDist)
    sDisplace = np.array(sDisplace)
    for x in np.nditer(sDisplace, op_flags=['readwrite']):
        if x > 100:
            x[...] = np.nan
    for x in np.nditer(subDist, op_flags=['readwrite']):
        if flag == 1:
            if x < 1.:
                x[...] = np.nan
        elif flag == 2:
            if x > 20:
                x[...] = np.nan
        else:
            continue
    distMean = np.nanmean(subDist, axis=0, dtype=np.float64)
    displaceMean = distMean - distMean[0]
    dispSTD = np.nanstd(subDist, axis=0, dtype=np.float64)
    dispSTDError = dispSTD/np.sqrt(testNum)
    sgDisplaceFilt = savgol_filter(displaceMean, 13, 5, axis=0)
    meanplot(forceVal, displaceMean, dispSTDError, FORCE_LABEL,
    displLabel, studyID, testName, sgDisplaceFilt, 'disp', path)

def main():
    compBool = False
    root = tk.Tk()
    root.withdraw()
    path = filedialog.askdirectory(parent=root,initialdir="/",title='
    Please select a directory')
    if not path:
```

```
275         sys.exit()
276 
277     while True:
278     #python why are you case sensitive?
279         testType = input('Test type (COMpression, SHear, BENding): ')
280         if testType not in {'COM', 'SH', 'BEN', 'com', 'sh', 'ben'}:
281             logger.warn('Incorrect input; entered as %s', testType)
282             continue
283         else:
284             break
285     while True:
286         compTest = input('Comprehensive (y/n) ?')
287         if compTest not in {'y', 'n', 'Y', 'N'}:
288             compTest = input('Comprehensive (y/n)?')
289             continue
290         if compTest in ['y', 'Y']:
291             compBool = True
292             break
293         else:
294             break
295     while True:
296         testName = input('Save as filename.png: ')
297         if not testName:
298             testName = input('Save as filename.png: ')
299             continue
300         else:
301             break
302 
303     #this is the worst hack ever
304     if compBool:
305         subDist = []
306         subForce = []
307         sDisplace = []
```

```
308         topdir = path
309         subdir = [os.path.abspath(x[0]) for x in os.walk(topdir)]
310         subdir.remove(os.path.abspath(topdir))
311         testNum = len(subdir)
312         logger.info('Compilation run start')
313         startTime = time.time()
314 
315         if testType in ('COM', 'com'):
316             typeFlag = 1
317             for currentdir in subdir:
318                 print(currentdir)
319                 topimg = os.listdir(currentdir)[0]
320                 #px2mm = PX2MM #toggle if reference swatch does not
    exist
321                 px2mm = size_calibration.sizecalib(currentdir, topimg
    , typeFlag)
322                 distVal, forceVal, deltaVal = compression(currentdir,
     px2mm, typeFlag)
323                 subDist.append(distVal)
324                 subForce.append(forceVal)
325                 sDisplace.append(deltaVal)
326             studyID = 'Compression Test (trials: {0})'.format(testNum
    )
327             distLabel = 'Distance (mm)'
328             dispLabel = 'Displacement $\Delta d$ (mm)'
329             statanalysis(subDist, forceVal, testNum, studyID,
    testName, distLabel, path, typeFlag)
330             statdisplacement(sDisplace, forceVal, testNum, studyID,
    testName, dispLabel,path)
331 
332         elif testType in ('SH', 'sh'):
333             planeID = input('Shear plane: ')
334             typeFlag = 2
```

```
335             for currentdir in subdir:
336                 print(currentdir)
337                 topimg = os.listdir(currentdir)[0]
338                 px2mm = size_calibration.sizecalib(currentdir, topimg
    , typeFlag)
339                 distVal, forceVal, deltaVal = thetadep(currentdir,
    px2mm, typeFlag)
340                 subDist.append(distVal)
341                 subForce.append(forceVal)
342                 sDisplace.append(deltaVal)
343             studyID = 'Shear Test (trials: {0}), {shear}-plane'.
    format(testNum, shear = planeID)
344             testName = testName + '-{shear}'.format(shear = planeID)
345             distLabel = 'Distance (mm)'
346             dispLabel = 'Displacement $\Delta d$ (mm)'
347             statanalysis(subDist, subForce, testNum, studyID,
    testName, distLabel, path, typeFlag)
348             #I don't know why the displacement adds a factor of ten
    to a simple subtraction, dear self, wtf
349             #statdisplacement(sDisplace, forceVal, testNum, studyID,
    testName, dispLabel, path)
350             #for when statdisplacement isn't working normally
351             statdisplacementalt(sDisplace, subDist, forceVal, testNum
    , studyID, testName, dispLabel,path,typeFlag)
352
353         else:
354             typeFlag = 3
355             for currentdir in subdir:
356                 print(currentdir)
357                 topimg = os.listdir(currentdir)[0]
358                 px2mm = size_calibration.sizecalib(currentdir, topimg
    , typeFlag)
359                 distVal, forceVal, delta = thetadep(currentdir, px2mm
```

```
, typeFlag)
                subDist.append(distVal)
                subForce.append(forceVal)
                sDisplace.append(delta)
            studyID = 'Bending Test (trials: {0})'.format(testNum)
            distLabel = (r'Angle ${\theta}$ (deg)')
            displLabel = (r'${\Delta} {\theta} $ (deg)')
            statanalysis(subDist, subForce, testNum, studyID,
testName, distLabel, path, typeFlag)
            statdisplacement(sDisplace, forceVal, testNum, studyID,
testName, displLabel, path)

        logger.info('Compilation run end')
        print('Processed {} files ({} bytes). Runtime: {} seconds.'.
format(
            fileCycle[0] , fileCycle[1], (time.time() - startTime)))

    else:

        while True:
            if not os.path.isdir(path):
                print('Not a valid subdirectory.')
                logger.warn('Input: Nonexistant subdir')
                continue
            else:
                currentdir = path
                startTime = time.time()
                topimg = os.listdir(currentdir)[0]


                if testType in ('COM', 'com'):
                    typeFlag = 1
                    px2mm = size_calibration.sizecalib(currentdir,
```

```
        topimg, typeFlag)
389                     distVal, forceVal, deltaVal = compression(path,
        px2mm, typeFlag)
390                     studyID = 'Compression Test'
391                 elif testType in ('SH', 'sh'):
392                     planeID = input('Shear plane: ')
393                     typeFlag = 2
394                     px2mm = size_calibration.sizecalib(currentdir,
        topimg, typeFlag)
395                     distVal, forceVal, deltaVal = thetadep(path,
        px2mm, typeFlag)
396                     studyID = 'Shear Test'
397                     testName = testname + '_{}'.format(planeID)
398                 else:
399                     typeFlag = 3
400                     px2mm = size_calibration.sizecalib(currentdir,
        topimg, typeFlag)
401                     distVal, forceVal, deltaVal = thetadep(path,
        px2mm, typeFlag)
402                     studyID = 'Bending Test'
403
404         singresplot(forceVal, distVal, FORCE_LABEL, distLabel,
        studyID, testName, path)
405         print('Processed {} files ({} bytes). Runtime: {} seconds.'.
        format(
406             fileCycle[0] , fileCycle[1], (time.time() - startTime)))
407
408 main()
```

code/color_detect.py

```
1 #import packages
2 import numpy as np
3 import argparse
```

```
import cv2
import os
from operator import itemgetter
from imutils import perspective
from imutils import contours as cnts
import ipdb

def cdetect(basepath, currentimage, testFlag):

    path = os.path.join(basepath, currentimage)
    image = cv2.imread(path)

    #compression
    if testFlag == 1:
        #define boundaries of RGB (in BGR order)
        boundaries = ([17,15,95], [90, 100, 255]) #red

        lower = np.array(boundaries[0], dtype="uint8")
        upper = np.array(boundaries[1], dtype="uint8")
        #find colors within boundaries, apply mask
        mask = cv2.inRange(image, lower, upper)
        output = cv2.bitwise_and(image, image, mask = mask)
        gray = cv2.cvtColor(output, cv2.COLOR_BGR2GRAY)
        gray = cv2.bilateralFilter(gray, 11, 17, 17)

        ret, thresh1 = cv2.threshold(gray, 15, 255, cv2.THRESH_BINARY
    )
        kernel = np.ones((5,5), np.uint8)
        erosion = cv2.erode(thresh1, kernel, iterations = 1)
        opening = cv2.morphologyEx(erosion, cv2.MORPH_OPEN, kernel)
        closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel)

