

5-2015

A Method for the Characterization and Selection of Compliant Joints

Ronald George Saleeby
Clemson University

Follow this and additional works at: https://tigerprints.clemson.edu/all_theses

 Part of the [Mechanical Engineering Commons](#)

Recommended Citation

Saleeby, Ronald George, "A Method for the Characterization and Selection of Compliant Joints" (2015). *All Theses*. 2157.
https://tigerprints.clemson.edu/all_theses/2157

This Thesis is brought to you for free and open access by the Theses at TigerPrints. It has been accepted for inclusion in All Theses by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.

A METHOD FOR THE CHARACTERIZATION AND
SELECTION OF COMPLIANT JOINTS

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Mechanical Engineering

by
Ronald George Saleeby
May 2015

Accepted by:
Dr. Gregory Mocko, Committee Chair
Dr. Georges Fadel, Committee Member
Dr. Gang Li, Committee Member

ABSTRACT

A compliant joint is a connection between two bodies that derives its movement from the deflection of flexible members rather than rigid connections, like traditional joints. Compliant joints have potential advantages that include longer part life, reduction of parts in assemblies, and reduced wear. Traditional compliant mechanism design methodologies have limitations involving the burden of necessary knowledge required to satisfactorily use them. The method presented in this thesis was developed to provide compliant joint design solutions independent of the traditional methods of compliant joint design by allowing the selection of compliant joints from a repository. The repository is populated by a set of twenty compliant joint models which are characterized by their geometric characteristics and parametric equations. A Finite Element Analysis (FEA) simulation is used to validate each of the individual models. The selection algorithm solves the models systematically using the design requirements set by the user. Results are presented to the user in the form of a list of compliant joints that fulfill the user requirements, and Pareto curves that represent the potential range of stiffness and deflection of compliant joints across the set of geometric characteristics in the design space. Ten test cases were applied to the selection algorithm to validate the output results.

DEDICATION

I dedicate this thesis to my wonderful wife, Mary. Without her support, love, and care, I would be lost.

ACKNOWLEDGEMENTS

I would like to thank all the people who helped me through the development and completion of this thesis.

My deepest gratitude goes to my advisor, Dr. Mocko. His knowledge, experience, guidance, and most importantly, enthusiasm, kept me engaged with my research, while pushing me to become more productive, inquisitive, and involved in the development of my thesis.

I offer my sincere appreciation to my committee members, Dr. Fadel and Dr. Li, for their support and encouragement, as well as the learning opportunities they provided me during my time at Clemson.

I would also like to acknowledge all my colleagues in the Clemson Engineering Design Applications and Research (CEDAR) lab. Their attentiveness, knowledge, and good humor were invaluable during my studies.

I would like to acknowledge Clemson University's Mechanical Engineering Department, which provided financial support and indispensable experience through my employment as an Undergraduate Lab Teaching Assistant.

Above all, I am indebted to my family, whose support and love were crucial components to my accomplishments throughout school. I acknowledge my wife, Mary, who contributed greatly to my happiness and success, at home and in the office.

TABLE OF CONTENTS

	<u>Page</u>
ABSTRACT.....	ii
DEDICATION.....	iii
ACKNOWLEDGEMENTS.....	iv
TABLE OF CONTENTS.....	v
LIST OF TABLES.....	vii
LIST OF FIGURES.....	viii
CHAPTER 1. INTRODUCTION AND MOTIVATION.....	1
1.1 Background and Importance.....	1
1.2 Current Methods for the Development and Design of Compliant Joints.....	3
1.3 Motivation for Research.....	8
1.4 Research Questions and Hypotheses.....	11
CHAPTER 2: LITERATURE REVIEW.....	13
2.1 Improvements to Methodology for the Development of Compliant Mechanisms.....	13
2.2 Design of Compliant Mechanisms for Specific Applications.....	22
2.3 Conclusions from the Literature Review.....	26
CHAPTER 3: DEVELOPMENT OF A COMPLIANT JOINT MODEL	
REPOSITORY.....	28
3.1 Adding Compliant Joints to the Repository.....	30
3.2 Example 1: Translational Compliant Joint – Smith Notch Hinge.....	34
3.3 Example 2: Rotational Compliant Joint – Jensen Cross Axis.....	39
3.4 Compliant Mechanism Repository Results.....	44
3.5 Conclusions from Compliant Joint Repository.....	52
CHAPTER 4: THE COMPLIANT JOINT SELECTION METHOD.....	53
4.1 Selecting Compliant Joints from the Repository.....	53
4.3 Validation of Test Cases.....	65
4.4 Selection Algorithm Conclusions.....	67
CHAPTER 5: CONCLUSIONS AND FUTURE WORK.....	68

TABLE OF CONTENTS (CONTINUED)

	<u>Page</u>
5.1 Conclusions.....	68
5.2 Potential Impact	69
5.3 Future Work.....	70
CHAPTER 6: REFERENCES	72
APPENDICES	75
APPENDIX A. COMPLIANT JOINT REPOSITORY	76
A.1 Translational Compliant Joints	76
A.2 Rotational Compliant Joints.....	90
APPENDIX B. COMPLETE SELECTION ALGORITHM MATLAB CODE	125
APPENDIX C. MATLAB GUI CODE	163
APPENDIX D. MATLAB PARETO CURVE GENERATING CODE	170
APPENDIX E. TEST CASES USED TO VALIDATE SELECTION	
ALGORITHM.....	178
E.1 Test Case 1	178
E.2 Test Case 2	181
E.3 Test Case 3	184
E.4 Test Case 4	187
E.5 Test Case 5	192
E.6 Test Case 6	195
E.7 Test Case 7	198
E.8 Test Case 8	201
APPENDIX F. ADDITIONAL APPROACHES CONSIDERED	204

LIST OF TABLES

<u>Table</u>	<u>Page</u>
Table 1. Research questions and research hypotheses	12
Table 2. Geometric properties of the Smith Notch Hinge compliant joint used for comparison between the MATLAB model and the FEA model.....	37
Table 3. Geometric properties of the Jensen Cross Axis compliant joint used for comparison between the MATLAB model and the FEA model.....	43
Table 4. Translational Compliant Joint Repository List	46
Table 5. Rotational Compliant Joint Repository List	47
Table 6. A comparison of FEA models and parametric equations for translational compliant joints for a unit force of 1 N.....	50
Table 7. A comparison of FEA models and parametric equations for rotational compliant joints for a unit force of 1 N.....	51
Table 8. Inputs for the test cases.	60
Table 9. Test Cases, with the True Positives, True Negatives, False Positives, and False Negatives determined for each.	66
Table A 1. Input values for Test Case 1.....	178
Table A 2. Input values for Test Case 2.....	181
Table A 3. Input values for Test Case 3.....	184
Table A 4. Input values for Test Case 4.....	187
Table A 5. Input values for Test Case 5.....	192
Table A 6. Input values for Test Case 6.....	195
Table A 7. Input values for Test Case 7.....	198
Table A 8. Input values for Test Case 8.....	201

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
Figure 1: A traditional joint pair of locking pliers and a compliant joint pair of locking pliers [1].	1
Figure 2. (a) Psuedo-rigid body model comparison for a flexure hinge, and (b) pseudo-rigid-body model comparison for a buckling compliant beam [2].....	3
Figure 3. Example of a size optimization design space [1].	4
Figure 4. An example of shape optimization: (a) is the design space used for the problem, and (b) through (d) are the potential shape topologies that connect the input and output ports. Note: These are not necessarily the solutions for this problem, just examples that illustrate possibilities [1].....	5
Figure 5. (a) Discretization of a design space for continuous material density topology optimization. (b) Possible topology solution with black (solid), white (void) and grey (intermediate density). (c) Possible interpretation of the topology solution. Note: This is not necessarily the solution for this problem, just an example that illustrates a possibility.	7
Figure 6. Overview of compliant joint selection method.	11
Figure 7. (a) A simple fixed pin segment, and (b) its pseudo-rigid-body model [7].....	14
Figure 8. (a) Design domain for compliant hand tool (b) optimal topologies for different fixed handle sizes (c) constructed hand tool from interpreted topology optimization design [15].....	18
Figure 9. (a) Compliant straight line mechanism (deformed and undeformed) created by pseudo-rigid-body design and (b) the same mechanism (deformed and undeformed) created by optimal synthesis with continuum model [17].....	19
Figure 10. A compliant translational joint and its characterization as a compliant building block [18].....	20
Figure 11. An example of a displacement amplifying compliant mechanism defined in Jhawar’s catalogue [19].	21

LIST OF FIGURES (CONTINUED)

<u>Figure</u>	<u>Page</u>
Figure 12. Prototype design of Mahler’s pediatric prosthetic knee [20].	22
Figure 13. Schematic of an isolation joint system [25].	25
Figure 14. (a) Direct compression and (b) application of the Inversion theory [25].	26
Figure 15. Defined movement of rotational compliant joints.	29
Figure 16. Defined movement of rotational compliant joints.	30
Figure 17. Outline of process used to develop models of compliant joints.	30
Figure 18. Geometric characteristics of the Smith Notch Hinge compliant joint.	35
Figure 19. The initial MATLAB file used to test the Smith Notch Hinge compliant joint model, <code>SmithNotchHinge.m</code>	36
Figure 20. The completed Smith Notch Hinge model, as a MATLAB function.	37
Figure 21. Boundary conditions used for the Smith Notch Hinge compliant joint model simulations.	38
Figure 22. Mesh and deformed shape for the Smith Notch Hinge FEA simulation.	39
Figure 23. Geometric characteristics of the Jensen Cross Axis compliant joint.	40
Figure 24. The initial MATLAB file used to test the Jensen Cross Axis compliant joint model, <code>JensenCrossAxis.m</code>	41
Figure 25. The completed Jensen Cross Axis model, as a MATLAB function.	42
Figure 26. Boundary conditions used for the Jensen Cross Axis compliant joint model simulations.	43
Figure 27. Mesh and deformed shape for the Jensen Cross Axis FEA simulation.	44
Figure 28. Overview of selection algorithm for determining compliant joints that fulfill the user’s requirements.	53
Figure 29. Graphical user interface for selection algorithm.	54
Figure 30. Example Pareto chart outputs from the algorithm.	57
Figure 31. Example output from the selection algorithm.	58
Figure 32. GUI input for Test Case 1.	61

LIST OF FIGURES (CONTINUED)

<u>Figure</u>	<u>Page</u>
Figure 33. Pareto output for Test Case 1.	62
Figure 34. MATLAB text output for Test Case 2.....	62
Figure 35. GUI input for Test Case 2.	63
Figure 36. Pareto curves for Test Case 2, using an applied force of 10 N.	64
Figure 37. MATLAB text output for Test Case 2.....	65
Figure A 1. Solid model representation of the Smith Rectilinear compliant joint.	76
Figure A 2. Geometric characteristics of the Smith Rectilinear compliant joint.....	77
Figure A 3. MATLAB function model of the Smith Rectilinear compliant joint.	77
Figure A 4. Solid model representation of the Kyusojin Parallel Strip compliant joint.	78
Figure A 5 Geometric characteristics of the Kyusojin Parallel Strip compliant joint.	78
Figure A 6. MATLAB function model of the Kyusojin Parallel Strip compliant joint.	79
Figure A 7. Solid model representation of the Kyusojin Linear 6L1 compliant joint, (a) original position and (b) deflected position.	80
Figure A 8. Side view of solid model representation of the Kyusojin Linear 6L1 compliant joint, (a) original position and (b) deflected position.	80
Figure A 9. Geometric characteristics of the Kyusojin Linear 6L1 compliant joint.	81
Figure A 10. MATLAB function model of the Kyusojin Linear 6L1 compliant joint.	82
Figure A 11. Solid model representation of the Trease Translational compliant joint.	83
Figure A 12. Geometric characteristics of the Trease Translational compliant joint.	83
Figure A 13. MATLAB function model of the Trease Translational compliant joint.	84

Figure A 14. Solid model representation of the Xu Translational compliant joint.....	85
Figure A 15. Geometric characteristics of the Xu Translational compliant joint.	85
Figure A 16. MATLAB function model of the Xu Translational compliant joint.	87
Figure A 17. Solid model representation of the Smith Notch Hinge compliant joint.	88
Figure A 18. Geometric characteristics for the Smith Notch Hinge compliant joint.	88
Figure A 19. MATLAB function model of the Smith Notch Hinge compliant joint.	89
Figure A 20. Solid model representation of the Lobontiu Symmetric Notch compliant joint.	90
Figure A 21. Geometric characteristics of the Lobontiu Symmetric Notch compliant joint.	90
Figure A 22. MATLAB function model of the Lobontiu Symmetric Notch compliant joint.	91
Figure A 23. Solid model representation of the Lobontiu Corner Filleted compliant joint.	92
Figure A 24. Geometric characteristics of the Lobontiu Corner Filleted compliant joint.	92
Figure A 25. MATLAB function model of the Lobontiu Corner Filleted compliant joint.	94
Figure A 26. Solid model representation of the Lobontiu Symmetric Circle compliant joint.	95
Figure A 27. Geometric characteristics of the Lobontiu Symmetric Circle compliant joint.	95
Figure A 28. MATLAB function model of the Lobontiu Symmetric Circle compliant joint.	97
Figure A 29. Solid model representation of the Tian V Shape Flexure compliant joint.	98

Figure A 30. Geometric characteristics of the Tian V Shape Flexure compliant joint.	98
Figure A 31. MATLAB function model of the Tian V Shape Flexure compliant joint.	100
Figure A 32. Solid model representation of the Tang Symmetric Circular compliant joint.	101
Figure A 33. Geometric characteristics of the Tang Symmetric Circular compliant joint.	101
Figure A 34. MATLAB function model of the Tang Symmetric Circular compliant joint.	102
Figure A 35. Solid model representation of the Smith Two Axis compliant joint.	103
Figure A 36. Geometric characteristics of the Smith Two Axis compliant joint.	103
Figure A 37. MATLAB function model of the Smith Two Axis compliant joint.	104
Figure A 38. Solid model representation of the Smith Annulus compliant joint.	105
Figure A 39. Geometric characteristics of the Smith Annulus compliant joint.	105
Figure A 40. MATLAB function model of the Smith Annulus compliant joint.	107
Figure A 41. Solid model representation of the Smith Cartwheel compliant joint.	108
Figure A 42. Geometric characteristics of the Smith Cartwheel compliant joint.	108
Figure A 43. MATLAB function model of the Smith Cartwheel compliant joint.	109
Figure A 44. Solid model representation of the Smith Cruciform compliant joint.	110
Figure A 45. Geometric characteristics of the Smith Cruciform compliant joint.	110
Figure A 46. MATLAB function model of the Smith Cruciform compliant joint.	111
Figure A 47. Solid model representation of the Jensen Cross Axis compliant joint.	112
Figure A 48. Geometric characteristics of the Jensen Cross Axis compliant joint.	112
Figure A 49. MATLAB function model of the Jensen Cross Axis compliant joint.	113
Figure A 50. Solid model representation of the Smith Rotationally Symmetric Leaf Hinge compliant joint.	114
Figure A 51. Geometric characteristics of the Smith Rotationally Symmetric Leaf Hinge compliant joint.	114

Figure A 52. MATLAB function model of the Smith Rotationally Symmetric Leaf Hinge compliant joint.	116
Figure A 53. Solid model representation of the Trease Rotational compliant joint.	117
Figure A 54. Geometric characteristics of the Trease Rotational compliant joint.....	117
Figure A 55. MATLAB function model of the Trease Rotational compliant joint.	118
Figure A 56. Solid model representation of the Kyusojin Rotational 6R2 compliant joint.	119
Figure A 57. Geometric characteristics of the Kyusojin Rotational 6R2 compliant joint.	119
Figure A 58. MATLAB function model of the Kyusojin Rotational 6R2 compliant joint.	121
Figure A 59. Solid model representation of the Goldfarb Conventional Split Tube compliant joint.	122
Figure A 60. Geometric characteristics of the Goldfarb Conventional Split Tube compliant joint.	123
Figure A 61. MATLAB function model of the Goldfarb Conventional Split Tube compliant joint.	124
Figure A 62. Complete MATLAB function selection algorithm.....	162
Figure A 63. The MATLAB function that controls the GUI.....	169
Figure A 64. Complete MATLAB function used to build Pareto curves for a translational joint set.	172
Figure A 65. Complete MATLAB function used to build Pareto curves for a rotational joint set.	177
Figure A 66. GUI inputs used for Test Case 1.....	179
Figure A 67. Pareto curve outputs for Test Case 1.....	180
Figure A 68. MATLAB text output for Test Case 1.....	180
Figure A 69. GUI inputs used for Test Case 2.....	182
Figure A 70. Pareto curve outputs for Test Case 2.....	183
Figure A 71. MATLAB text output for Test Case 2.....	183

Figure A 72. GUI inputs used for Test Case 3.....	185
Figure A 73. Pareto curve outputs for Test Case 3.....	186
Figure A 74. MATLAB text output for Test Case 3.....	186
Figure A 75. GUI inputs used for Test Case 4.....	188
Figure A 76. Pareto curve outputs for Test Case 4, using an applied force of 0.1 N.....	189
Figure A 77. Pareto curve outputs for Test Case 4, using an applied force of 1 N.	189
Figure A 78. Pareto curve outputs for Test Case 4, using an applied force of 10 N.	190
Figure A 79. MATLAB text output for Test Case 4.....	191
Figure A 80. GUI inputs used for Test Case 5.....	193
Figure A 81. Pareto curve outputs for Test Case 5.....	194
Figure A 82. MATLAB text output for Test Case 5.....	194
Figure A 83. GUI inputs used for Test Case 6.....	196
Figure A 84. Pareto curve outputs for Test Case 6, using an applied force of 0.1 N.....	196
Figure A 85. Pareto curve outputs for Test Case 6, using an applied force of 1 N.	197
Figure A 86. Pareto curve outputs for Test Case 6, using an applied force of 10 N.	197
Figure A 87. MATLAB text output for Test Case 6.....	197
Figure A 88. GUI inputs used for Test Case 7.....	199
Figure A 89. Pareto curve outputs for Test Case 7.....	200
Figure A 90. MATLAB text output for Test Case 7.....	200
Figure A 91. GUI inputs used for Test Case 8.....	202
Figure A 92. MATLAB text outputs for Test Case 8.....	203

CHAPTER 1. INTRODUCTION AND MOTIVATION

The primary objective of this research is the development of a method of characterizing compliant joints that enables their selection from a repository. The repository will be comprised of compliant joints of different characteristics and types, which can be defined geometrically by certain parameters and by parametric equations. An algorithm will be used to select compliant joints from the repository based on user requirements.