        #determine pixel coordinates
```

```
36          _, bounds, hierarchy = cv2.findContours(closing, cv2.
    RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
37          cv2.drawContours(closing, bounds, -1, (255, 255, 255), 3)
38          com = []
39          threshpass = []
40          threshold_area = 1000
41          for i, cnt in enumerate(bounds):
42              area = cv2.contourArea(cnt)
43              if area > threshold_area:
44                  threshpass.append((area, bounds[i]))
45
46          threshpass.sort(key=itemgetter(0), reverse=True)
47          for x in range(0, 2):
48              moi = cv2.moments(threshpass[x][1])
49              com.append((int(moi['m10']/moi['m00']), int(moi['m01']/
    moi['m00'])))
50
51          return bounds, com
52
53      #shear
54      elif testFlag == 2:
55          centPts = []
56          #define boundaries of RGB (in BGR order)
57          boundaries = ([15,30,95], [90, 70, 255]) #red
58          image = cv2.resize(image, None, fy = 1/6, fx=1/6,
    interpolation=cv2.INTER_CUBIC)
59          imgW, imgH, channels = image.shape
60          roiW = int(np.asarray(imgW) * 0.2)
61          roiH = int(np.asarray(imgH) * 0.5)
62          image = image[0:imgH, roiW:imgW]
63
64          lower = np.array(boundaries[0], dtype="uint8")
65          upper = np.array(boundaries[1], dtype="uint8")
```

```
66          mask = cv2.inRange(image, lower, upper)
67          output = cv2.bitwise_and(image, image, mask = mask)
68
69          gray = cv2.cvtColor(output, cv2.COLOR_BGR2GRAY)
70          gray = cv2.GaussianBlur(gray, (5, 5), 0)
71          edged = cv2.Canny(gray, 50, 150)
72          edged = cv2.dilate(edged, None, iterations = 1)
73          edged = cv2.erode(edged, None, iterations = 1)
74
75          conts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2
    .CHAIN_APPROX_SIMPLE)
76          conts = conts[1]
77          (conts, _) = cnts.sort_contours(conts)
78
79          for c in conts:
80              if cv2.contourArea(c) < 100:
81                  continue
82
83              x1,y1,w1,h1 = cv2.boundingRect(c)
84              midpt = ((x1+w1/2), (y1+h1))
85              centPts.append(midpt)
86
87          return centPts
88
89      #bending
90      else:
91          linePts = []
92          boundaries = ([15,30,95], [90, 70, 255]) #red
93          image = cv2.resize(image, None, fy = 1/6, fx=1/6,
    interpolation=cv2.INTER_CUBIC)
94          lower = np.array(boundaries[0], dtype="uint8")
95          upper = np.array(boundaries[1], dtype="uint8")
96          mask = cv2.inRange(image, lower, upper)
```

```
97          output = cv2.bitwise_and(image, image, mask = mask)
98
99          gray = cv2.cvtColor(output, cv2.COLOR_BGR2GRAY)
100         gray = cv2.GaussianBlur(gray, (3, 3), 0)
101         edged = cv2.Canny(gray, 50, 150)
102         edged = cv2.dilate(edged, None, iterations = 1)
103         edged = cv2.erode(edged, None, iterations = 1)
104
105         conts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2
    .CHAIN_APPROX_SIMPLE)
106         conts = conts[1]
107         (conts, _) = cnts.sort_contours(conts)
108
109         for c in conts:
110             if cv2.contourArea(c) < 100:
111                 continue
112             box = cv2.minAreaRect(c)
113             box = cv2.boxPoints(box)
114             box = np.array(box, dtype='int')
115             cv2.drawContours(image, [box.astype('int')], -1, (0, 255,
     0), 2)
116
117             (tl, tr, br, bl) = box
118             (lx, ly) = ((tl[0] + tr[0])*0.5, (tl[1] + tr[1])*0.5)
119             (rx, ry) = ((bl[0] + br[0])*0.5, (bl[1] + br[1])*0.5)
120             subline = [(lx, ly), (rx, ry)]
121             cv2.line(image, (int(lx),int(ly)), (int(rx),int(ry)), (0,
     255, 255), 2)
122             linePts.append(subline)
123
124         return linePts
```

code/size_calibration.py

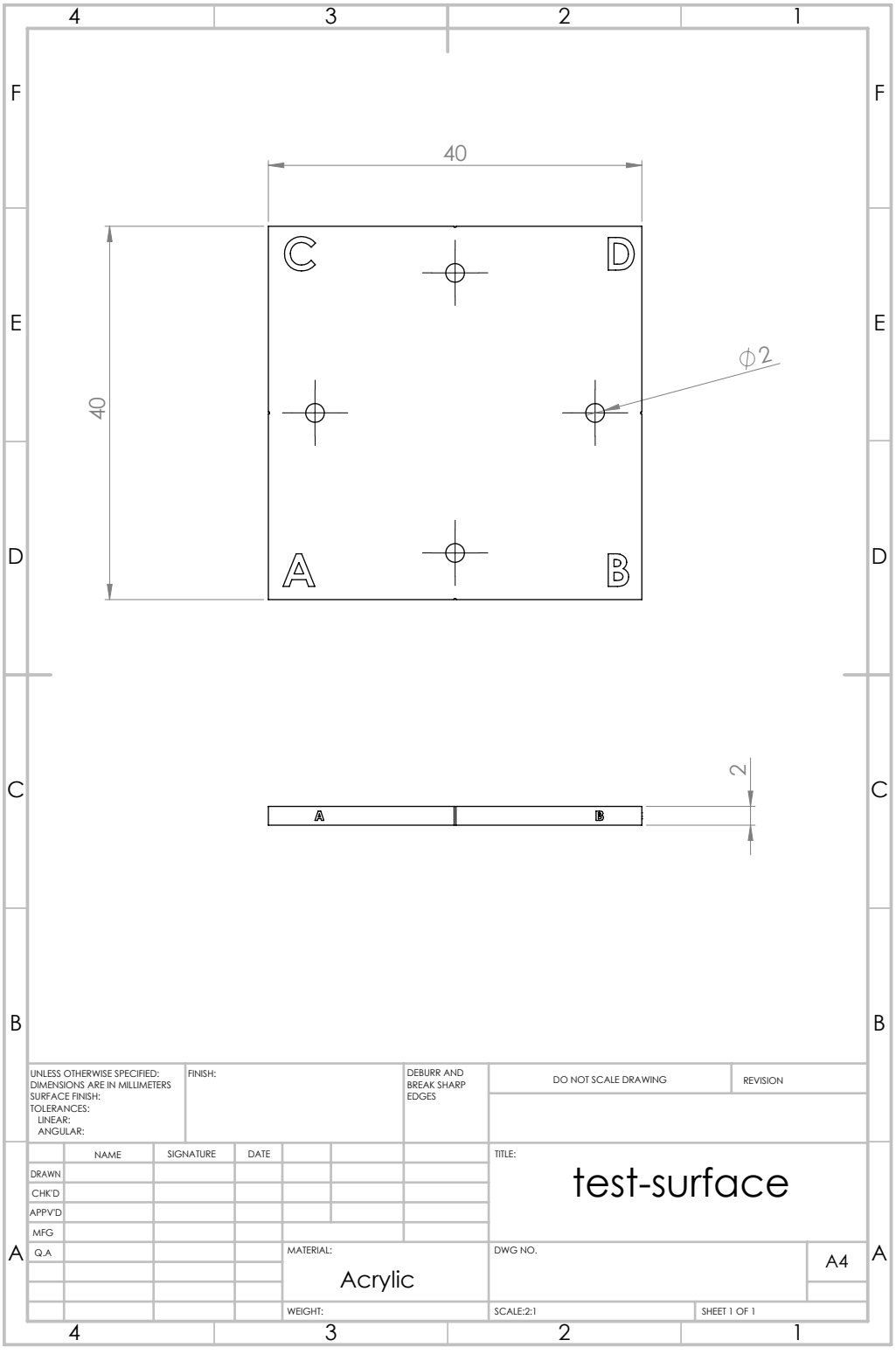
```
import os
from scipy.spatial import distance as dist
from imutils import perspective
from imutils import contours as cnts
import numpy as np
import imutils
import cv2
import ipdb

def midpt(ptA, ptB):
        return((ptA[0] + ptB[0])*0.5, (ptA[1] + ptB[1])*0.5)

def sizecalib(basepath, currentimage, flag):
        realWidth = 10 #actual size of white swatch in mm

        path = os.path.join(basepath, currentimage)
        image = cv2.imread(path)
        imgW, imgH, channels = image.shape
        if flag == 1:
                roiW = int(np.asarray(imgW) * 0.75)
                roiH = int(np.asarray(imgH) * 0.5)
                roiImage = image[roiH:imgH, 0:roiW]
                whiteBound = ([200, 200, 200], [255, 255, 255])
                areaThresh = 1000
        elif flag == 2:
                image = cv2.resize(image, None, fy = 1/6, fx=1/6,
    interpolation=cv2.INTER_CUBIC)
                roiW = int(np.asarray(imgW)*0.50)
                roiH = int(np.asarray(imgH)*0.30)
                roiImage = image[0:roiH, 0:roiW]
                whiteBound = ([200, 200, 200], [255, 255, 255])
                areaThresh = 1000
        else:
```

```
33              roiW = int(np.asarray(imgW)*0.75)
34              roiH = int(np.asarray(imgH)*0.30)
35              roiImage = image[roiH:imgH, 0:roiW]
36              whiteBound = ([145, 145, 145], [255, 255, 255]) #
    shadowing is a problem
37              areaThresh = 1000
38
39      lower = np.array(whiteBound[0], dtype = 'uint8')
40      upper = np.array(whiteBound[1], dtype = 'uint8')
41
42      mask = cv2.inRange(roiImage, lower, upper)
43      output = cv2.bitwise_and(roiImage, roiImage, mask = mask )
44      gray = cv2.cvtColor(output, cv2.COLOR_BGR2GRAY)
45      gray = cv2.GaussianBlur(gray, (3, 3), 0)
46      edged = cv2.Canny(gray, 50, 150)
47      edged = cv2.dilate(edged, None, iterations = 1)
48      edged = cv2.erode(edged, None, iterations = 1)
49
50      contours = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
51      contours = contours[0] if imutils.is_cv2() else contours[1]
52      (contours, _) = cnts.sort_contours(contours)
53
54      assert len(contours) != 0
55      for c in contours:
56              if cv2.contourArea(c) < areaThresh:
57                      continue
58
59              orig = roiImage.copy()
60              box = cv2.minAreaRect(c)
61              box = cv2.cv.BoxPoints(box) if imutils.is_cv2() else
    cv2.boxPoints(box)
62              box = np.array(box, dtype='int')
```

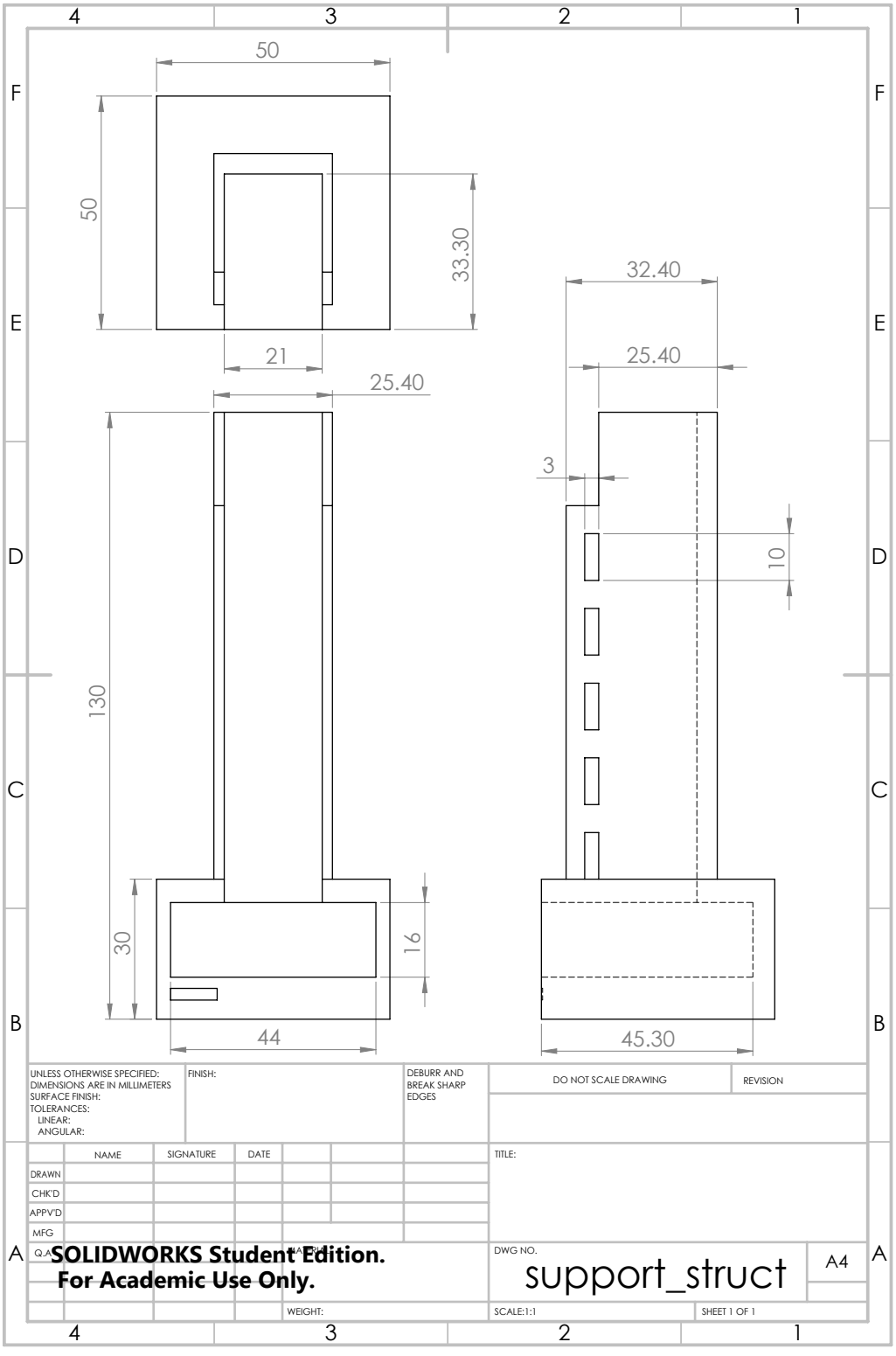
```
63                box = perspective.order_points(box)
64                cv2.drawContours(orig, [box.astype('int')], -1, (0,
    255, 0), 2)
65