1.1 Background and Importance

A connection between two bodies that imposes constraints on their relative movement is called a joint. Compliant joints are a type of joint which derive their motion from the deflection of flexible members rather than rigid connections, like traditional mechanical joints. A compliant mechanism is a type of mechanism that uses a combination of compliant joints to achieve a desired motion [1]. A comparison of a traditional joint pair of locking pliers and a compliant pair of locking pliers is shown in Figure 1.

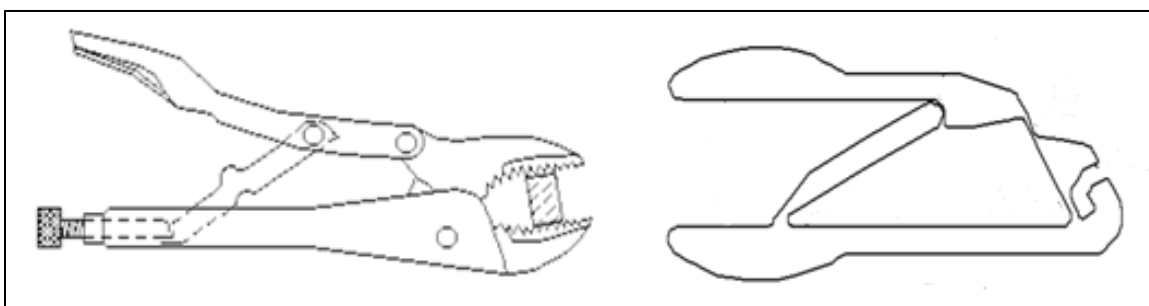


Figure 1: A traditional joint pair of locking pliers and a compliant joint pair of locking pliers [1].

Certain advantages are inherent to this type of construction. Compliant joints are typically produced as a single part, which results in a reduction in total parts needed for assembly. This can attribute to a reduction in assembly time in mass production. Since compliant joints rely on deflection rather than mobile interconnections, they typically will not be as affected by contaminants or abrasives, and therefore can experience reduced wear and less need for lubrication. This can result in a longer part life. Disadvantages of compliant joints include reduced range of motion compared to traditional joints, since the motion is derived from deflection. They have from reduced absolute strength, since the deflection of thin members generally cannot handle large loads. It is also possible for compliant joints to experience degradation of bending characteristics over time, due to the loss of resilience of polymers over extended use [1, 2, 3].

A goal of this research is to utilize the advantages of compliant joints by increasing accessibility to the end-user. Designers, fabricators, and inventors do not typically have the background or experience to design complex compliant joints, and may not be aware of the capabilities of them. If a user inexperienced in compliant joint design can reach a satisfactory design without the knowledge required for traditional compliant joint design methodology, then it is possible for them to have the opportunity to incorporate it into their designs. A designer could set requirements, select, and rapid prototype a compliant joint in a matter of hours. This is further bolstered by the ease at which compliant joints are made from thermoplastics and other common rapid prototyping materials. To facilitate this, a methodology that simplifies the selection of compliant joints and presents them in an ordered fashion to a user is necessary. Fields of

production have not traditionally had access to the software and design theory behind the creation of new compliant joints, or the use of them to satisfying standing criteria, and thus they have suffered a lack of use in mass production.

1.2 Current Methods for the Development and Design of Compliant Joints

Two main methods exist for the design of new compliant joints and the implementation of compliant joints into various applications. The first is the pseudo-rigid-body model approach, also known as the kinematics approach. This method involves drawing the compliant sections of a mechanism as rigid links and calculating the required spring constant of those sections. This technique is most effective to represent the movement of joints that undergo large deflections or have complex movement. The disadvantage of this method is that it requires known rigid-body comparisons for all compliant sections that will be used [2]. An example of how this method is applied to represent a leaf spring flexure and a buckling beam is shown in Figure 2.

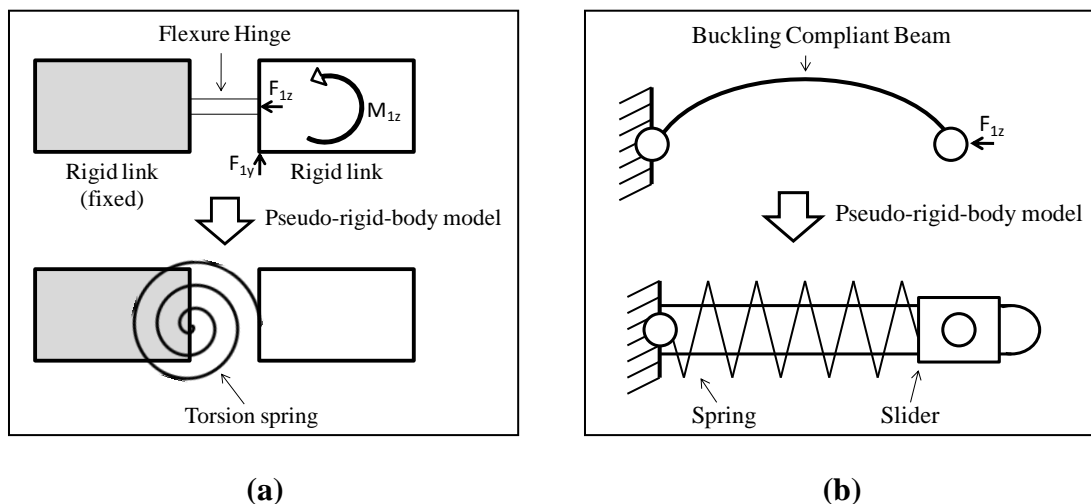


Figure 2. (a) Pseudo-rigid body model comparison for a flexure hinge, and (b) pseudo-rigid-body model comparison for a buckling compliant beam [3].

The second method of compliant mechanism design is the structural optimization approach. This method uses a design domain that is divided into discretized elements in a mesh. The locations of the loads, supports, and desired characteristics of the joint are modeled on nodes within this mesh, and then various numerical techniques can be solved iteratively to determine the final characteristics of the compliant joint. Howell [1] divides structural optimization for compliant mechanisms into three levels of hierarchy.

1. Size optimization is the simplest type of structural optimization. It can be used to determine the cross section profile and thickness dimensions of beam and truss elements, thickness of plate and shell elements, size of a holes and similar design variables. Figure 3 is an example of the design space for an application of size optimization. In this example, the thickness profile $w(x)$ of a simply supported beam of length L is the design variable. The beam is subjected to the constraints of a certain transverse deflection at distance a , and a certain load across the beam, $p(x)$. The objective is the minimization of the volume of the beam.

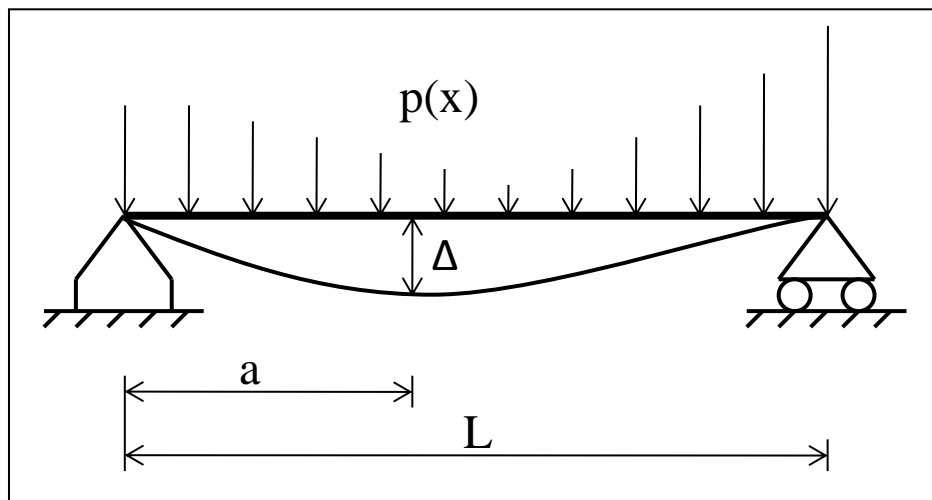


Figure 3. Example of a size optimization design space [1].

2. Shape optimization is generally more difficult and computationally intensive than size optimization. The design space is a set of all potential shapes. The topology of these shapes can be specified through different types of design variables. An example of these design variables could be the control points on a Bezier or spline interpolation curve, which would determine the overall shape of the design. The shape changes with each iteration, which can lead to the accuracy of the finite element model diminishing unless directed to remesh. The designer must be aware of the effects of the shape design variables on the objective and constraint functions by performing a sensitivity analysis. An example of various shape optimizations for a given design space is shown in Figure 4.

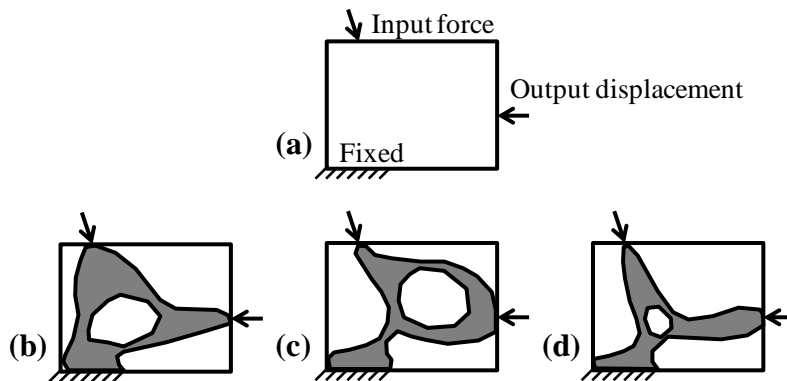
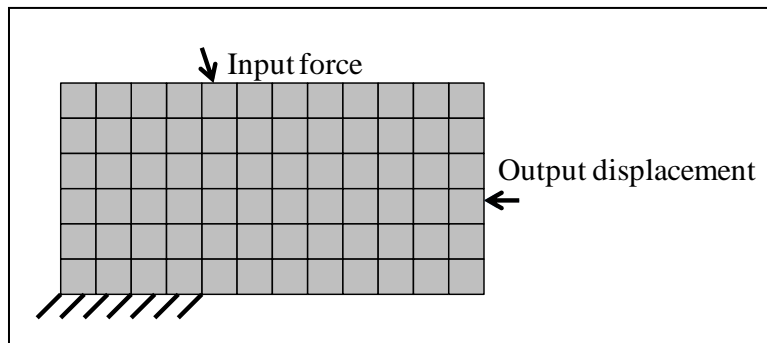


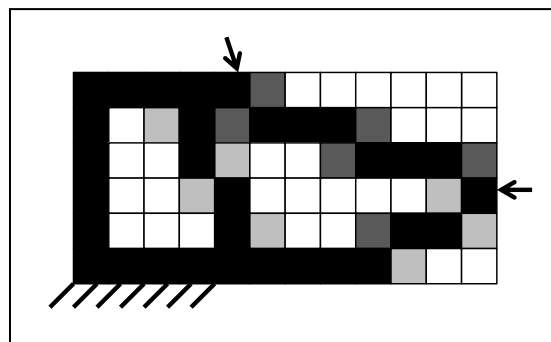
Figure 4. An example of shape optimization: (a) is the design space used for the problem, and (b) through (d) are the potential shape topologies that connect the input and output ports. Note: These are not necessarily the solutions for this problem, just examples that illustrate possibilities [1].

3. Topology optimization is considered the most general type of optimization. This method determines the location of material across the entire design domain, like the connectivity between input ports, output ports, and fixed locations. This is

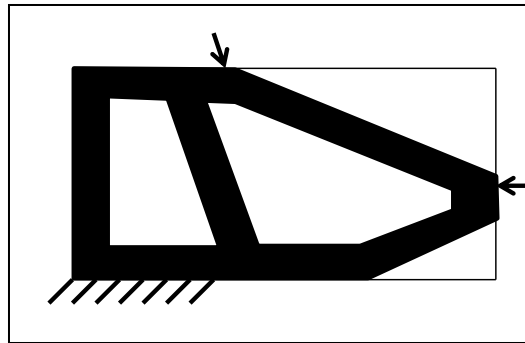
useful because a designer does not have to commit to a certain shape like size optimization, or a set of shapes defined by a mathematical relation, like shape optimizations. Topology optimization utilizes a mesh that discretizes the entire design space, and then determines the material density that will be distributed in each element of the mesh. The form of the resultant design is derived from the final presented mesh, but depending on the fidelity of the mesh, some post-processing to the design may have to be completed before it can be finalized. Figure 5 shows a topology optimization of a design space with a set input force, output displacement, and fixed region. The density of each cell is a design variable ranging from void to solid [4].



(a)



(b)



(c)

Figure 5. (a) Discretization of a design space for continuous material density topology optimization. (b) Possible topology solution with black (solid), white (void) and grey (intermediate density). (c) Possible interpretation of the topology solution.

Note: This is not necessarily the solution for this problem, just an example that illustrates a possibility.

Topology optimization considers the widest variety of potential topologies for a design space. This makes it a useful technique to be used in the design of new compliant mechanisms. The designer does not have to have an underlying background or comprehension of the various building blocks of compliant mechanisms to design a particular topology. Instead, the optimization algorithm determines the shape and size simultaneously to satisfy the design requirements. The weakness of this technique is that the success of a topology optimization is entirely dependent on the user's initial setting of the design domain. There is a possibility that an initial guess can result in no potential possibilities due to local minima or maxima. To avoid this, a sensitivity analysis should be performed iteratively with the analysis to determine the effects of changes in the design variables to the objective and constraint functions [1].

The pseudo-rigid-body method requires that the user know and understand the underlying compliant joint to rigid-body comparisons to be used as a design method. Similarly, the structural optimization method requires the user to understand how to set up a design space such that potential solutions are possible, while minimizing iteration time. The high burden of knowledge required to use these methods translates to user difficulty when attempting to design compliant joints.

1.3 Motivation for Research

The properties of compliant joints have resulted in many diverse research applications being developed, ranging from microscopic compliant joints for precision movement in Microelectromechanical systems (MEMS) to macroscopic compliant joints that provide structural support in lieu of beams. A gap analysis was performed to identify potential research gaps and the research applications that have not been fully explored. The following are several of the identified research gaps.

1. **Compliant joint designs have not been developed that are optimized to take advantage of the parasitic nature of axial drift to achieve a desired motion.**

Research has been done previously on the minimization of axial drift to increase the precision of compliant joints, especially on a micro-scale level. The use of axial drift to develop slider-crank type motion joints has not been well established [5].