66                for (x, y) in box:
67                        cv2.circle(orig, (int(x), int(y)), 5, (0, 0,
    255), -1)
68
69                (tl, tr, br, bl) = box
70                (topX, topY) = midpt(tl, tr)
71                (botX, botY) = midpt(bl, br)
72                (leftX, leftY) = midpt(tl, bl)
73                (rightX, rightY) = midpt(tr, br)
74
75                distA = dist.euclidean((topX, topY), (botX, botY))
76                distB = dist.euclidean((leftX, leftY), (rightX,
    rightY))
77
78                if flag == 2:
79                        pxCalib = distA/realWidth
80                else:
81                        pxCalib = distB/realWidth
82
83        return(pxCalib)
```

CHAPTER B

Model Drawings

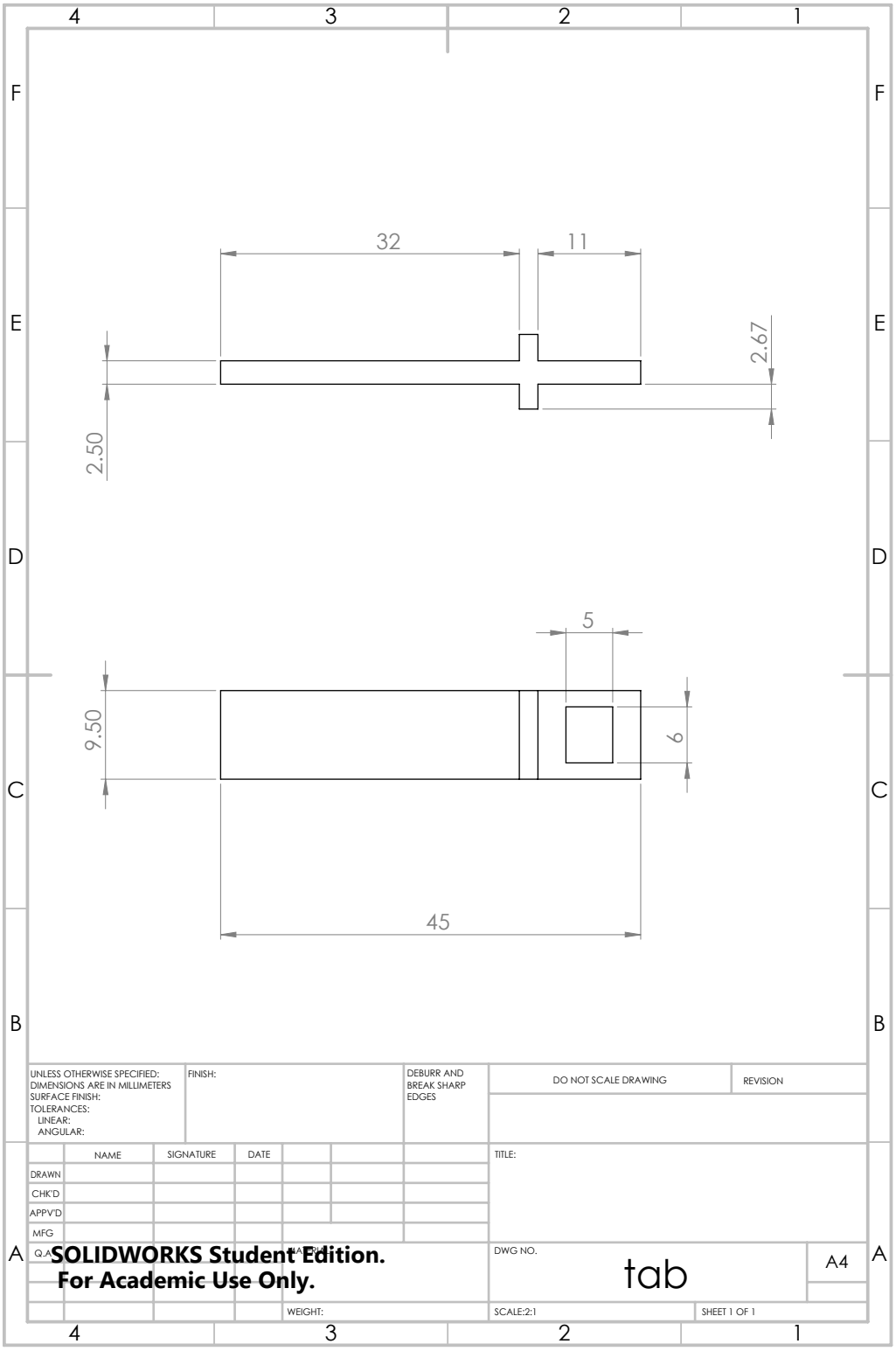


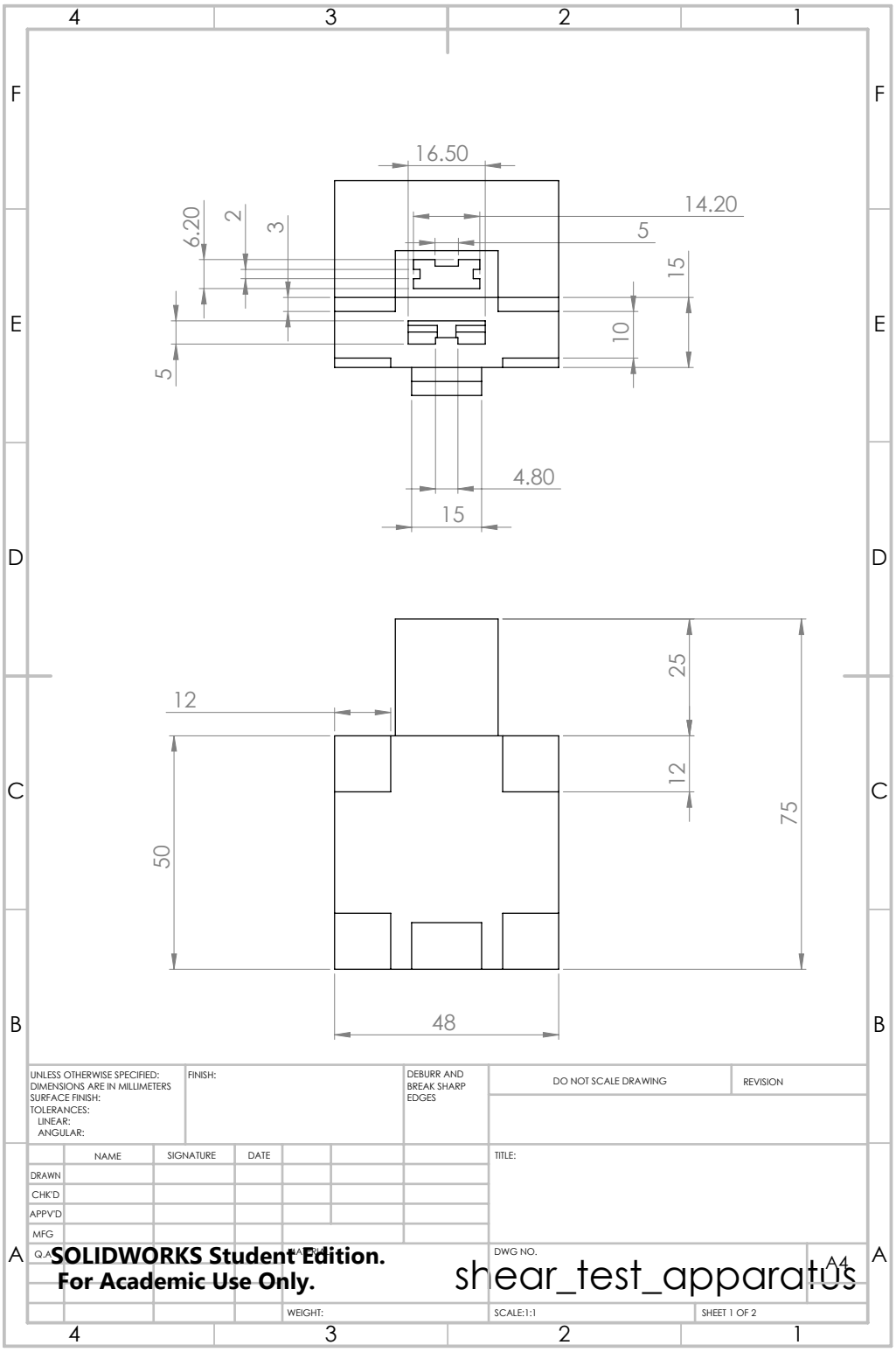
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE	TITLE:	
DRAWN				test-surface	
CHK'D					
APPV'D					
MFG					
Q.A					
MATERIAL: Acrylic				DWG NO.	A4
WEIGHT:				SCALE:2:1	SHEET 1 OF 1



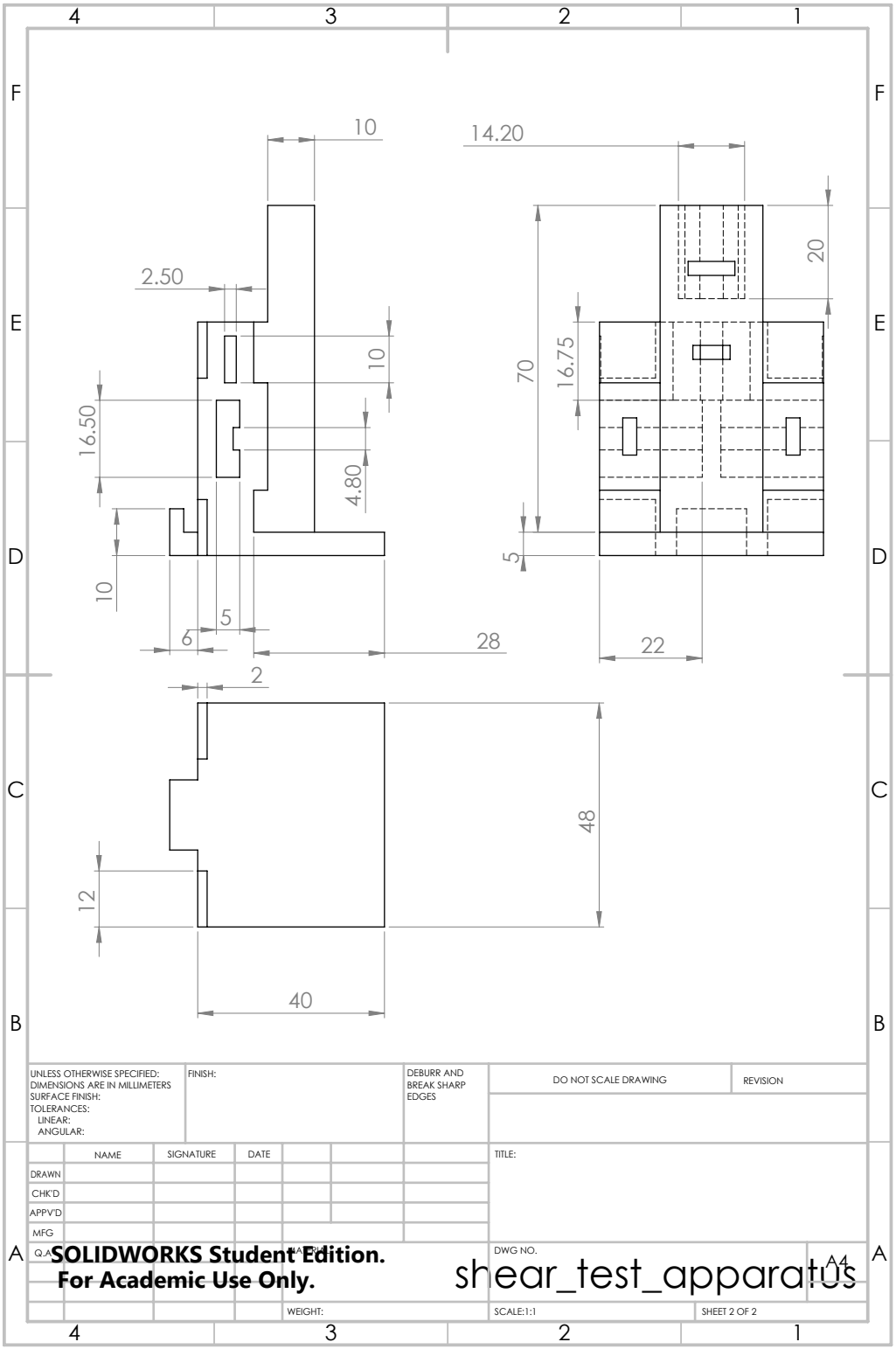
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE				TITLE:
DRAWN							
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL		DWG NO.	A4
				WEIGHT:		SCALE:1:1	SHEET 1 OF 1

support_struct

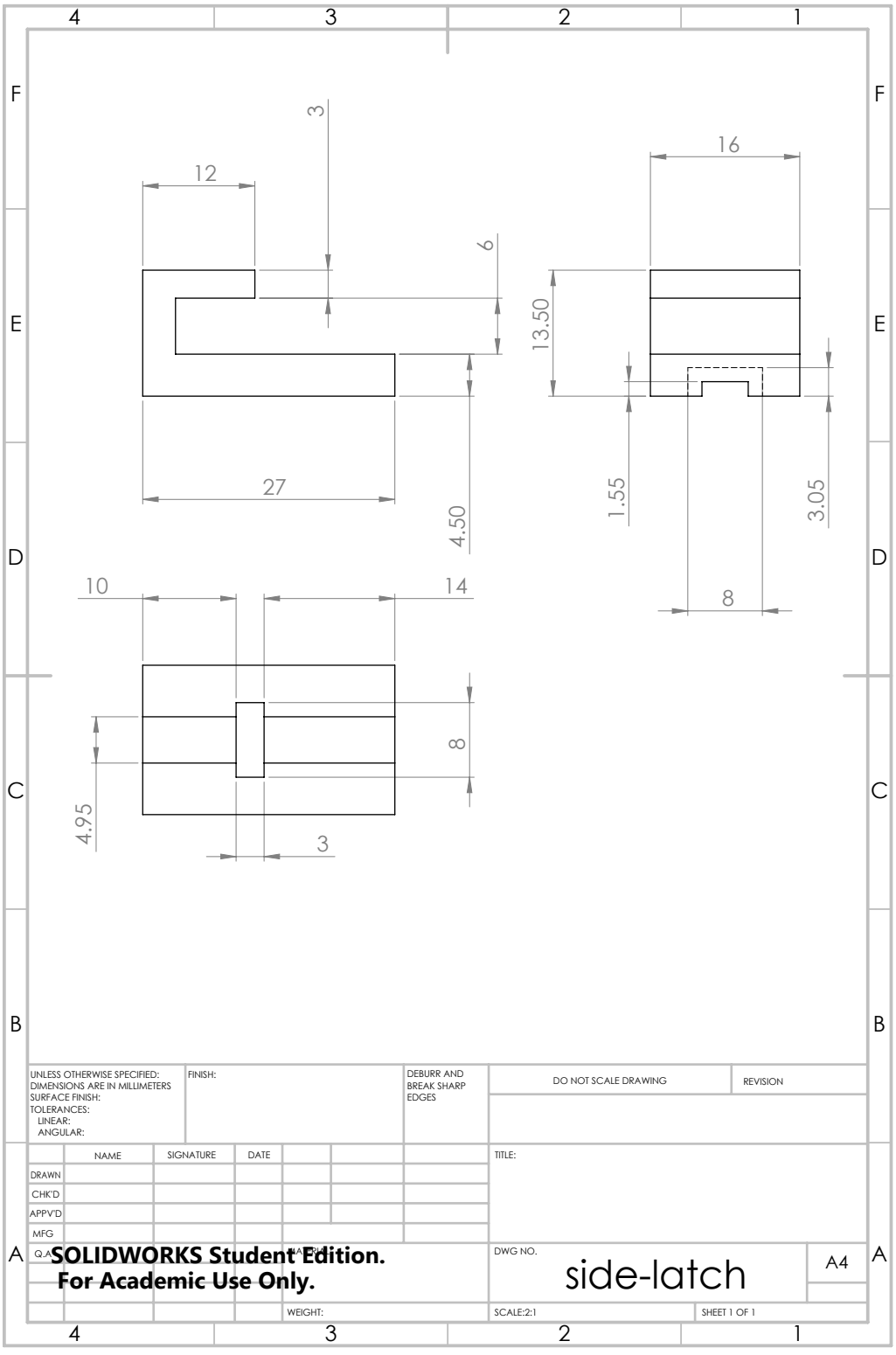

4
3
2
1
F
E
D
C
B
A
32
11
2.67
2.50
5
9.50
6
45
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL
TITLE:
DWG NO.
tab
A4
WEIGHT:
SCALE:2:1
SHEET 1 OF 1
SOLIDWORKS Student Edition.
For Academic Use Only.



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING		REVISION