2. **The application of newly developed compliant mechanisms in the macroscale (hand-size) as a replacement for traditional joints has not been adequately explored.** Much of compliant mechanism research focuses on the application of

compliant mechanisms in the micro- and nano-scales, due to their inherent advantages over traditional joints (like lack of required lubrication), which make them ideal in this environment. Development of compliant mechanisms for use in the macro-design industries could increase the volume of compliant mechanisms in the mainstream industry [2]. An illustration of this potential can be seen with the compliant locking pliers in Chapter 1.

3. **To design and develop new compliant mechanisms for an application, the pseudo-rigid-body method, topology optimization method, a derivative of these two methods, or trial and error must be used.** Current compliant mechanism design methodologies are constantly improving, but some limitations of these methods have not been addressed. Parallel methodologies and alternatives to these methods are scarce. If a pseudo-rigid-body model has been constructed, but transformations into compliant structures do not exist, it cannot be converted into a compliant mechanism. Topology optimization can fail if the user does not have an understanding of the design domain, and how it affects the final result. [1, 4, 7].
4. **Current compliant mechanism design methodologies have a steep learning curve, and the accompanying software is difficult to use.** The previously outlined methods are the most widely used techniques for developing new compliant mechanisms. An understanding of pseudo-rigid-body to compliant joint comparisons is required for the pseudo-rigid-body method to be feasible. Alternatively, the learning curve for developing a design space that will produce

feasible solutions in topology optimization software requires that the user understand the design space and how its implementation will affect the final result. Both of these methods of selection and design of compliant mechanisms are only as powerful as the user developing the problem [1, 4, 6].

5. The focus of a large amount of compliant mechanism research is on a single or limited number of applications, rather than a large range of applications.

A section of compliant mechanism research and development has been the creation of new compliant joints for specified applications. Compliant joints can be applied to a vast number of applications, but limited research has been done in this arena. A methodology that can be applied to the widest reaching amount of compliant mechanism design would bridge this gap [4, 8, 9].

This research primarily focuses on the gaps of user accessibility and the broad reach of potential applications. The overall goal is the development of a method that allows a user to select a compliant joint that has the capability to fulfill their design requirements. The approach used to bridge these gaps is as follows. A repository of compliant joint models will be created to represent different compliant joint possibilities. Users will be able to narrow the selection of compliant joints by filling in some requirements that are desired. The selection algorithm will not be limited by user input, and will still be able to successfully present potential solutions to the user even with missing requirements. The algorithm can determine a range of values for the missing parameters and provide the user with a list of compliant joints that most closely fulfill their requirements. A brief overview of this approach is shown in Figure 6.

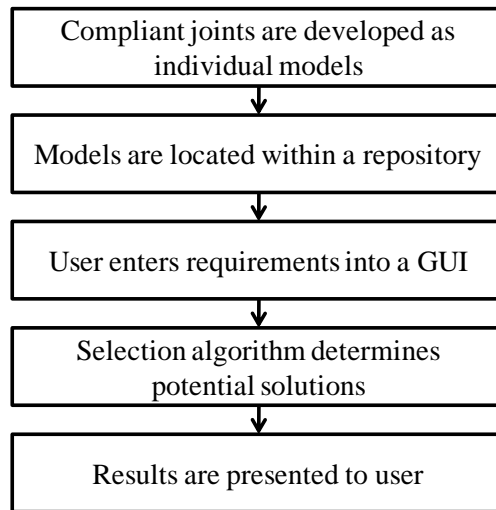


Figure 6. Overview of compliant joint selection method.

1.4 Research Questions and Hypotheses

Research questions and accompanying hypotheses were developed to aid in the clarification of goals for this research. These research questions and support of the hypotheses will result in satisfactory fulfillment of the research. In Table 1, they are divided into the research questions, hypotheses, and the potential solution for fulfillment.

Table 1. Research questions and research hypotheses

Research Questions	Research Hypotheses	Fulfillment
What parameters can a compliant joint be characterized by such that it can be objectively compared to other compliant joints?	A compliant joint can be characterized by geometry and parametric equations such that it can be compared by objective metrics to other compliant joints.	Compliant joints can be compared to one another based on normalized characteristics, to facilitate selection.
How can a user achieve results that include a compliant joint that most closely fulfills their requirements?	A compliant joint can be consistently selected from a repository, to fulfill a user's requirements.	A correct compliant joint can be determined and presented to the user, after the selection process has been completed.
How can the information be presented to the user so that it is possible to differentiate between multiple satisfactory solutions?	Using the characteristics specific to a compliant joint, it is possible to present information that assists in the selection of the most applicable compliant joint from a set of two or more.	The research provides the tools for a list of compliant joints to be differentiated objectively by their characteristics, to facilitate final selection.

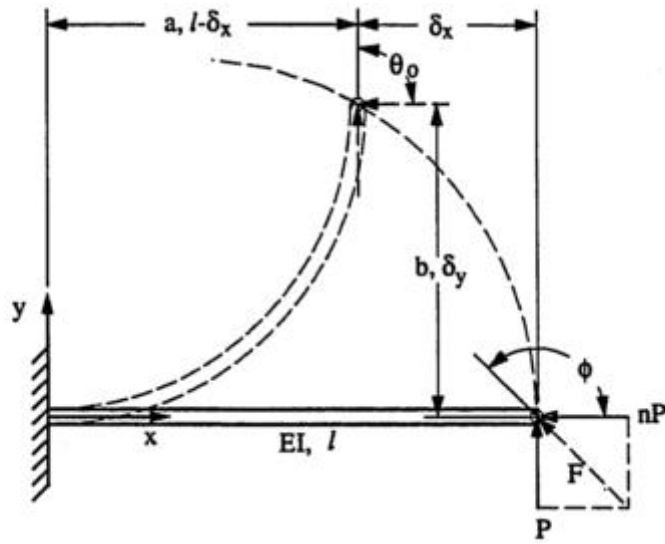
For the compliant joint repository and selection method to be considered successful, it will need to accomplish the goals outlined here and thus prove its value as a compliant joint selection methodology. This includes facilitating characterization and normalization of a number of compliant joints, allowing them to be compared to one another objectively, providing a means of selection between these models such that they fulfill user requirements, and providing results that allow a user to determine the “best” compliant joint from a list which contains all satisfactory compliant joints.

CHAPTER 2: LITERATURE REVIEW

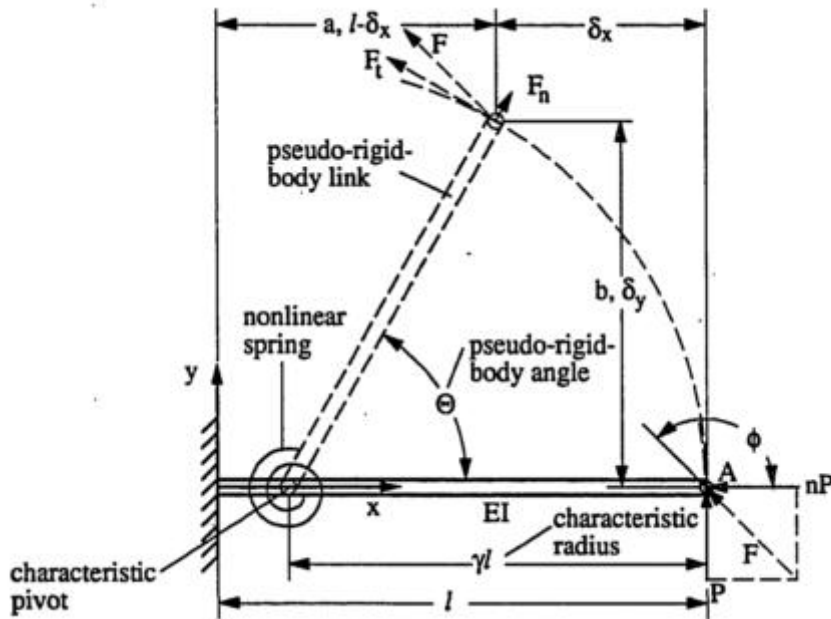
To create the frame of reference for this thesis, a literature review had to be performed on the following topics: (1) the improvement of currently existing methods for the design and synthesis of compliant joints, and (2) the design and implementation of new compliant joints. This includes a study of some of the compliant joints that have been designed for a variety of specific applications. These two topics are common themes within compliant mechanism research, driving new innovations and improvements.

2.1 Improvements to Methodology for the Development of Compliant Mechanisms

Pseudo-rigid-body models of compliant mechanisms were developed to represent the design requirements of potential applications. A designer using this method can take advantage of well documented rigid-body mechanisms to approximate the movement of a compliant mechanism. This allows large nonlinear deflections of compliant flexures to be modeled as rigid links attached at pin joints, that have equivalent force-deflection characteristics [1]. Howell and Midha [6] developed some of the first techniques for the pseudo-rigid-body model approach when they replaced flexural pivots with kinematic joints at the center of a flexible segment, using a torsion spring to represent the stiffness of the joint. Prior to the development of this method, the design and development of new compliant mechanisms was primarily trial and error. An example of the pseudo-rigid-body model approach models that Howell developed can be seen in Figure 7.



(a)



(b)

Figure 7. (a) A simple fixed pin segment, and (b) its pseudo-rigid-body model [6].

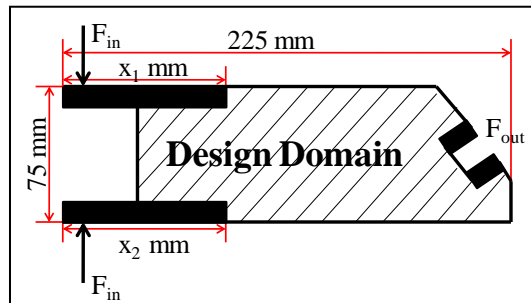
As the need to represent more complex compliant mechanisms for a variety of applications increased, more approximations and representations were required to ensure

accuracy of a pseudo-rigid-body model. Howell [7] offered an improved model of large deflection beams under end load, which was able to approximate the nonlinear path of a deflected cantilever beam within 0.5 percent of the closed-form elliptic integral solutions. Many other researchers have contributed additional improvements to pseudo-rigid-body models catalogue. Edwards [8] produced a concept capable of simulating pinned-pinned connections as two individual rigid members connected at pin joints, with a torsion spring at the pin joint representing the flexible member's stiffness. The accuracy of this model was tested analytically using fabricated aluminum, steel, and polypropylene segments. It was shown that the model accurately predicted the segment's deflection characteristics. Additional work in the field of pseudo-rigid-body models has been completed by Espinosa [9] for the purpose of improving the modeling of straight and curved flexures subject to compressive loads. He included a new parameter called the characteristic radius factor, which is a function of the moment of the beam, to describe the motion of the deflected beam over a large range of motion. This was developed for use in the design of ortho-planar micro-electromechanical systems (MEMS), which can achieve motion out of the plane of fabrication. Tang and colleagues [10] studied a method of modeling large displacement precision positioning flexures, by decoupling the kinematic structure and comparing it to a pseudo-rigid-body model. This work was verified through the design of a large displacement prismatic joint that achieved very precise linear movements with minimized parasitic rotation (less than 1.5 mrad). They also showed that their method was more accurate than a traditional pseudo-rigid-body model.

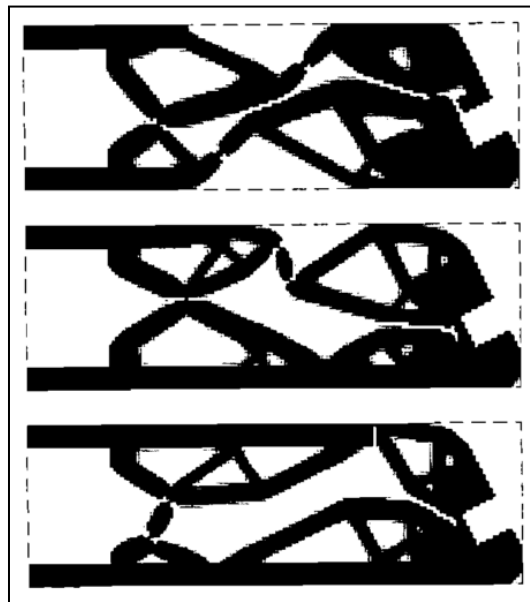
Topology optimization methodology for the development and design of compliant mechanisms has undergone various improvements as well. Topological synthesis for compliant joints, which is the development of new compliant joints from a series of flexures, allows flexures to be combined in the most optimal way to achieve a desired output. Frecker and colleagues [11] contributed to the methodology of topological synthesis with a method that utilized multi criteria optimization. This was developed to combine the conflicting design objectives of flexibility and stiffness required for a specific deflection. The functionality of the solutions was proved through both finite element models and prototype designs. Frecker and her coauthors [12] also developed two methods of topology optimization for compliant mechanisms with multiple outputs, both of which used a ratio of potential energy to strain energy as a design objective to produce compliant joints that have a specified stiffness and flexibility. The methodologies that were produced were named *combined virtual load* and *weighted sum of objectives*. Of the two, combined virtual load used significantly less computational time. Another topology synthesis development that was function driven was Lu's work [13] in the synthesis of compliant mechanisms to achieve a specific curve, while simultaneously using a minimum number of actuators. This was applied to topologies that require an adaptive shape change, like antenna reflectors, which can potentially increase system performance and flexibility through changing the signal pattern or coverage area of the reflectors.

Continuum topology optimization of compliant mechanisms, which uses a set design domain and specified positions and directions of all input and outputs, has been

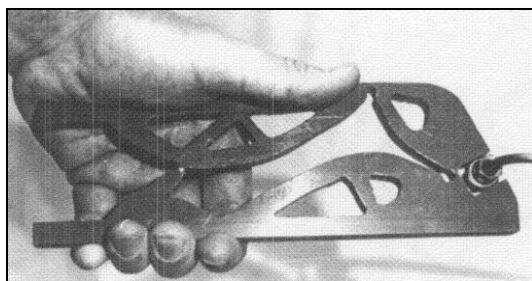
improved upon by the development of certain techniques. Sigmund [14] used a technique that controlled the maximum stress level in a compliant mechanism by constraining the allowed displacement at the input port. Another technique he used involved utilizing a design constraint of reduced internal stresses by limiting allowed displacement output. This could be used to design compliant mechanisms with specific complex behaviors. These methods allowed him to design and develop a compliant hand tool which maximized the mechanical advantage between the input and output ports. The process and design domain for this compliant hand tool is detailed in Figure 8.



(a)



(b)



(c)

Figure 8. (a) Design domain for compliant hand tool (b) optimal topologies for different fixed handle sizes (c) constructed hand tool from interpreted topology optimization design [14].

Pedersen [15] further improved continuum topology optimization methods in large displacement compliant mechanisms through a Finite Element Analysis (FEA) method that utilized non-linear analysis. He demonstrated that the gain in output performance can be as high as 2.5 when comparing non-linear analysis to linear analysis, although the computational cost increases substantially. These techniques were used to demonstrate that topology optimization can be used for the generation of complex path-generating compliant mechanisms.

A deterministic approach was tested by Pavlović [16] which developed two rectilinear compliant mechanisms that each had a specific output deflection. A direct comparison was made between a compliant mechanism created using the pseudo-rigid-body method and a compliant mechanism created using optimal synthesis with continuum models. He found that the resultant compliant mechanism from these methods had different optimized characteristics depending on the method used. The rigid-body-model approach of the coupler points showed greater guiding accuracy than the optimal synthesis with continuum model. The greater overall stiffness of the compliant

mechanism can be obtained through the optimal synthesis with continuum model, and as a result of this, this model has a higher velocity of guiding point, higher output force as well as higher mechanical advantage. The two compliant straight-line mechanisms that were designed and compared are shown in Figure 9.