	NAME	SIGNATURE	DATE				TITLE:	
DRAWN								
CHK'D								
APPV'D								
MFG								
Q.A				MATERIAL			DWG NO.	A4
							shear_test_apparatus	
				WEIGHT:		SCALE:1:1	SHEET 1 OF 2	

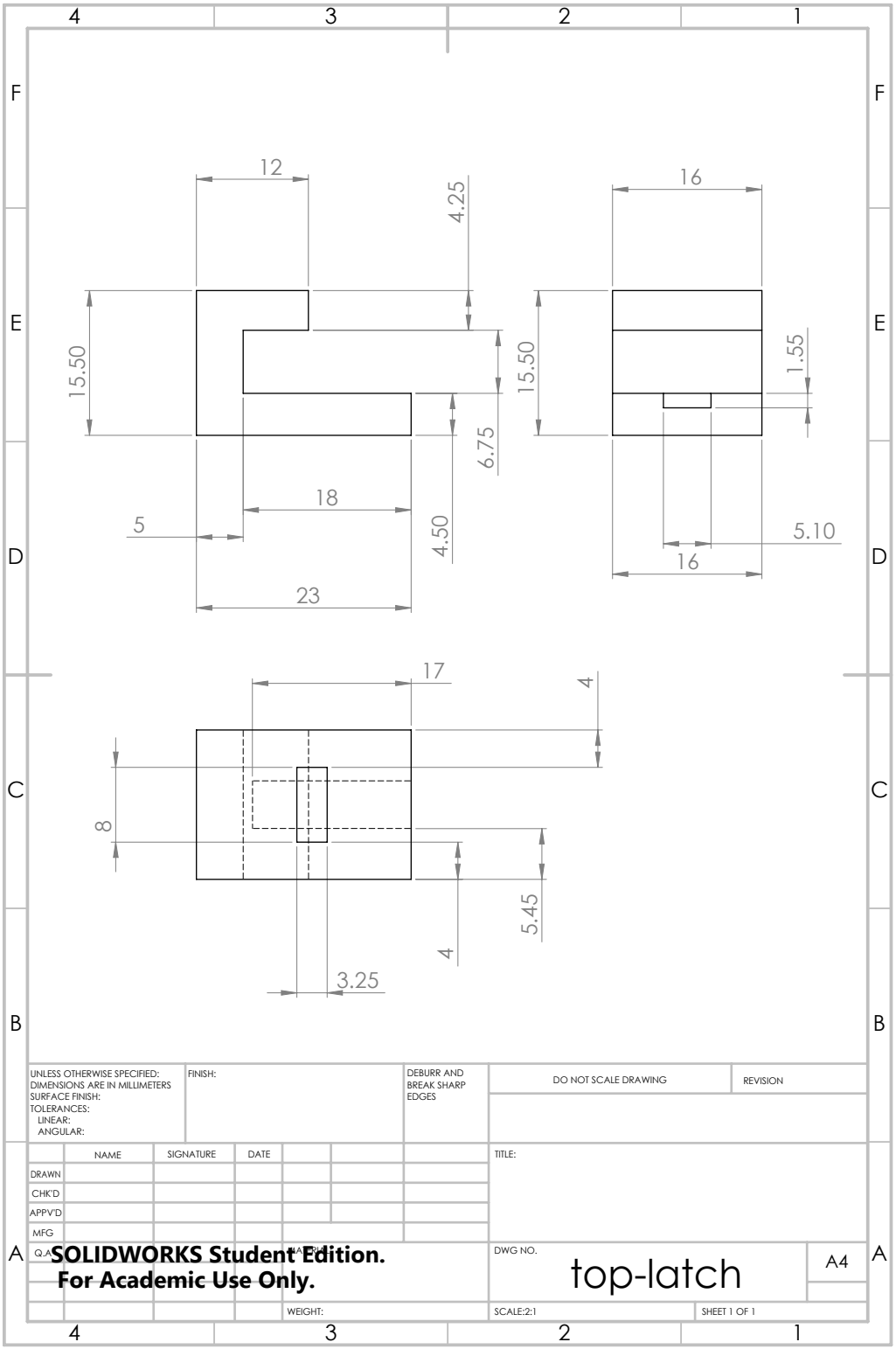


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:			DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE				TITLE:	
DRAWN								
CHK'D								
APPV'D								
MFG								
Q.A				MATERIAL			DWG NO.	A4
							shear_test_apparatus	
				WEIGHT:			SCALE:1:1	SHEET 2 OF 2



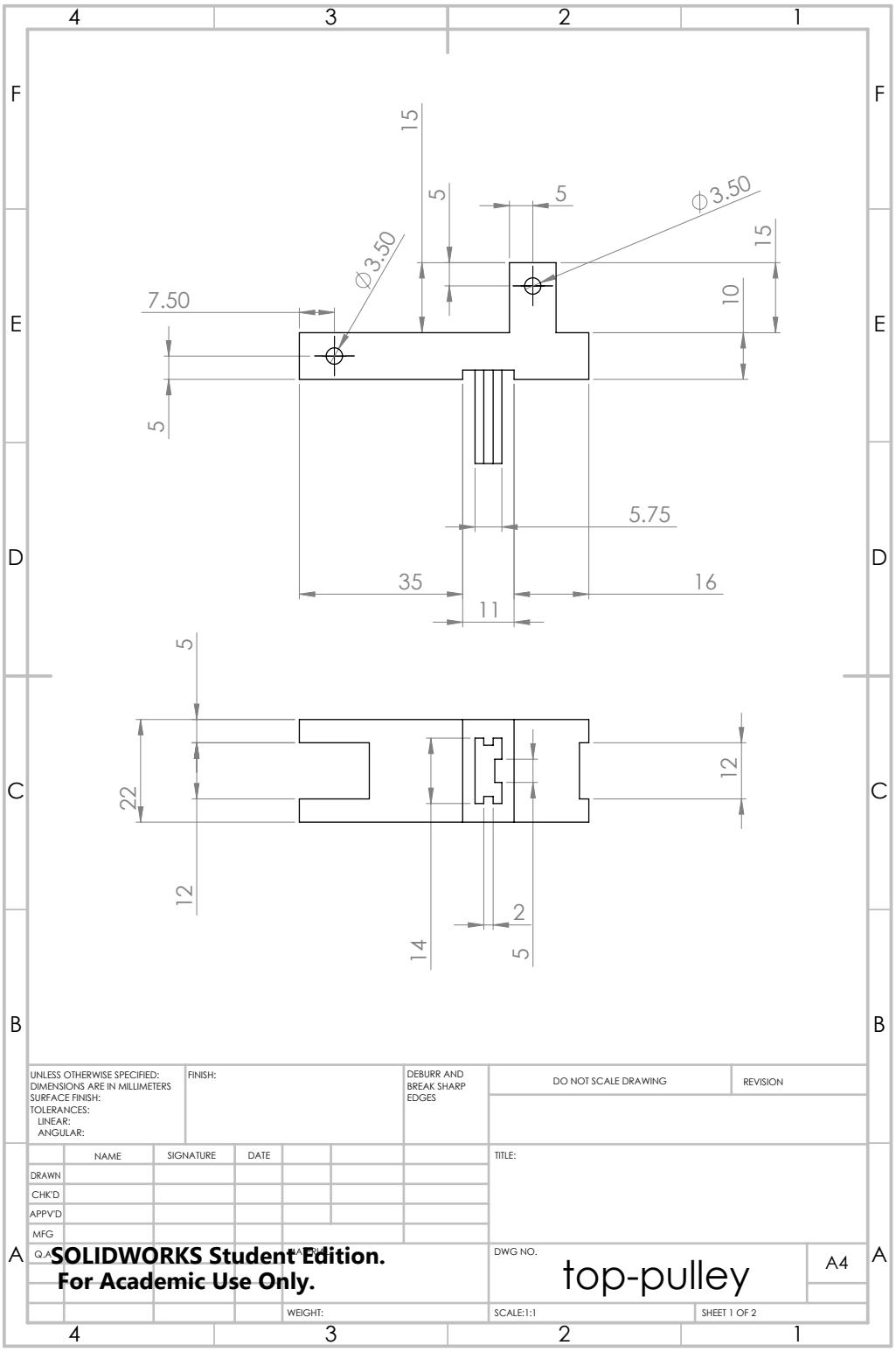
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE			TITLE:	
DRAWN							
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL		DWG NO.	A4
				WEIGHT:		SCALE:2:1	SHEET 1 OF 1

side-latch



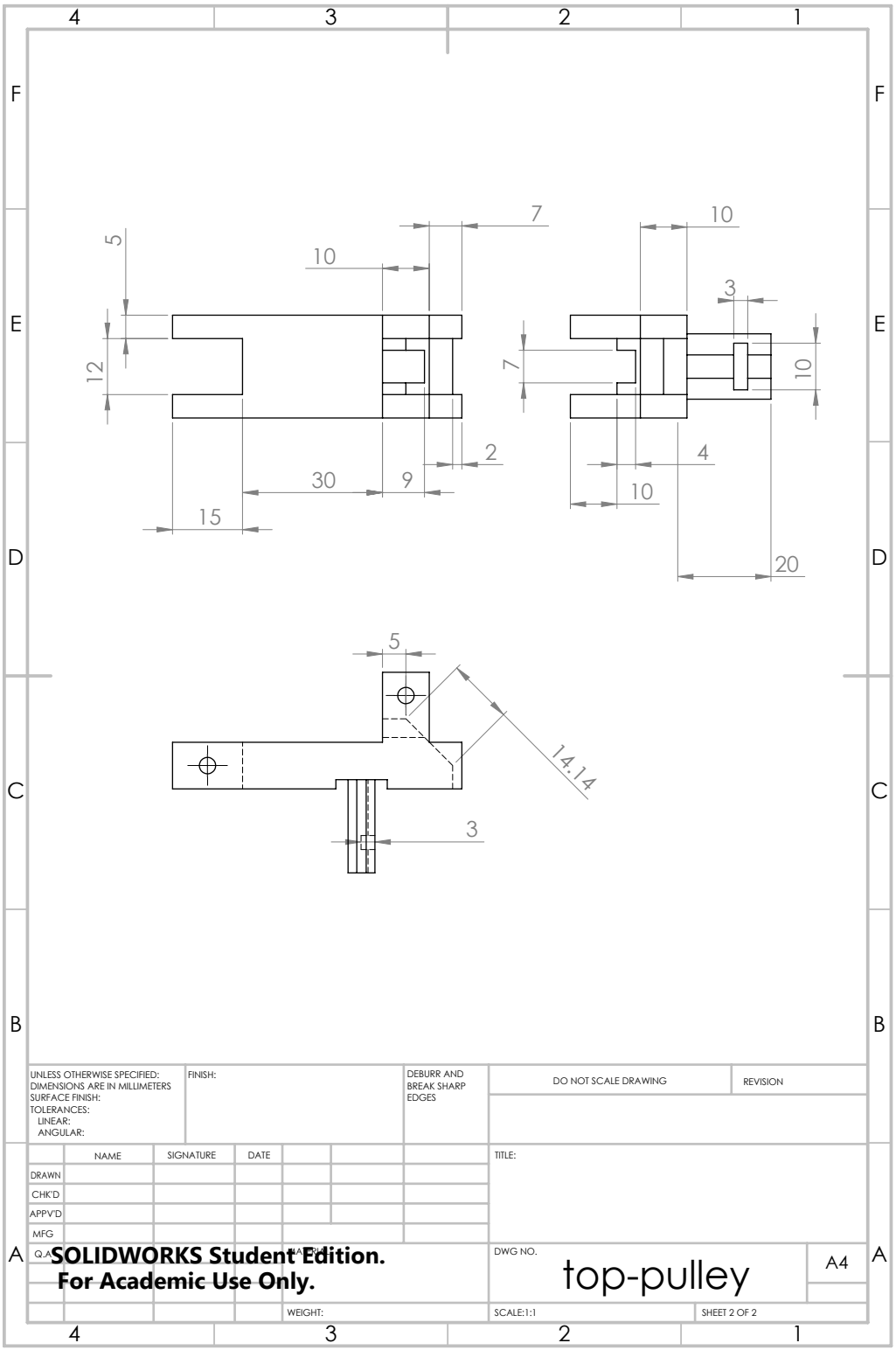
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE				TITLE:
DRAWN							
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL		DWG NO.	A4
				WEIGHT:		SCALE:2:1	SHEET 1 OF 1

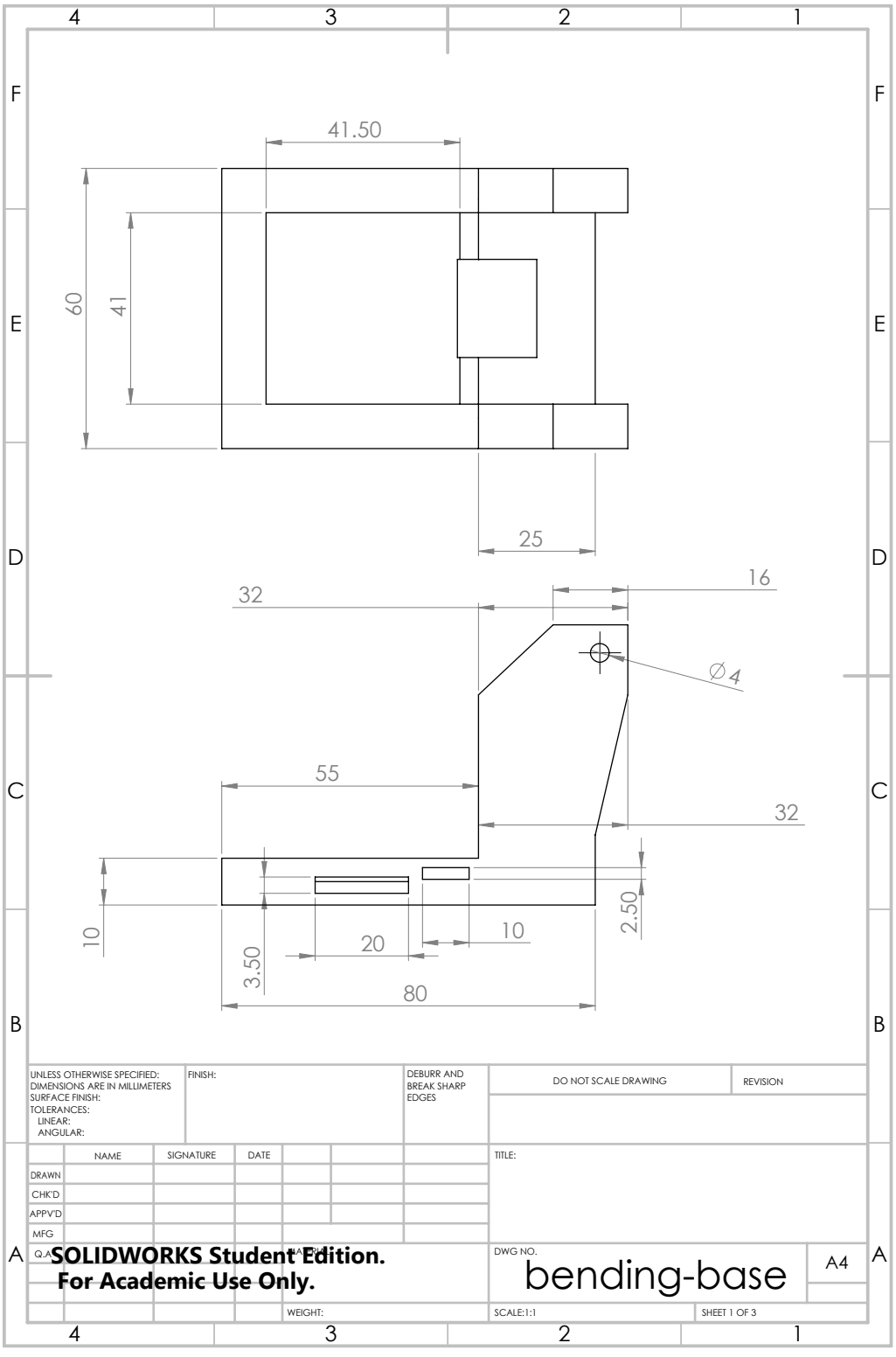
top-latch

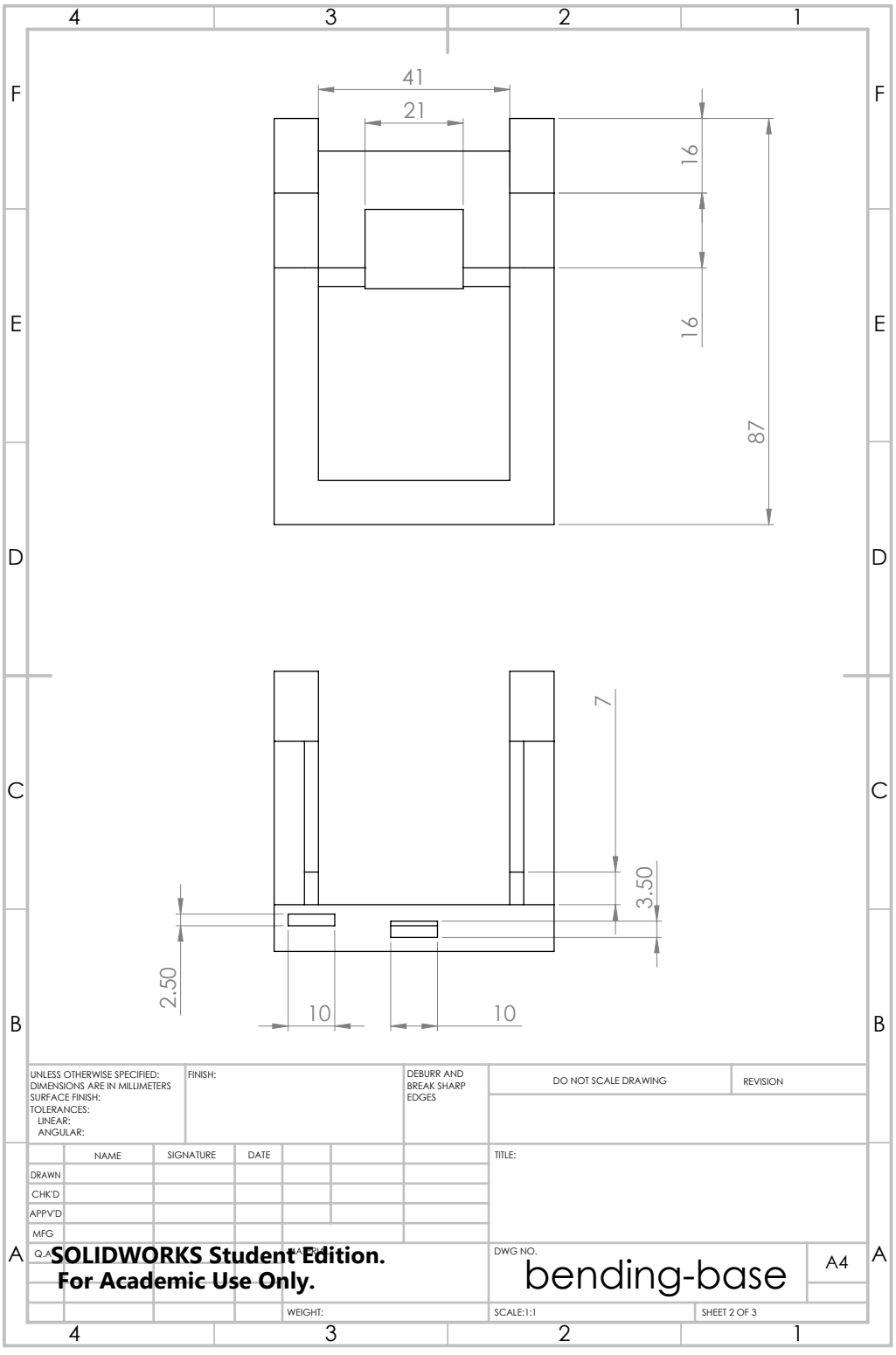


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE			TITLE:	
DRAWN							
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL		DWG NO.	A4
				WEIGHT:		SCALE:1:1	SHEET 1 OF 2

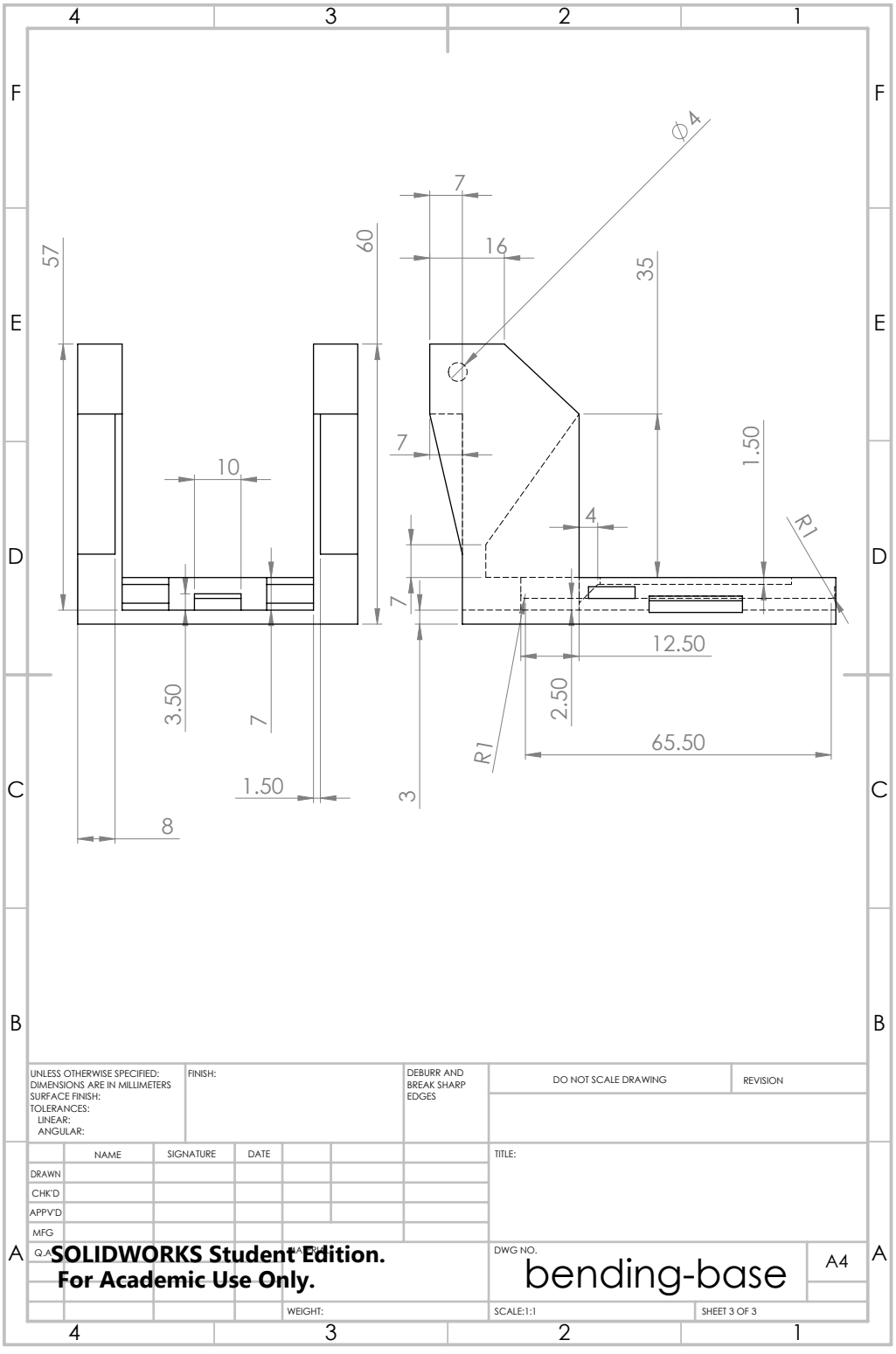
top-pulley


4
3
2
1
F
E
D
C
B
A
7
10
5
10
3
12
7
10
2
4
30
9
10
15
20
5
14.14
3
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL
TITLE:
SOLIDWORKS Student Edition.
For Academic Use Only.
DWG NO.
top-pulley
A4
WEIGHT:
SCALE:1:1
SHEET 2 OF 2


41.50