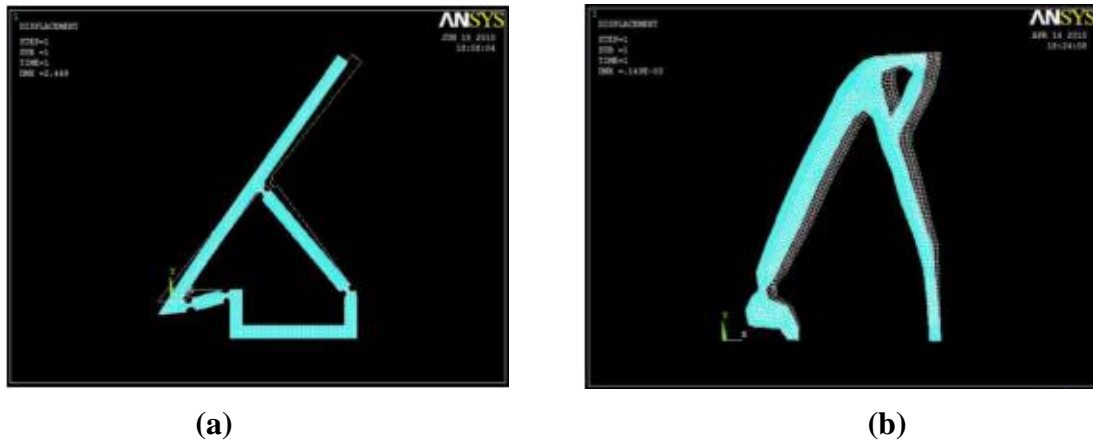


Figure 9. (a) Compliant straight line mechanism (deformed and undeformed) created by pseudo-rigid-body design and (b) the same mechanism (deformed and undeformed) created by optimal synthesis with continuum model [16].

Alternatives to the previously highlighted methods of compliant mechanism design and selection have been posed as well. Bernardoni [17] proposed a method which considered compliant mechanisms an assembly composed of compliant “building blocks” which are modeled by elementary frame ground structures. The method characterizes the structural parameters of the blocks between flexures by height, width, and thickness, as well as the Young’s modulus, Poisson ratio, and density. The whole structure is complete as a “square block,” acting as a compliant sliding joint characterized by defined stiffness matrices that link the nodes. Figure 10 presents a translational compliant joint as realized as a compliant building block.

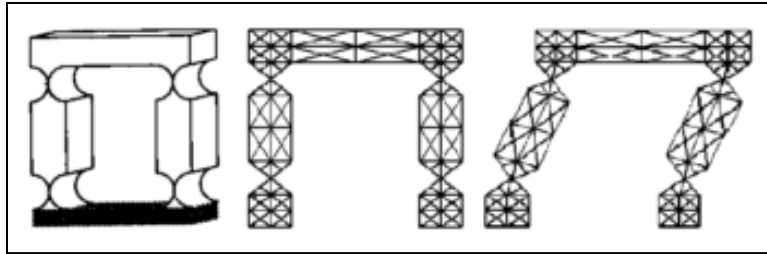


Figure 10. A compliant translational joint and its characterization as a compliant building block [17].

These building blocks are then used in a topological optimization. The user must define the discrete topology and boundary-condition variables that will define the desired compliant joint. The discrete topology variables that the user must define are the type of blocks, size of blocks, material, and thickness. The user must also define the boundary-conditions that are steady for all outputs. These are the minimum and maximum number of fixed nodes, locations of these fixed nodes, the minimum and maximum number of contacts, the allowed location of contacts, and the backlash of the contacts. The boundary-conditions that change for each output, which include the minimum and maximum number of actuators, the allowed location of actuators, the forces provided by the actuators and the maximum strokes, and the stiffness of each actuator, must also be defined. The topology, dimensions, material, contacts, fixed frame, and actuators are generated using a multi objective genetic algorithm (based on NSGA2) with the design object of achieving a maximized force/motion ratio. A trial and error procedure must be performed by the user to determine if the optimization has found errors, like buckling phenomenon or Von Mises stresses that exceed the material limits. This method was tested to create a two degree of freedom compliant mechanism, and references the

procedure was completed in a short computational time, although that time is not listed [17].

A catalogue and selection method was put forth by Jhwar [18] to select Displacement amplifying Compliant Mechanisms (DaCM) for a given application. This method uses a catalogue of defined compliant mechanisms to select the most suitable mechanism for the quantitative specifications of the user. This required data includes force, displacement and stiffness specifications at the input and output. The DaCM catalogue consists of slender beams connected by flexures configured in specific topologies. These are analyzed using finite element analysis and the mechanism's springs-mass lever (SML) model parameters are extracted and stored parametrically as functions of the size of the device. These pre-computed SML models are used to select a suitable DaCM of appropriate user-specified size. The final catalogue presented with eight models that contain different topologies found in current literature, although more could be completed with topology optimization. Parallels could be drawn between this research's approach to a selection algorithm and the research presented in this thesis. An example of a DaCM model that is used in the catalogue is shown in Figure 11.

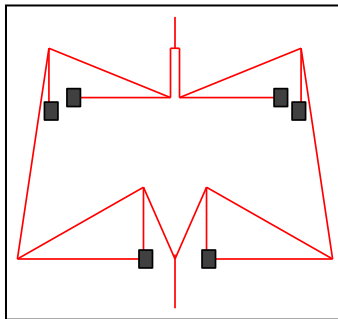


Figure 11. An example of a displacement amplifying compliant mechanism defined in Jhwar's catalogue [18].

2.2 Design of Compliant Mechanisms for Specific Applications

Efforts in compliant joint research have also been geared towards the development of new compliant mechanisms for specific applications. The precision, lack of friction, reduced wear, and absence of backlash make compliant mechanisms a convenient and practical tool to be used in design for the use in medical and micromanufacturing fields.

The application of these unique qualities can lead to innovations in prosthetics, like Mahler's design [19] of a compliant joint mimicking the human knee for pediatric patients. He was able to implement a design that maintained the requirements of light weight, durability, and stability, while maintaining simplicity and a single-joint design. He was also able to address the unique scenarios that pediatric knees tend to find use, like the harsh environments that children tend to play in, such as water or sand. Allowing adjustability required to accommodate for differences in gait between individuals was also proposed. The prototype design is shown in Figure 12.

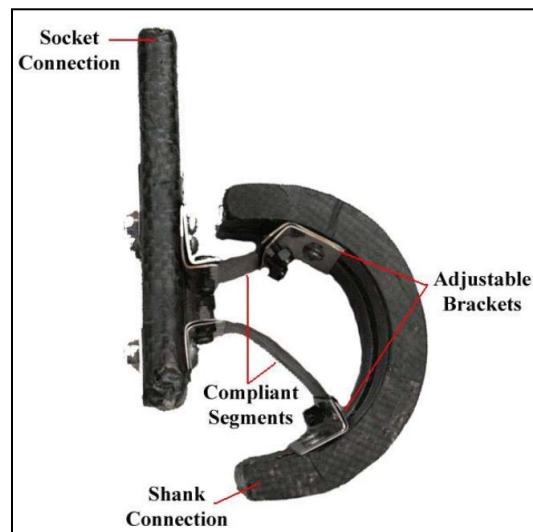


Figure 12. Prototype design of Mahler's pediatric prosthetic knee [19].

Innovations in compliant mechanism design can also be applied to the design of microstructures, since there are requirements for carefully calibrated flexures. This has been applied many times for use within MEMS. All traditional compliant mechanism design methods can be applied, and single part manufacturing of compliant mechanisms make miniaturization simple [4]. Kyusojin and colleagues [20] designed a compliant mechanism that implements flexible strips in parallel, relying on their good linear displacement characteristics to move precise horizontal distances while minimizing vertical displacement. They were able to conclude that a submicron degree of accuracy could be achieved in the horizontal movement, with a sufficiently long platform. Due to the simplicity of the instruments, the results showed good repeatability and reproducibility characteristics. This study laid the ground work for many other compliant mechanism designs that were developed later.

Another precision translational compliant mechanism was developed by Xu and colleagues [21], who achieved very precise linear movements with their design. It showed over 5000 linearity within a 5 mm displacement, and their work proposed that large deflection flexural joints can be developed into approximate straight line joints, with incredibly high precision.

Other work that increased the overall knowledge of compliant flexures' characteristics includes Tian's development of closed form equations for V-shaped flexures [22]. Typically, compliant mechanisms devised of multiple flexures use notch hinges or circular flexure hinges, due to the simplicity of manufacture, but the development of descriptions of geometry and deflection characteristics allows V-shape

flexures to be better defined, with the goal of wider integration. A unique compliant mechanism was developed by Goldfarb and Speich [23], who created a unique revolute joint called a split tube flexure. It was defined by its ability to achieve “well behaved” motions, which were defined as the optimization of compliance in the desired rotation direction, while maintaining stiffness along structural axes. This compliant mechanism can be designed to be used in macrostructure applications, due to its large potential rotation and fixed axis of rotation.

Trease and colleagues [5] designed multiple translational and rotational compliant mechanisms that were developed to have an increased range of motion and good movement characteristics. The intention of this design was minimizing the traditional drawbacks of compliant mechanisms. These were verified through finite element analysis, with attention to stress concentrations, to ensure the joints can be utilized in multiple scales of size.

Guérinot [24] used compliant joints in the unique arena of support of compression loads. This work attempts to avoid the drawback of compliant joints of being unable to handle high compressive loads due to buckling. His work includes two principles, isolation and inversion, which can be applied to compliant joints to increase the maximum load handled by avoiding buckling-prone conditions. Isolation involves the isolation of the flexible segments from the compressive load that the system must carry. The large compressive load will be diverted from the flexible segments to the rigid elements. In this case, the compliant sections are limited to precise motion control and

energy storage that benefit the device carrying large compressive loads. An example of this isolation theory is shown in Figure 13.

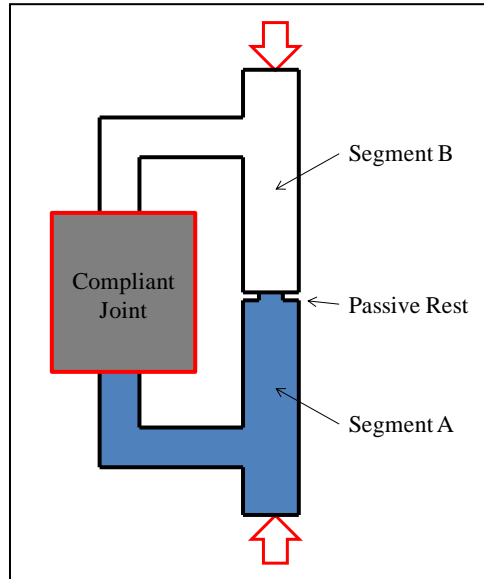


Figure 13. Schematic of an isolation joint system [24].

The inversion principle uses the proposition of tensorial pivots, which are flexures loaded in tension. This follows the idea that flexible segments generally have a higher maximum tensile force before yielding than critical buckling force. Therefore, a system made to function under a direct compressive force can be inverted in order to use flexible segments in tension rather than compression [24]. The concept inversion is illustrated in Figure 14.

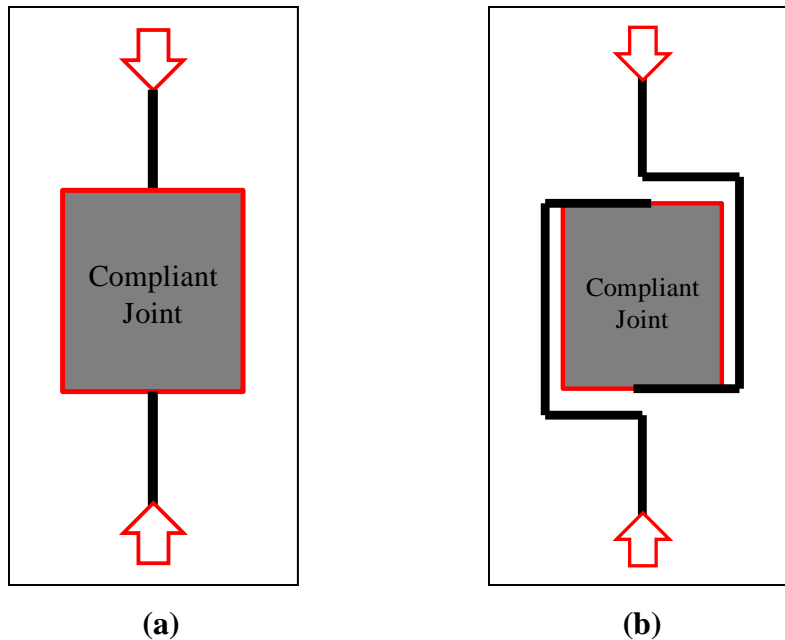


Figure 14. (a) Direct compression and (b) application of the Inversion theory [24].

2.3 Conclusions from the Literature Review

Compliant mechanism design and development techniques have improved vastly since their introduction. As the desire to build more complex compliant mechanisms grew, so did the requirements of the techniques to build them. Although these techniques have improved over the years, there are still gaps in them to be addressed. For pseudo-rigid-body methodology, this consists of the burden of knowledge on the designer to understand the catalogue of transformations between compliant mechanisms and their pseudo-rigid-body representations. For topology optimization, the burden of knowledge on the designer is the understanding of the creation of a design space that will get the correct and satisfactory results.

Many compliant mechanisms have been designed for use in specific and diverse applications. The advantages of compliant mechanism when compared to traditional

mechanical mechanisms allow their use in a wide variety of applications where traditional mechanisms fall short. However, this breadth of use has not been fully integrated into a wide variety of processes, possibly due to the lack of knowledge and experience needed to design these mechanisms.

CHAPTER 3: DEVELOPMENT OF A COMPLIANT JOINT MODEL REPOSITORY

A compliant joint selection method must produce results that lead a user to an optimal solution for their requirements. To facilitate this, a repository of compliant joints is required that provides solutions for the widest variety of user needs possible. This chapter details the process used to convert current research publications of compliant joints designs into models to be used within the repository and selection algorithm.

Six translational compliant joints (also known as linear compliant joints) and fourteen rotational compliant joints are included in this repository. These were chosen to represent a wide variety of applications, as well as distinct characteristics between different compliant joints. The compliant joints included in the repository have varied characteristics in the regions of precision, stiffness, and mobility.

The six linear compliant joints were defined as compliant joints that move and have compliance in the linear x-direction when a force is applied. These joints typically have little rotational and translational compliance and movement in the y and z directions, which is sometimes called the off axis compliance/movement. Generally, the movement in these directions would be minimized so that the intended linear movement of the joint will be more accurate. Movement in these directions that is not intended is called parasitic movement. A diagram of the expected movement of the translational compliant joints is shown in Figure 15.

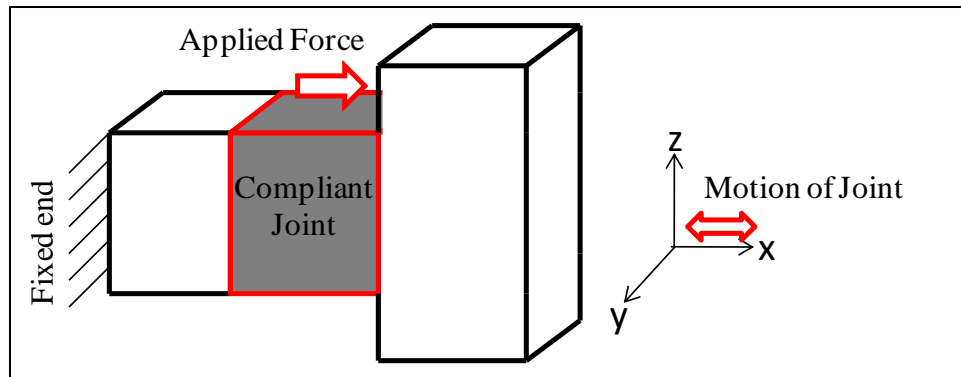


Figure 15. Defined movement of rotational compliant joints.

Fourteen rotational joints were chosen from current publications and research. These rotational joints were defined as compliant joints that can rotate an end around the y-axis when a force is applied to the end of the joint in the z-direction. These joints have little compliance and range of motion in the z- and x-directions – this is normally minimized through the development of the compliant joint. It is important to note that there is much of the research develops rotational joints prioritizes the minimization of axial drift, or motion that moves the axis of rotation of the joint away from its original position. This is typically a goal of MEMS research, where accuracy and precision are the most important characteristics of a compliant joint. A diagram of the expected movement of the rotational compliant joints is shown in Figure 16.

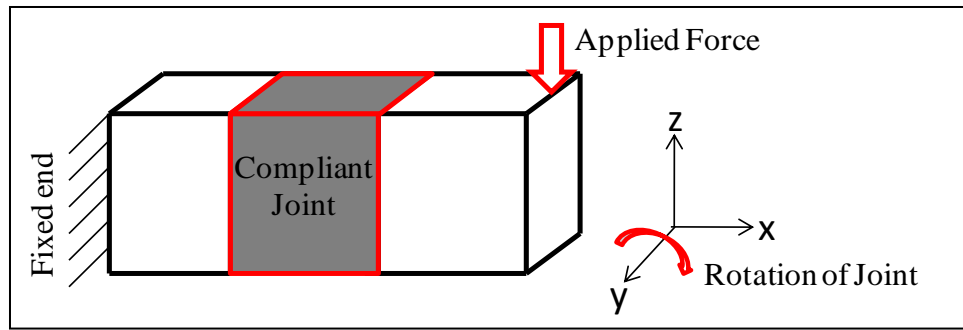


Figure 16. Defined movement of rotational compliant joints.