60
41
25
32
16
∅4
55
32
10
2.50
20
10
3.50
80
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL
TITLE:
DWG NO.
bending-base
A4
WEIGHT:
SCALE:1:1
SHEET 1 OF 3
SOLIDWORKS Student Edition.
For Academic Use Only.



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE	TITLE:	
DRAWN					
CHK'D					
APPV'D					
MFG					
Q.A					
		MATERIAL		DWG NO. bending-base	A4
		WEIGHT:		SCALE:1:1	SHEET 2 OF 3


4
3
2
1
F
E
D
C
B
A
Ø4
7
16
60
57
35
10
7
1.50
4
R1
7
12.50
3.50
7
2.50
R1
65.50
1.50
3
8
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
MATERIAL
TITLE:
SOLIDWORKS Student Edition.
For Academic Use Only.
DWG NO.
bending-base
A4
WEIGHT:
SCALE:1:1
SHEET 3 OF 3

CHAPTER C

Material Spec Sheets

RUBBER-LIKE MATERIALS					
TANGOBLACKPLUS FLX980 AND TANGOPLUS FLX930					
	ASTM	UNITS	METRIC	UNITS	IMPERIAL
Tensile strength	D-412	MPa	0.8-1.5	psi	115-220
Elongation at break	D-412	%	170-220	%	170-220
Compressive set	D-395	%	4-5	%	4-5
Shore Hardness (A)	D-2240	Scale A	26-28	Scale A	26-28
Tensile Tear resistance	D-624	Kg/cm	2-4.	Lb/in	18-22
Polymerized density	ASTM D792	g/cm3	1.12-1.13		

FIGURE C.1 Sheet from Stratasys http://usglobalimages.stratasys.com/Main/Files/Material_Spec_Sheets/MSS_PJ_PJMaterialsDataSheet.pdf

CHAPTER D

Computer Simulations

All tests were done in COMSOL Multiphysics 5.2 Structural Mechanics module, using the built-in values for polydimethylsiloxane (PDMS). PDMS was chosen because it is the polymer with the closest characteristics to TangoPlus given within the COMSOL package. It is classified as a