3.1 Adding Compliant Joints to the Repository

A defined methodology needs to be followed when adding new compliant joints to the repository to ensure that each joint has consistently defined inputs and outputs. This establishes direct comparisons between compliant joints of the same type, even when the characteristics they were designed for differ greatly. The process used to develop and add new compliant joints to the repository is outlined in Figure 17.

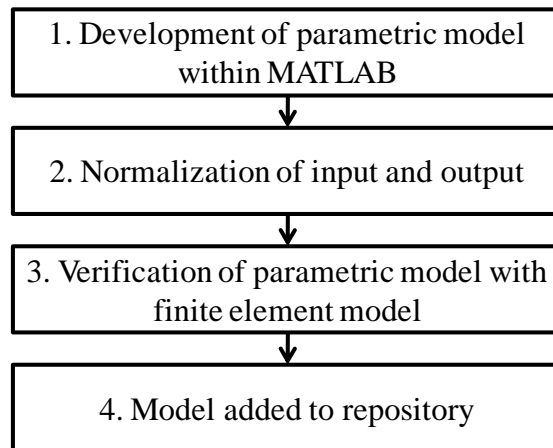


Figure 17. Outline of process used to develop models of compliant joints.

The joints are first developed as parametric models within MATLAB, which includes defining the joint's movement and stiffness characteristics as functions of the

geometric characteristics of the joint. Then the compliant joints are normalized, since all joints do not have their characteristics outlined identically within research. This normalization consists of manipulating the compliant joints models to produce outputs of stiffness (K/mm for translational and K/degree for rotational) and displacement (mm or degrees). This normalization is used not to reduce the complexity of the compliant joints' parametric equations, but to allow joints with different applications to be compared on similar performance standards. These standards were determined to allow a user to specify basic information about the compliant joint, so that more results could be selected for the user. Verification of these parametric equations is performed using FEA simulations that compared the displacement output of the joint from the parametric equations to the displacement output of the FEA model. Finally, if the models were proven to simulate the compliant joint correctly, the model was added to the repository for use in the selection algorithm.

Two criteria were necessary of the compliant joints for them to be selected for the repository. These criteria are:

1. The joint must be characterized by parametric equations, which defined the movement of the joint.
2. The joint must be geometrically defined so that it was possible to construct the joint within finite element analysis (FEA) software.

These are required so that normalization and verification of the joint's parametric equations could be performed. The geometric dimensions of the compliant joints were

used in the search algorithm to determine a range of potential displacements and spring constants within the user defined constraints of the compliant joint.

3.1.1 Development of Parametric Models within MATLAB

The parametric models are developed within MATLAB as individual program files. Parametric equations are derived from current research publications on the design of specific compliant joints. These equations are used such that a single MATLAB code file contains all the information necessary to test the input and output relations of the compliant joint. In all cases, the geometric dimensions of the joint need to be defined as variables, with appropriate comments included so that the joint could be later constructed within the FEA software. These geometric variables typically include the thickness of leaf springs, the length of leaf springs, the depth of the joint, and other characteristics unique to the individual joint design. The file also must include material properties and an applied force that is used with the stiffness to determine the amount of displacement the joint experiences.

The MATLAB files are named using the developer's name, and a brief description of the compliant joint. For example, a compliant joint from Smith [25], which was identified as a cartwheel shape, would be named `SmithCartwheel.m`. To ensure consistency, this name is also used for the MATLAB function files and FEA models that will be discussed later in this chapter.

3.1.2 Normalization of Inputs and Outputs

Normalization of inputs and outputs of the compliant joint models is performed to ensure consistency between all of the compliant joint models, as well as giving a standard method of comparison between different compliant joints. The MATLAB models are transformed into MATLAB function files, which use inputs of material properties, geometric constraints, and applied force. The outputs of these functions are the displacement of the joint and its stiffness in the movement direction. The parametric equations are normalized such that these inputs and outputs are the same for every MATLAB function. For example, literature containing a rotational compliant joint may produce its end displacement value as a linear distance. The model is required to have a displacement of a rotational value in degrees so that it can be related to the other rotational compliant joints within the repository.

Each function runs under an assumption that the thickness of leaf springs or similar parameters within each model is 20% of the “smallest size” entered by the user. This is used because the accuracy of the models increases as the ratio of length to thickness of the leaf springs increases. This also provides a larger range of results for the range of motion. Without this assumption the displacement ranges are small and difficult to pinpoint for a user to get satisfactory results. It is the user’s responsibility to fill this parameter to adequately represent their manufacturing capabilities.

3.1.3 Verification of Parametric Models with Finite Element Models

The parametric equations that define the motion of the selected compliant joints are verified using an FEA program. For translational compliant joints, the x-direction

motion is directly compared between the parametric model and the FEA model, using a consistent applied force. For the rotational compliant joints, the rotational motion is determined using the end deflection, and then is converted into a linear value for comparison with total deflection within the FEA software. The results of the two models are compared to determine if the parametric model adequately described the motion of the joint. However, due to limitations of the parametric equations, some of the equations lose accuracy as the deflection becomes increasingly nonlinear. Therefore, it is recommended that a user not exceed potential deflections of more than 30% of the total length of the joint without doing additional studies of the compliant joint's movement. This will maintain a reasonable degree of accuracy.

3.1.4 Model Added to Repository

The model is added to the repository so that it can be used in the selection algorithm alongside the other compliant joints. The geometric characteristics are pre-allocated using the user defined size constraints so that a maximum displacement value/minimum stiffness value is calculated, as well as a minimum displacement value/maximum stiffness value. The algorithm then will select models that fulfill the user's requirements, using these limits.

3.2 Example 1: Translational Compliant Joint – Smith Notch Hinge

The following section will exemplify the process of developing a translational compliant joint model, specifically the Smith Notch Hinge. This compliant joint is detailed by Smith [25] as a compound compliant joint that approximates straight line

motion. The geometric characteristics that define this compliant joint are shown Figure 18.

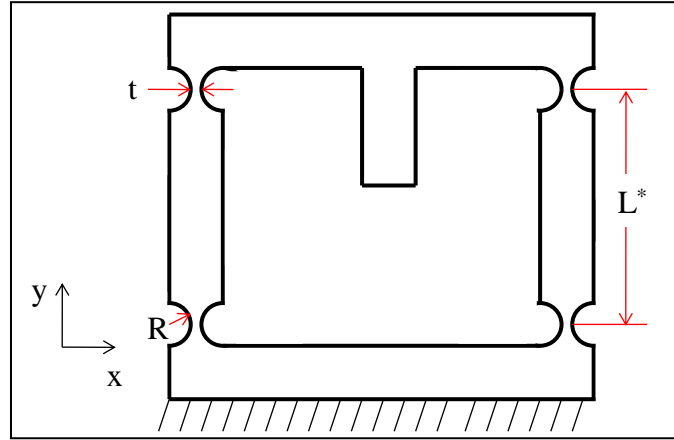


Figure 18. Geometric characteristics of the Smith Notch Hinge compliant joint.

The parametric equations that represent the movement characteristics of this compliant joint are as follows. The total ‘static’ linear stiffness of this flexure is defined by

$$\frac{F_x}{x} = \frac{8Ebt^{5/2}}{9\pi R^{1/2}L^{*2}}, \quad (1)$$

where E is Young’s modulus, b , t , R , and L are geometric characteristics, x is movement in the x-direction of the movement point, and F_x is the force applied in the x direction to the upper table of the compliant mechanism.

From the equation, an MATLAB model is developed to represent the characteristics of the compliant joint. This facilitates early testing of the model with various parameters. The MATLAB model, `SmithNotchHinge.m`, is shown in Figure 19.

```

E = 73000; %N/mm^2 (73.1 GPa)
%Young's Modulus

%%----Joint Dimensional Characteristics----%%
%Length between flexures edges
L=9.2; %mm
%Depth of joint
b=1; %mm
%Radius of flexures
R=0.4; %mm
%Thickness between flexures
t=0.2;
%Force Applied
F=1;

%%----Solution of Joint----%%
%Distance from centerpoint of flexures
Lstar=L+2*R;
%Stiffness
K=(8*E*b*t^(5/2))/(9*pi()*R^.5*Lstar^2);
%Linear Displacement
Disp=F/K;

```

Figure 19. The initial MATLAB file used to test the Smith Notch Hinge compliant joint model, `SmithNotchHinge.m`.

This model must now be normalized to ensure that this compliant joint model is consistent with the others in the repository. All of the linear compliant joints must have outputs of a linear stiffness as well as a linear range of motion, so that it is possible to compare them to one another. The MATLAB file is changed into a MATLAB function file, where the outputs are the stiffness and deflection and the inputs are the material properties, geometric constraints, and force applied. A representation of the resultant function is shown in Figure 20.

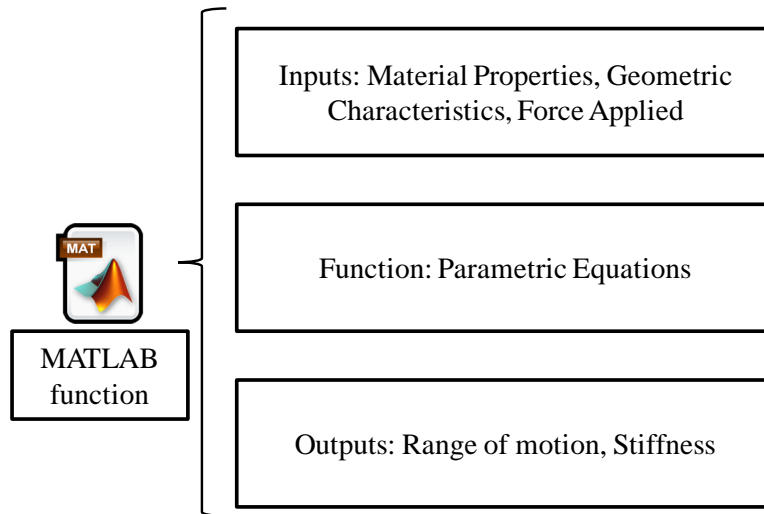


Figure 20. The completed Smith Notch Hinge model, as a MATLAB function.

An FEA model is developed of the compliant joint to compare with the parametric model. The geometric values used to calculate the two models are listed in Table 2.

Table 2. Geometric properties of the Smith Notch Hinge compliant joint used for comparison between the MATLAB model and the FEA model.

Parameter	Value
b	10 mm
t	0.2 mm
r	0.4 mm
L	9.2 mm
F	1 N

The boundary conditions between the two models must be the same. Since the parametric models are already developed for certain boundary conditions, the FEA model is made to mimic the parametric model. The boundary conditions used for both model simulations are given in Figure 21.

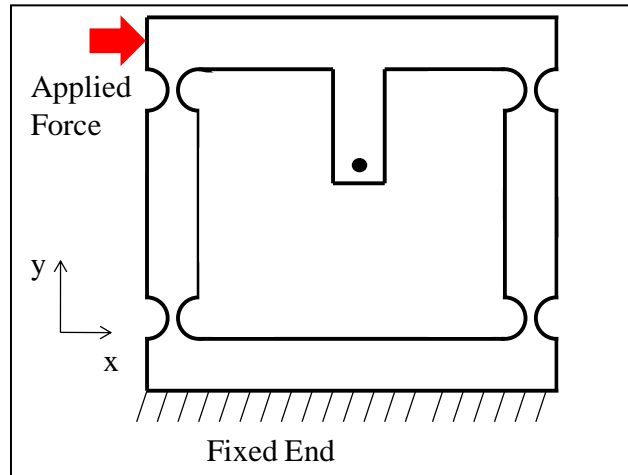


Figure 21. Boundary conditions used for the Smith Notch Hinge compliant joint model simulations.

The FEA model required a mesh with necessary refinement in the areas of interest in order to provide correct results. When the compliant joint deformed as expected and represented the literature response correctly, the FEA model was determined to be providing correct results. A mesh of elements approximately 0.5 mm in length was used. The mesh and deflected FEA model are shown in Figure 22.

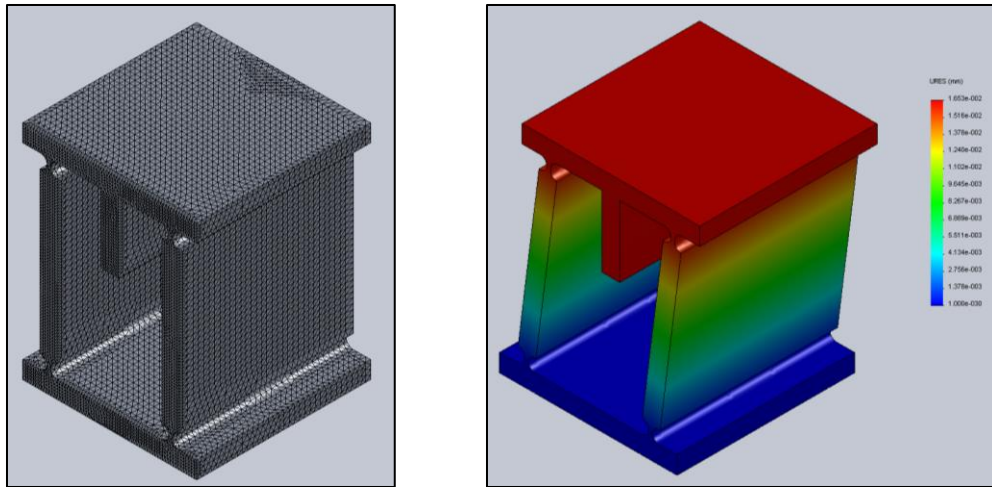


Figure 22. Mesh and deformed shape for the Smith Notch Hinge FEA simulation.

The result of this comparison is a deflection of 1.71×10^{-2} mm from the parametric model and a 1.65×10^{-2} mm deflection using the FEA model. These two models have a difference of 5.70×10^{-4} mm, and a percent difference of 3.45%. This result shows that both the models have similar deflection characteristics, and the parametric model can be considered correct. For most of the compliant joints models, as displacements becomes more nonlinear the accuracy is reduced. It is important to note that the FEA simulation is simply used for verification of the parametric model, and the final design does not necessarily reflect the characteristics used for the FEA representations.

3.3 Example 2: Rotational Compliant Joint – Jensen Cross Axis

The following section describes the process of developing a rotational compliant joint model, specifically the Jensen Cross Axis. This compliant joint’s load-deflection behavior was detailed by Jensen and Howell [26], and verified by comparing results to

non-linear FEA models and physical models made of polypropylene and steel. The geometric characteristics that defined this compliant joint are shown in Figure 23.

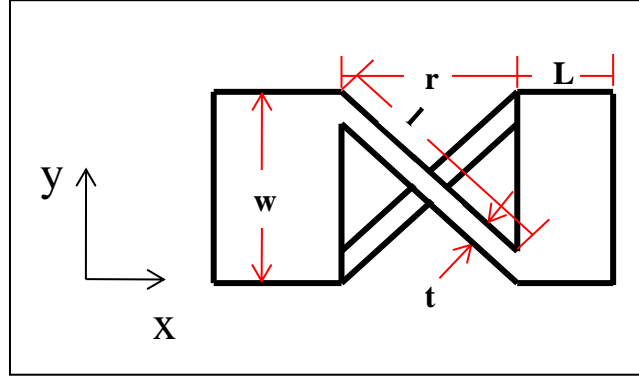


Figure 23. Geometric characteristics of the Jensen Cross Axis compliant joint.

Jensen and Howell [26] describe this compliant joint's characteristics as if it were a torsional spring. In this case, the spring stiffness is given by

$$K = \frac{K_{\theta}EI}{2l}, \quad (2)$$

where E is Young's modulus, I is the moment of inertia of the flexible sections, l is the length of the flexible segments, and K_{θ} is known as the "stiffness coefficient". The stiffness coefficient is determined over a range of values, which are a function of a geometric relationship n , where

$$n = rw. \quad (3)$$

The polynomial curve fit that defines the relation of n with K_{θ} is

$$K_{\theta} = 5.300185 - 1.6866n + 0.885356n^2 - 0.2094n^3 + 0.018385n^4. \quad (4)$$

This curve is valid for $0.5 \leq n \leq 4.0$, with a correlation coefficient of 0.99910 [26].

From the equations provided, an initial MATLAB model can be developed to test different parameters with this compliant joint model. The initial MATLAB model, `JensenCrossAxis .m`, is shown in Figure 24.

```

E = 73000; %N/mm^2 (73.1 GPa)
%Young's Modulus

%%----Joint Dimensional Characteristics----%%
%Total length of gap
r = 10; %mm
%Length of cross spring
l = 14.142136; %mm
%Height of joint
w = 10; %mm
%Thickness of Leaf Spring
t = 0.707107; %mm
%Lever arm
L = 0.1; %mm
%Depth of joint
D = 10; %mm
%Force Applied
F = 1; %N

%%----Solution of Joint----%%
n=r/w;
KTheta=5.300185-1.6866*n+0.885356*n^2-0.2094*n^3+0.018385*n^4;
I=(1/12)*D*t^3;
%Stiffness
K=(KTheta*E*I)/(2*l);
Torque=F*(L+r);
%Angular Displacement
Disp=Torque/(K);

```

Figure 24. The initial MATLAB file used to test the Jensen Cross Axis compliant joint model, `JensenCrossAxis .m`.

The normalization of this model is performed next through the transforming of it into a MATLAB function. This ensures that this compliant joint model is consistent with the other models in the repository. Since some parametric equations for rotational compliant joints were developed with results of linear displacements, it is required to normalize them to ensure all rotational compliant joint models have the same outputs of rotational displacement and rotational stiffness. It is also required that the model have

outputs of the stiffness and deflection, and inputs of the material properties, geometric constraints, and force applied. Figure 25 shows the resultant function from this normalization, where the final line shows a normalize of the stiffness to Nmm/deg.

```
function [Disp,K] = JensenCrossAxisFun( E,t,w,r,D,F)
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----%%
% r = Total length of gap
%mm
% l = Length of cross spring
% w = Height of joint
% t = Thickness of Leaf Spring
% L = Lever arm
% D = Depth of joint
% F = Force Applied

%%----Solution of Joint----%%
L=0.1; %
t=t/5;
x = 1.404*t; %estimation, can be determined geometrically

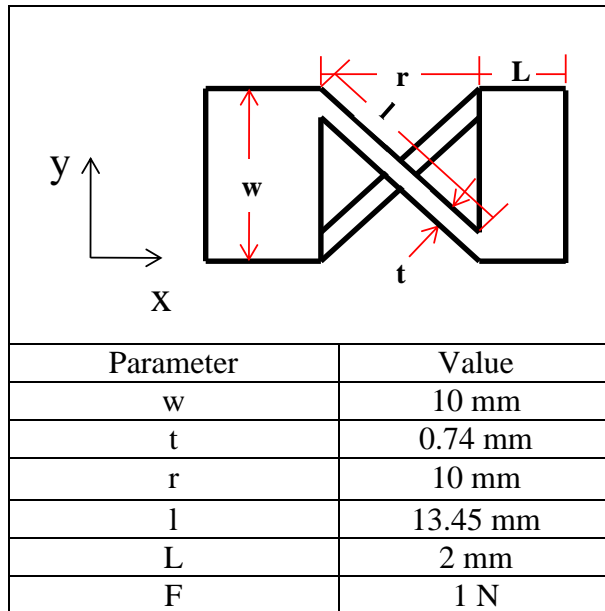
l=sqrt((w-x)^2+r^2);
n=r/w;

KTheta=5.300185-1.6866*n+0.885356*n^2-0.2094*n^3+0.018385*n^4;
I=(1/12)*D*t^3;
%Stiffness
K=(KTheta*E*I)/(2*l);
M=F*(L+r);
%Angular Displacement
Theta=M/(K);
-%Transform stiffness to Nmm/deg
K=K*pi()/180;
end
```

Figure 25. The completed Jensen Cross Axis model, as a MATLAB function.

The FEA model must be constructed to represent the inputs used by the parametric model. The geometric values used to compare the two models are listed in Table 3.

Table 3. Geometric properties of the Jensen Cross Axis compliant joint used for comparison between the MATLAB model and the FEA model.



The boundary conditions that the two models use to calculate the displacements must be the same. The parametric models are has defined boundary conditions, so the FEA model was created to represent the parametric model. The boundary conditions used in both model simulations are given in Figure 26.

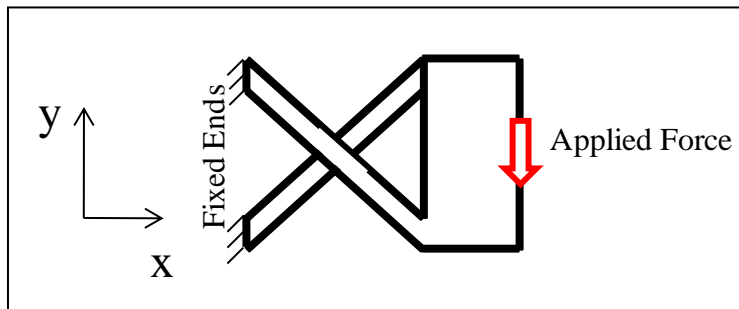


Figure 26. Boundary conditions used for the Jensen Cross Axis compliant joint model simulations.

The FEA model was composed of a mesh of elements approximately 0.5 mm in length. This was determined to provide a result in line with literature defined responses. The accuracy of the FEA was determined to be adequate because extreme accuracy is not required for the calculation, due to the FEA results only being needed verify the parametric equations. The mesh and deflected FEA model are shown in Figure 27.

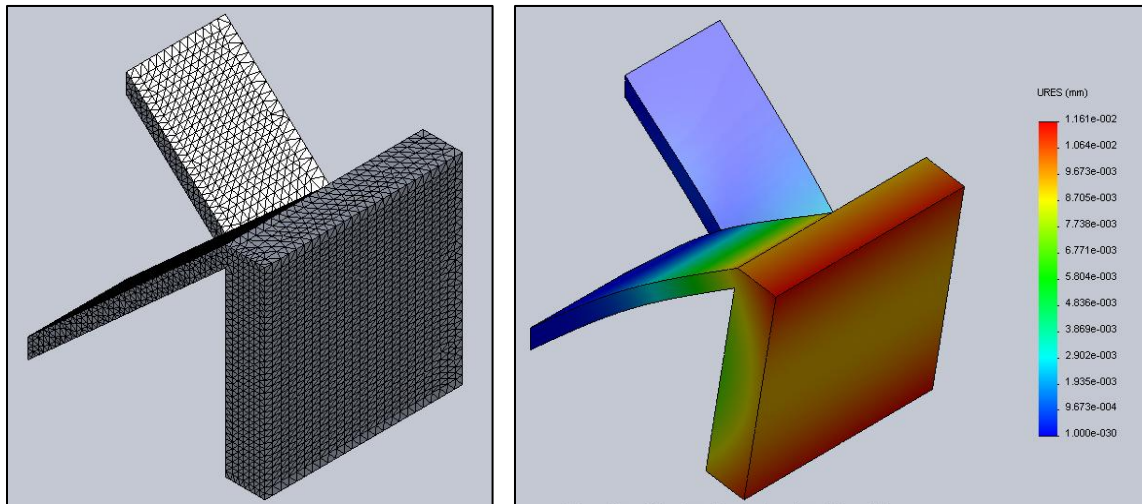


Figure 27. Mesh and deformed shape for the Jensen Cross Axis FEA simulation.

The result of this comparison is a deflection of $9.9\text{e-}3$ mm given by the parametric model and a $9.6\text{e-}3$ mm deflection using the FEA model. This has a difference of $2.27\text{e-}4$ mm, and a percent difference of 2.35%. This result shows that the models are experiencing a similar deflection amount. It should be noted, however, that many of the parametric equations lose accuracy as deflection increases.

3.4 Compliant Mechanism Repository Results

A full catalogue of the compliant joints in the repository is listed in Appendix A. All compliant joints in the repository were validated using this method. Appendix A

includes their parametric equations as listed in the original research, diagrams of their geometric characteristics, and their MATLAB function models. Table 4 lists the names given to the translational compliant joints that are stored within the repository, the literature reference where the compliant joint's model was sourced, and a 3D computer-aided design (CAD) solid model representation. These joints have a variety of characteristics, some being focused on accurate movement, while others focusing on maximum possible displacement. It should be noted that some linear compliant joints are created from a set of rotational compliant joints attached to a "table" which moves. This allows small rotational movements to be transferred into linear movements.

Table 4. Translational Compliant Joint Repository List

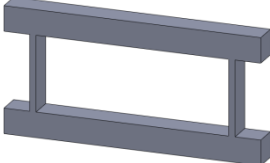
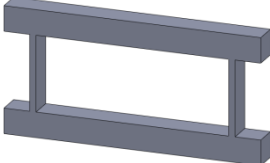
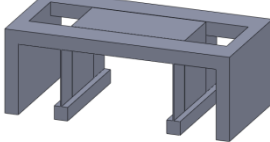
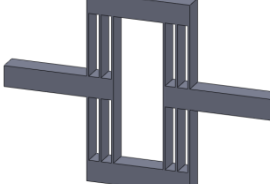
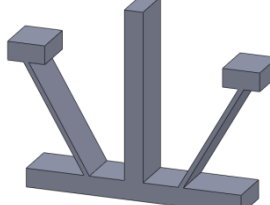
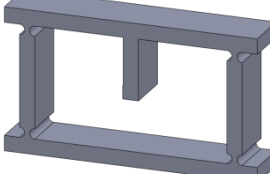
Compliant Joint Name	Solid Model Representation
Smith Rectilinear [25]	
Kyusojin Parallel Strip [20]	
Kyusojin Linear 6L1 [20]	
Trease Translational [5]	
Xu Translational [27]	
Smith Notch Hinge [25]	

Table 5 lists the rotational compliant joints stored in the repository, as well as their sources, and a 3-D CAD solid model representation. These compliant joints were designed for many different applications. Some maintain higher precision of movement,

others maximize rotational movement (these are typically called large-displacement compliant joints), and others were developed for the minimization of axial drift.

Table 5. Rotational Compliant Joint Repository List

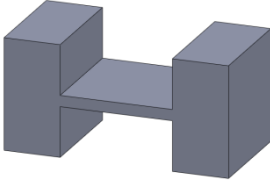
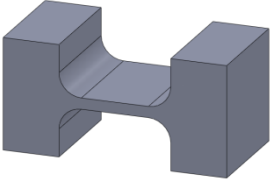
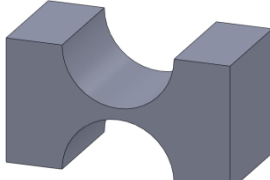
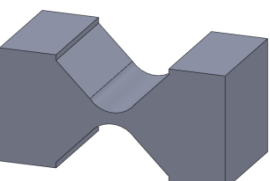
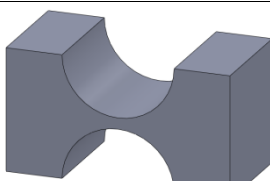
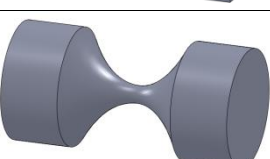
Compliant Joint Name	Solid Model Representation
Lobontiu Symmetric Notch [3]	
Lobontiu Corner Filleted [3]	
Lobontiu Symmetric Circular [3]	
Tian V Shape Flexure [22]	
Tang Symmetric Circular [10]	
Smith Two Axis [25]	

Table 5 Cont'd. Rotational Compliant Joint Repository List

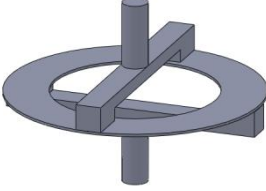
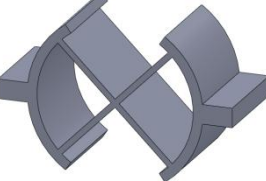
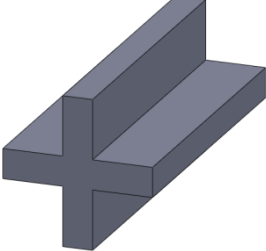
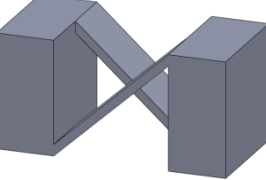
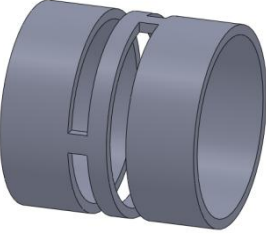
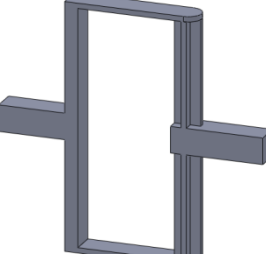
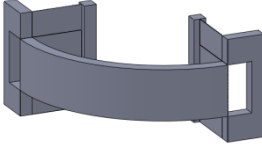
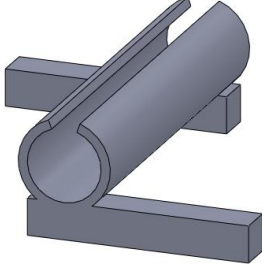
Compliant Joint Name	Solid Model Representation
Smith Annulus [25]	
Smith Cartwheel [25]	
Smith Cruciform [25]	
Jensen Cross Axis [26]	
Smith Rotationally Symmetric Leaf Hinge [25]	
Trease Rotational [5]	

Table 5 Cont'd. Rotational Compliant Joint Repository List

Compliant Joint Name	Solid Model Representation
Kyusojin Rotational 6R2 [28]	
Goldfarb Conventional Split Tube [23]	

All of the compliant joints in these two lists, translational and rotational, were modeled using the method described previously in this chapter. Each model was validated through comparison to a matching FEA model. All of the comparisons were performed using a material of aluminum 2024. The results of the comparisons for the translational compliant joints are shown in Table 6. The dimensions used for comparison are related to examples found in the literature of the compliant joint. The locations of the geometric dimensions used in the models can be found in Appendix A.

Table 6. A comparison of FEA models and parametric equations for translational compliant joints for a unit force of 1 N.

Compliant Joint Name	Dimensions used for Comparison [mm]	Deflection [mm]		Difference [mm]	% Difference
		FEA	Parametric		
Smith Rectilinear	t=0.1, b=10, L=10	6.15E-01	6.85E-01	7.04E-02	11.46%
Kyusojin Parallel Strip	t=0.1, b=10, L=10	6.15E-01	6.85E-01	7.04E-02	11.46%
Kyusojin Linear 6L1	t=1, w=50, l=50	2.74E-02	2.74E-02	1.00E-05	0.04%
Trease Translational	t=1, w=10, LB=30	1.19E-02	1.23E-02	3.60E-04	3.02%
Xu Translational	hf=8.75, H=20, Hp=26.89, t=0.5, dc=5	1.41E-02	1.10E-02	3.08E-03	28.00%
Smith Notch Hinge	R=0.4, t=.2, b=10, L=9.2	1.65E-02	1.71E-02	5.70E-04	3.45%

These results suggest that the parametric equations are reasonable estimates of the deflection of the joint, as they relate to the FEA models. However, the Xu translational compliant joint has inherent error due to its design. This compliant joint's linear motion is achieved through the deflection of long thin leaf springs, utilizing their motion to achieve small linear motions. This design can achieve precise movements in the micro-range. The precision of the joint decreases as the size of the joint increases. Due to open-ended nature of the design repository, joints that were designed for the purpose of small precise movements and large deflections were included to offer users a wide variety of potential selectable joints.

The results of the comparisons of the rotational compliant joint parametric equations and finite element analysis are shown in Table 7.

Table 7. A comparison of FEA models and parametric equations for rotational compliant joints for a unit force of 1 N.