non-linear, hyperelastic material suitable for large-deformation models [45]. Additionally, PDMS is another polymer frequently used in creating flexible devices used in soft robotics [12] [6] [8], soft lithography [46], and biomimetics [47] and in other soft lithography applications. However, one of the drawbacks of using PDMS to model is that it is very hard to generically categorize, a problem it shares with TangoPlus. The material properties of PDMS are highly dependent on mixing ratios of Parts A and B, mixing quality, curing time, and curing temperature. Quake notes that, like most elastomers, PDMS have the mechanical property where the Young's modulus can vary over two orders of magnitude simply by controlling the crosslinking between the polymer chains during the fabrication process [46]. Johnson et al [1] have characterized properties of Sylgard 184, the Dow Corning elastomer most commonly used in microfluidic applications, under different crosslinking conditions.

TangoPlus is a proprietary material from Stratasys, in the PolyJet family. It is a UV curable polymer plastic formed by fused deposition modeling (FDM) [8]. The information from the material spec sheet can be found in Appendix C.

TABLE D.1 PDMS values from COMSOL.

Property	Value	Unit
Poisson's Ratio	0.49	
Young's Modulus	750	kPa
Density	970	kg/m^3

TABLE D.2 PDMS values from Johnson [1], taken at 100 °C curing temperature

Property	Value	Unit
Young's modulus	2.05 ± 0.12	MPa
Ultimate tensile strength	6.25 ± 0.84	MPa
Bulk modulus (G, tension)	0.68 ± 0.04	MPa