Compliant Joint Name	Dimensions used for Comparison [mm]	Deflection [mm]		Difference [mm]	% Difference
		FEA	Parametric		
Lobontiu Symmetric Notch	t=1, w=10, l=10	5.20E-03	5.50E-03	3.01E-04	5.79%
Lobontiu Corner Filleted	r=2, t=1, w=10, l=10	3.59E-03	3.80E-03	2.14E-04	5.97%
Lobontiu Symmetric Circular	r=5, t=1, w=10	9.62E-04	1.10E-03	1.38E-04	14.31%
Tian V Shape Flexure	R=2.25, t=1, l=5, c=4, $\theta=20^\circ$, b=10	8.92E-04	8.47E-04	4.55E-05	5.37%
Tang Symmetric Circular	r=5, t=1, b=10	9.62E-04	1.10E-03	1.38E-04	14.31%
Smith Two Axis	r=5, t=1	1.69E-02	1.53E-02	1.55E-03	10.13%
Smith Annulus	r1=5, r2=3.25, t=.1	3.44E-01	3.18E-01	2.52E-02	7.92%
Smith Cartwheel	r=4, t=0.3, b=10	2.07E-02	1.95E-02	1.17E-03	6.00%
Smith Cruciform	t=1, L=10, d=2	2.00E-03	2.40E-03	4.02E-04	20.12%
Jensen Cross Axis	t=0.74, w=10, r=10, D=5	9.67E-03	9.90E-03	2.27E-04	2.35%
Smith Rotationally Symmetric Leaf Hinge	d=1, Ro=5	2.54E-03	2.70E-03	1.64E-04	6.47%
Trease Rotational	L=40, t=1, w=10	6.99E-04	7.24E-04	2.45E-05	3.51%
Kyusojin Rotational 6R2	t=1, w=50, l=50	2.71E-01	2.74E-01	3.00E-03	1.11%
Goldfarb Conventional Split Tube	t=1, r=5, L=25	2.78E-03	2.10E-03	6.82E-04	32.48%

The results for the rotational compliant joints are similar to the translational compliant joints, in that most parametric equations represent the actual deflection within 15%. Two joints are outliers within this result, the Smith Cruciform joint and the Goldfarb Conventional Split Tube joint. These two joints rely on the torsional deflection

of the joint around an axis instead of conventional bending. Since the method of comparison used between the FEA model and the parametric model was a linear displacement (node to node for the FEA, a geometric relationship for the parametric models), this may have resulted in additional error. A node to node linear distance on a cross-section in the FEA will give a smaller estimation of rotational distance than the parametric model. This could be analyzed and corrected for using an additional verification of physically produced compliant joint models. This would also add another layer of robustness to this method, but is beyond the scope of what is covered in this thesis.

3.5 Conclusions from Compliant Joint Repository

A compliant joint repository of parametric equation based models has been developed. Each model is an individually packaged MATLAB function, which can be called through an overall MATLAB selection algorithm. Twenty models have currently been developed, which include six linear compliant joints and fourteen rotational compliant joints. The parametric equation function models have shown to produce similar deflection results when compared with 3-D FEA models. A benefit of this structure is that additional compliant joints could be added to the repository using this method. This would increase the number of potentially selected joints, and which would increase the likelihood a user achieve results of a compliant joint that adequately fulfills their requirements.

4.1 Selecting Compliant Joints from the Repository

The compliant joint selection algorithm detailed can be implemented by a user to find compliant joints from the repository that fit their requirements. The user must list their requirements in a format detailed enough to allow selection, but vague enough to allow the algorithm to select from the largest possible number of compliant joint models, which gives the algorithm the opportunity of picking the most adequate one. The overall compliant joint selection algorithm is handled by a MATLAB function that is listed in full in Appendix B. A flowchart of the selection algorithm's process is outlined in Figure 28.

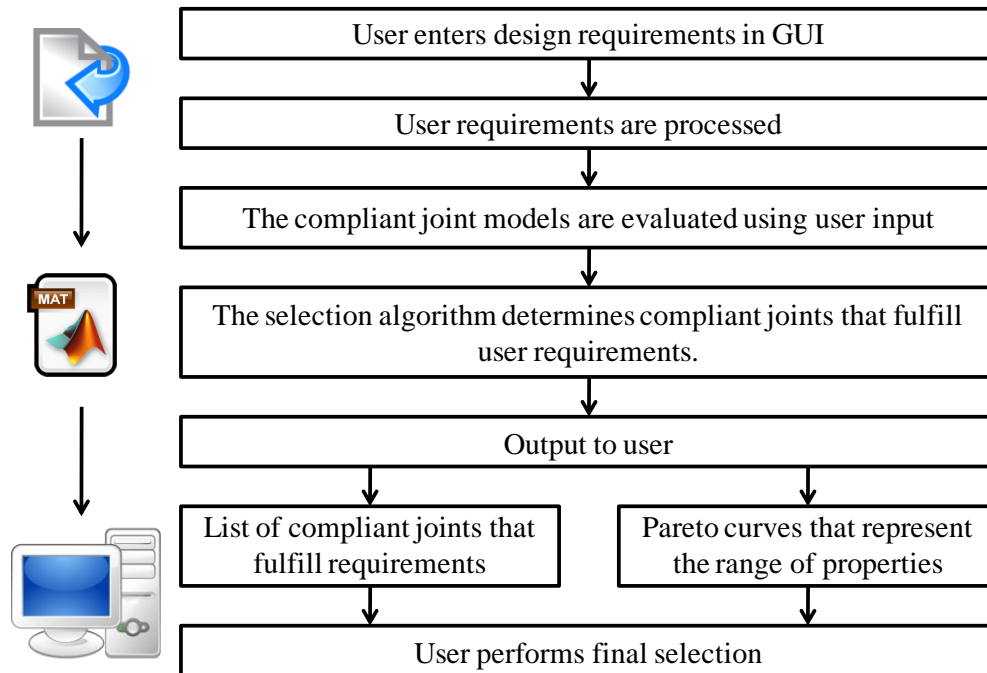


Figure 28. Overview of selection algorithm for determining compliant joints that fulfill the user's requirements.

For a designer to interact with the selection algorithm and find a compliant joint that fulfills their requirements, a user interface was desired. This Graphical User Interface (GUI) handles the transfer of information from the user to the algorithm. The user can specify up to six potential design requirements for the selection algorithm to use. The MATLAB code that generates the graphical interface is a MATLAB file, `JointMenu.m`, which is shown in Appendix C. The graphical interface that was developed is shown in Figure 29.

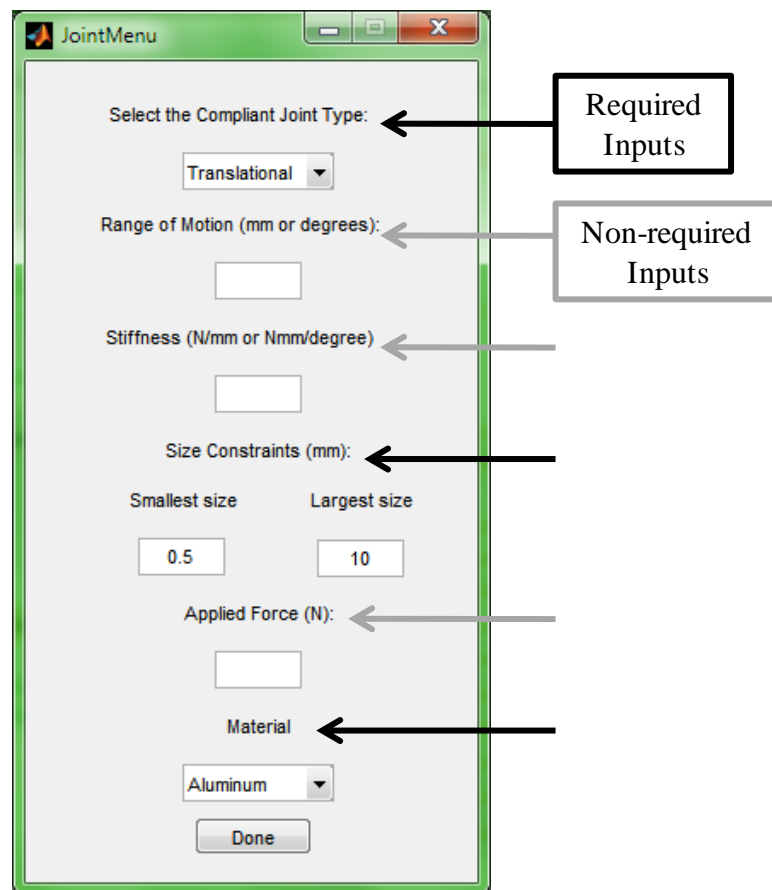


Figure 29. Graphical user interface for selection algorithm.

To achieve a final result, the three inputs designated by arrows in Figure 29, **Type of joint**, **Size constraints**, and **Material**, must be filled in by the user. If these are not defined, the selection algorithm cannot achieve a result. All three have default values that will be used if a user does not enter any input. This is done so that it is impossible for the user to not fulfill these requirements. It should be noted again that an assumption the algorithm uses within the individual models is that a user can manufacture compliant joints with thicknesses 20% of the *smallest size* text input. As stated previously, this is used because the accuracy of the models increases as the ratio of length to thickness of the leaf springs increase. This also provides a larger range of results for the range of motion. Without this assumption the displacement ranges are small and difficult to pinpoint for a user to get satisfactory results. It is recommended to use higher *smallest size* values when using ABS or PLA, due to the flexibility of these materials. Small thicknesses with these materials can cause large ranges of motion that are primarily non-linear, and will produce less accurate results. A user can also add additional materials to the GUI through the use of the `JointMenu.m` file, if so desired. The other available inputs are *Range of motion*, *Stiffness*, and *Applied Force*. It should be noted that range of motion should be entered as desired linear range of motion in millimeters for translational joints, and as desired rotational range of motion in degrees for rotational joints. Stiffness should be entered in the units N/mm for translational joints and Nmm/degree for rotational joints. Applied force is the force that is to be applied to the compliant joint, which can potentially be the force of a piston or motor that is used to actuate the joint. None of these three values are required for a solution to be displayed, however, and if

none are entered, the selection algorithm will return all possible joints, organized by range of motion, for specified applied force values (0.1 N, 1 N, 10 N). When the applied force is not specified, the same set of generic values will be used. These are to provide a sense of possibilities at difference scales of force values. It is possible for a user to change this set of applied force values by changing the `fset` variable in the `DecisiontreeFunction.m` MATLAB function file.

After a user has filled in the desired requirements in the GUI, the MATLAB function processes the user information. It ensures that all the data that has been entered is in numerical values. Then it processes which input data has been filled in and which is missing, so that it can execute the algorithm accordingly. Based on the user's initial requirements, different results can be given to the user.

The set of compliant joints (rotational or translational) that was selected by the algorithm have the displacement and stiffness values calculated through the parametric models. Each parametric model is utilized twice, to determine the minimum and maximum of both of these values. Then a search is performed between all of the compliant model's properties to determine which compliant joints have the potential to fulfill the user's design requirements. The algorithm saves this set of joints and performs post-processing.

During the post-processing, Pareto charts of the selected compliant joints are generated and displayed to the user. This allows the user to view the potential output parameters of the compliant joints within the constraints they have used. Pareto charts were determined to be an adequate display of satisfactory compliant joints due to their

ability to show a user the potential range of motion and stiffness values, across all geometric parameters within their size input. An example of this output is shown in Figure 30.

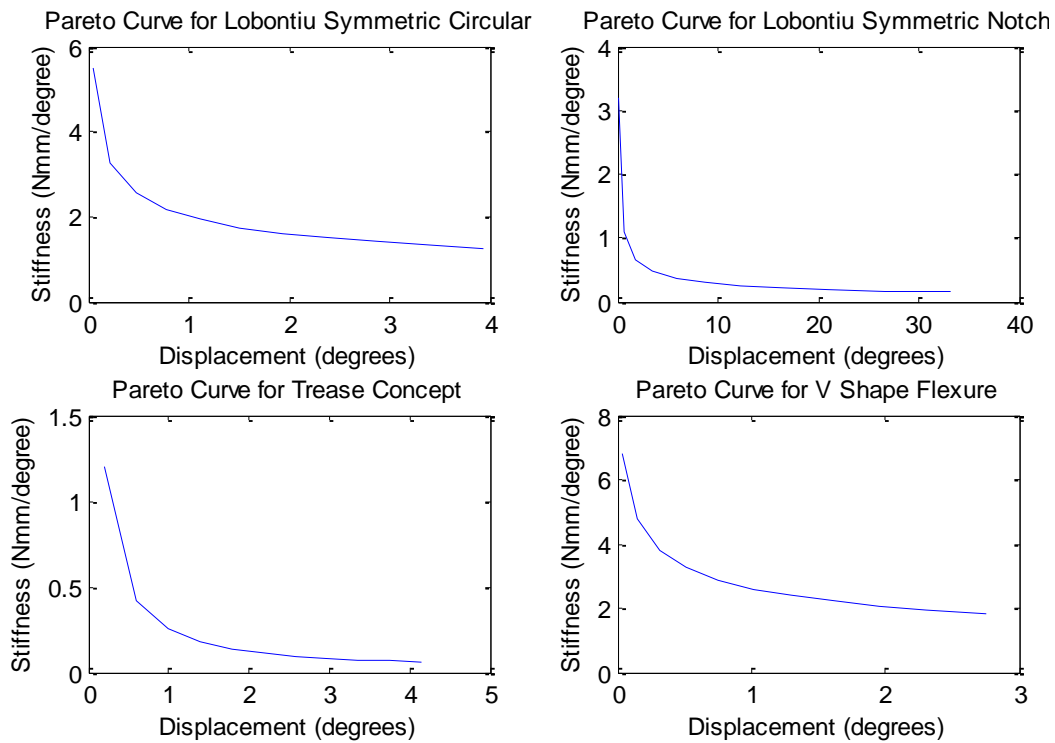


Figure 30. Example Pareto chart outputs from the algorithm.

The Pareto curves generated by the algorithm can be interpreted by the user in various ways. For each individual joint, the Pareto curve represents the range of stiffness and displacement values possible within the geometric constraints the user has chosen with the *size constraints* input. The charts are shown in parallel so that a user can see the comparative stiffness and deflections between the compliant joints that satisfy their requirements. For example, using the results in Figure 30, a user can determine that the Trease Concept rotational compliant joint and the Smith Cartwheel rotational compliant

joint have similar range of motion characteristics. However, the stiffness of the Smith Cartwheel compliant joint is potentially an order of magnitude higher than the Trease concept compliant joint. A user viewing these Pareto curves can make a determination of which compliant joint has better characteristics for their needs between the compliant joints that fulfill their requirements.

In addition to the Pareto charts, the algorithm displays the non required inputs the user has used and then lists the compliant joints that fit the user's requirements. For outputs that do not produce Pareto curves, this will also list the potential range of motion of the selected joints, as well as order them by this metric. An example of this output is shown in Figure 31.

```
>> JointMenu
Rotational Joints are selected.
RoM has a value, 1.00 degrees.
k does not have a value (na)
F has a value, 1.00 N.

Possible joints for this user input are:
JensenCrossAxis
LobontiuSymmetricCircular
LobontiuSymmetricNotch
SmithCartwheel
TangSymmetricCircular
TreaseConcept
VShape
ConventionalSplitTube
```

Figure 31. Example output from the selection algorithm.

The user must now select a compliant joint they determine most adequately fits the requirements they have used. The user can iterate here to continue the process of getting a satisfactory compliant joint, or complete the design of any specific compliant joint. It is relatively trivial at this point to use a parametric model in conjunction with an

optimization program to determine the exact geometric parameters that a compliant joint will require within the range of the selection algorithm.

Accuracy of the compliant joint models can be reduced when experiencing large nonlinear deflections. For the purposes of this research, the accuracy is assumed to be reduced when the compliant joint experiences deflections larger than 30% of the total length of that compliant joint. A warning was added to all output of the algorithm to inform users of potential inaccuracies caused by this assumption. This display of the limitations of the model allows the user to understand the range in which the model is considered valid, such that the user can gain a quantitative understanding of the context of the results of the model. It is considered that the publishing of this information will reduce the likelihood of abuse of the model [29].

4.2 Testing Procedure for the Compliant Joint Selection Algorithm

The selection algorithm methodology was tested through a series of inputs that were entered into the GUI. The results achieved were checked to test the precision and recall of the algorithm. The test inputs were selected to test all combinations of inputs used and unused. Table 8 contains the input conditions of the selection algorithm that were used in the tests shown within this chapter.

Table 8. Inputs for the test cases.