Bulk modulus (Ec, compression)	148.8 ± 5.47	MPa
Shear modulus	3.42 ± 0.17	GPa

The major issue with computer simulations is seen when comparing the property values in Tables D.1, D.2, and D.3. The values for Young's modulus alone differ by an order of magnitude, making these act as completely separate materials rather than one being a subselection of the other. In regards to the COMSOL values and the data reported from the Polymer Data Handbook [2], the density differs by O(3). This subset of property values shows that it is impossible to make a fully accurate model of structures made from this material without any kind of experimental data of the physical reality.

Nevertheless, we were interested in the efficiency of an incomplete computer

TABLE D.3 PDMS values from MIT 6.777/2.751J material property database [2]

Property	Value	Unit
Mass density	0.96	kg/m^3
Young's modulus	360 - 870	kPa
Poisson's ratio	0.5	
Tensile strength	2.24	MPa

simulation tests were run in COMSOL using both the uniaxial data material model and noting that PDMS is classified as a hyperelastic non-compressible material (the Poisson's ratio is listed as either 0.49 or 0.5). The simulations were run over a 0 - 1.5N range, similar to that of the experimental loading conditions and timed. For a non-linear material test, the simulation was completed in 3 hours and required some linear assumptions in the model. Simulations attempted without these assumptions failed to converge.

For comparison, 11 tests on the x-compression rig were completed within 45-60 minutes.