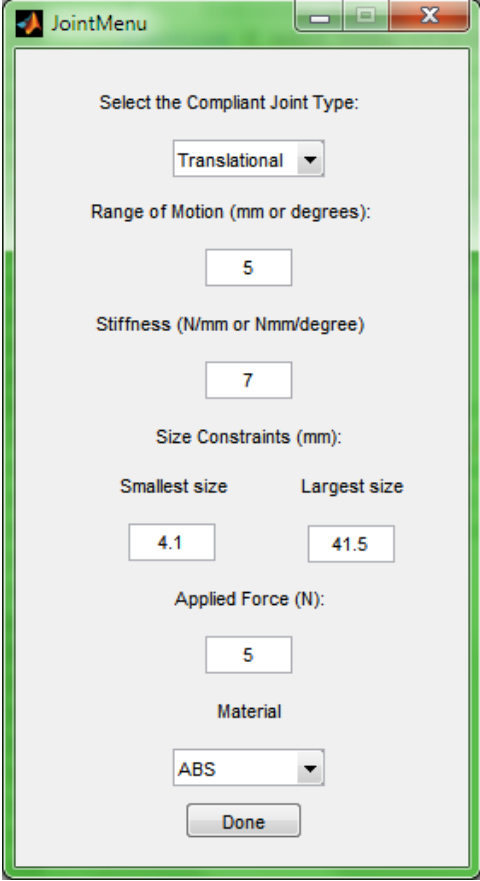
Test Case	Type of Joint	Range of Motion [mm or degrees]	Stiffness [N/mm or Nmm/degree]	Size Constraints [mm]	Applied Force [N]	Material
1	Translational	5	7	4.1 - 41.5	5	ABS
2	Rotational	10	3	1.2 - 29.2	-	Aluminum
3	Translational	2	-	1.7 - 27.5	3	ABS
4	Rotational	4	-	2.5 - 45.9	-	ABS
5	Translational	-	1	1.3 - 14.3	6	PLA
6	Rotational	-	2	3 - 37.9	-	Aluminum
7	Translational	-	-	2.4 - 19.0	2	PLA
8	Rotational	-	-	1.8 - 28.4	-	Aluminum

As shown in Table 8, four of the tests had translational compliant joints selected, and four of the tests had rotational compliant joints selected. *Range of motion*, *stiffness*, and *applied force*, when used, were randomly generated integers between 1 and 10 (mm or degrees, N/mm or Nmm/degree, and N, respectively). This value range was chosen because it was known to have guaranteed results within the solution space. The input *smallest size* was randomly selected between 1 and 5 mm, and the input *largest size* was randomly selected between 5 and 50 mm. The reasons for these ranges were that the largest size must be a larger number than smallest size, and the large range of the *largest size* variable allows for more potential differences between compliant joint results. The *smallest size* input was limited to values larger than 1 mm because the *range of motion* for many of the parametric models becomes incredibly nonlinear with very small thicknesses, especially when the material is a thermoplastic. The material was randomly selected using a random integer 1-3, with 1 representing aluminum, 2 representing PLA, and 3 representing ABS. Test Case 1 and Test Case 2 are presented in the following

subsections as examples. Each of the eight test cases' inputs, outputs, and explanations are shown in Appendix E.

4.2.1 Test Case 1

Test Case 1 considers the condition of a user entering all three of the non-required inputs. The algorithm must select compliant joints that satisfy the *stiffness* and *range of motion* requirements, using the *force applied*, *size constraints*, and *type of joint* inputs. The GUI showing the input values can be seen in Figure 32.



The screenshot shows a window titled "JointMenu" with a green title bar. The window contains the following input fields and controls:

- "Select the Compliant Joint Type:" with a dropdown menu set to "Translational".
- "Range of Motion (mm or degrees):" with a text input field containing "5".
- "Stiffness (N/mm or Nmm/degree)" with a text input field containing "7".
- "Size Constraints (mm):" with two text input fields: "Smallest size" containing "4.1" and "Largest size" containing "41.5".
- "Applied Force (N):" with a text input field containing "5".
- "Material" with a dropdown menu set to "ABS".
- A "Done" button at the bottom.

Figure 32. GUI input for Test Case 1.

Since all of the non-required inputs have been used, the output will be a list of compliant joints satisfying the requirements, and Pareto curves that show the range of properties the compliant joints can achieve, using the geometric characteristics that were initialized from the size constraints. Only one compliant joint within the repository satisfies the requirements that were input, the Smith Notch Hinge compliant joint. The Pareto output is shown in Figure 33, and the MATLAB text output is shown in Figure 34.

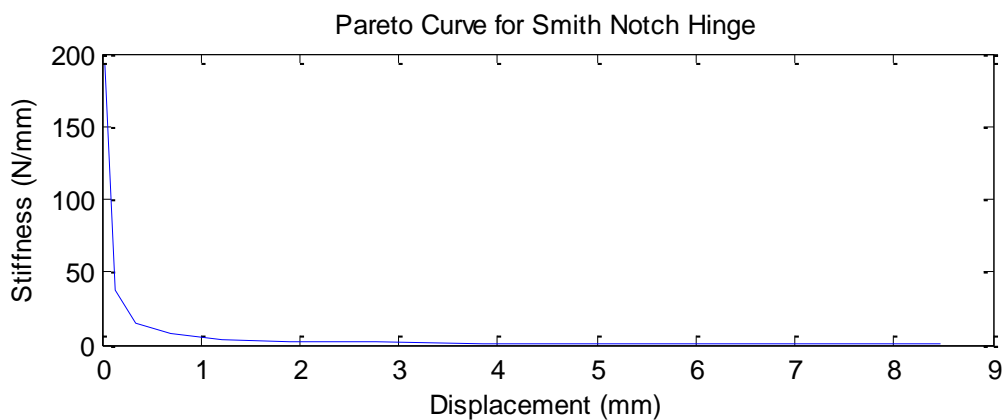


Figure 33. Pareto output for Test Case 1.

```
>> JointMenu
Translational Joints are selected
RoM has a value, 5.00 mm.
K has a value, 7.00 N/mm.
F has a value, 5.00 N.

Possible joints for this user input are:
SmithNotchHinge
```

Figure 34. MATLAB text output for Test Case 2.

4.2.2 Test Case 2

Test Case 2 considers the condition of the user entering a value for the non-required inputs *range of motion* and *stiffness* only. The GUI showing the input values is shown in Figure 35.

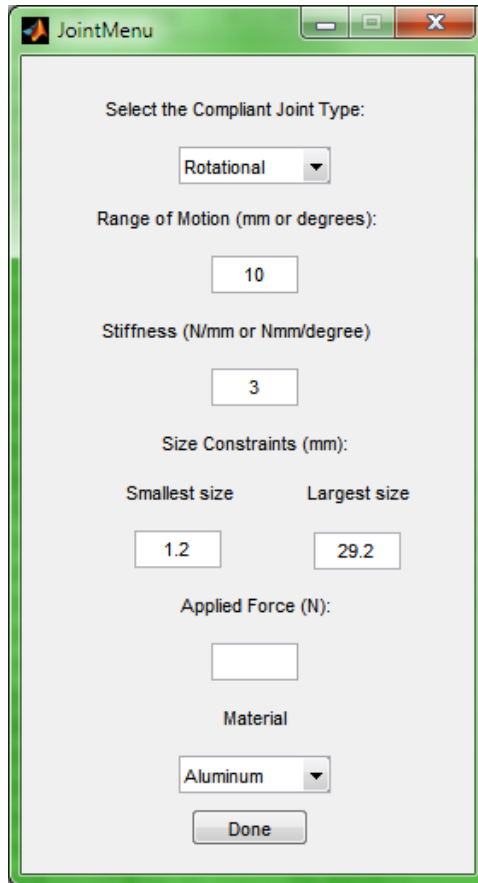


Figure 35. GUI input for Test Case 2.

When the applied force input is left blank, the selection algorithm uses a series of forces for the calculation of joint potential characteristics. This set of forces includes 0.1 N, 1 N, and 10 N. These values are an order of magnitude apart, and used to show the characteristics of the potential compliant joints at different scales. These allow a user to understand how the force applied affects the range of motion for the individual compliant joints. Due to the geometric characteristics of the compliant joint remaining constant, the stiffness of each set will remain the same. However, the displacement achieved by the different forces will change. This leads to Pareto curves that are identical on the y-axis (stiffness), but different on the x axis (displacement). Depending on the complexity of the

model, the displacement is usually a linear increase in relation to the force. This can be seen as a reasonable estimation until large non-linear displacements are reached.

In this test case, no results are returned for the first two generic applied forces. This means that there are no joints that will achieve the *stiffness* and *range of motion* required for those applied force values. The result of this is Pareto curve figures that are generated as blank. These figures were omitted for this reason. Two compliant joints satisfy the user requirements using a force applied of 10 N. The two joints are Jensen Cross Axis and Lobontiu Symmetric Notch. The Pareto curves for these compliant joints are shown in Figure 36. The MATLAB text output is shown in Figure 37.

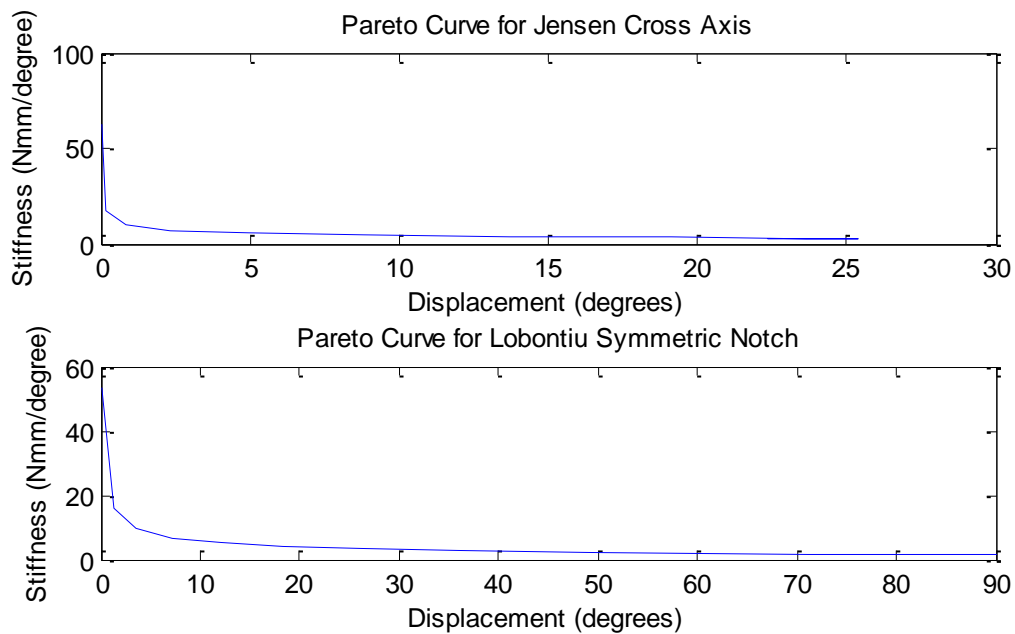


Figure 36. Pareto curves for Test Case 2, using an applied force of 10 N.

```
>> JointMenu
Rotational Joints are selected.
RoM has a value, 10.00 degrees.
K has a value, 3.00 N/mm.
F does not have a value (na)

Possible joints for this user input using a force of 0.100000 N are:

Possible joints for this user input using a force of 1.000000 N are:

Possible joints for this user input using a force of 10.000000 N are:
JensenCrossAxis
LobontiuSymmetricNotch
```

Figure 37. MATLAB text output for Test Case 2.

4.3 Validation of Test Cases

The test cases were validated through an analysis of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). A true positive is identified by being a correct solution to the design problem. A true negative is identified by being a non-solution to the problem. False positives and False negatives are solutions provided as correct when they are not, and solutions provided as incorrect when they are, respectively.

Test cases that had no applied force value are divided into a, b, and c sub cases, since the set of forces will be applied, and it will return three sets of results independent of one another. It should be noted that it is impossible to have a True Positive number of results of 20 due to translational compliant joints always being true negative for a rotational compliant joint search, and rotational compliant joints always being a true negative for a translational compliant joint search. The results for each of the test cases can be seen in Table 9.

Table 9. Test Cases, with the True Positives, True Negatives, False Positives, and False Negatives determined for each.

Test Case	True Positive	True Negative	False Positive	False Negative	Total
1	1	19	0	0	20
2a	0	20	0	0	20
2b	0	20	0	0	20
2c	2	18	0	0	20
3	4	16	0	0	20
4a	3	17	0	0	20
4b	9	11	0	0	20
4c	7	13	0	0	20
5	4	16	0	0	20
6a	1	19	0	0	20
6b	1	19	0	0	20
6c	1	19	0	0	20
7	6	14	0	0	20
8a	14	6	0	0	20
8b	14	6	0	0	20
8c	14	6	0	0	20

The validation shows that no False Positives or False Negatives were determined by the algorithm. Test Cases 2, 4, 6, and 8 returned three sets of results (a, b, and c) each because the *applied force* input was not specified. A default set of forces is used to showcase the range of motion possibilities of each compliant joint. These Test Cases will determine potential compliant joints for the user requirements, using the default set of forces. Results are returned in sets based on which force in the set was applied. Test Case 6 only returns a single compliant joint for each variation because it is a search for a certain stiffness, which is limited by the geometry. Test case 8 returns all fourteen of the rotational compliant joints because no non-required inputs have been used. This means

the algorithm returns the entire set of compliant joints of the correct type, and orders them by range of motion to display as a text output. The user must further refine the search to gain additional information.

4.4 Selection Algorithm Conclusions

A selection algorithm has been developed that selects compliant joint models from a repository. This selection algorithm can determine applicable compliant joints from any set of user inputs into a GUI. The models are used to determine the range of characteristics resulting from different geometric characteristics used by the models. The results are returned to the user in Pareto curves and a text list, so that the user can make a final decision on the compliant joint that most satisfactorily fulfills their requirements.

The selection algorithm was determined to perform adequately when given various test scenarios with random initial inputs. All possible input conditions were tested to verify that each received satisfactory results. No false positive or false negative results were displayed from the input conditions given.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

The research presented has achieved the objective of creating a repository of compliant joints, which are characterized by parametric equations and geometric characteristics. The first research question of this thesis was “What parameters can a compliant joint be characterized by such that it can be objectively compared to other compliant joints?” The compliant joints within the repository were normalized through a process which adjusted the parametric equations to utilize the same inputs across all joints of the same type, and to produce outputs of stiffness and range of motion of the same units across all models. The second research question asks “How can a user achieve results that include a compliant joint that most closely fulfills their requirements?” The repository facilitates selection of compliant joints through a selection algorithm, and presents results to the user. The results are based upon three required user inputs, and the option of three non-required user inputs. The selection algorithm will produce results with any combination of non-required inputs. The final research question poses “How can the information (results) be presented to the user so that it is possible to differentiate between multiple satisfactory solutions?” The results include Pareto curves, which relate the potential range of motion to the stiffness of the compliant joint, over all possible geometries available from the user input. This allows the user to compare compliant joints that fulfill their requirements side by side on additional performance metrics, to determine objectively which compliant joint has the preferred qualities for the given requirements.

All models within the repository were validated through FEA comparisons of deflection. The FEA models show that the parametric models of each compliant joint were characterizing the movement of the compliant joints correctly. Ten test cases composed of input values within a range were applied to the selection algorithm. The algorithm generated results to all test cases, showing that all solutions sets only contained true positives, with no false positives or false negatives provided.

5.2 Potential Impact

The research presented has some potential impacts that should be highlighted. Due to the extensible design of the parametric model repository, it is possible for new users to follow the procedure outlined, and add additional models of compliant joints to the repository. This open ended work would increase the versatility of the selection algorithm by allowing more potential compliant joints to be highlighted.

Another potential impact would be available if the repository and selection algorithm was integrated with Computer-Aided Design (CAD) software. Since the selection algorithm determines which compliant joints will fulfill user requirements over certain geometries, this information could be exported to use within solid-body models. From this, additional FEA testing could be performed, or the files could be converted into .STL files for use in additive manufacturing. Fused deposition modeling (or 3D printers) could be used to immediately print design solutions. Compliant joints are especially appropriate to be made using 3D printing due to their ability to be made as a singular piece. In this way, an inexperienced user could go from a design problem requiring a compliant joint to a functional, physical compliant joint in a matter of hours.

5.3 Future Work

A number of additional problems could be addressed to increase the robustness and applicability of the method outlined in this thesis. First and foremost would be the addition of more compliant joints of each type, as well as potential combinations of the flexures listed here. More compliant joints in the repository would lead to a greater variety of selection, which could lead to a more satisfactory solution for the user. Combinations of flexures and compliant joints would open up a new solution space, with unique design combinations. This would give the repository some ability to create new compliant joints.

The robustness of the current algorithm should be addressed as well. A number of additional options could be added to the material selection, as well as the types of compliant joints to select from. Allowing users to interact with the algorithm itself and change the geometric parameters that are explored over each joint would add a large amount of customizability, which power users may desire.

The value of the algorithm would also increase if additional characteristics were explored to further define individual joints. Precision, especially for translational compliant joints, can be a more desired trait than range of motion or stiffness. The same could be said for axial drift control on rotational compliant joints. Additionally, mass of individual compliant joints could be used as a performance characteristic, which would allow for designs that wish to minimize material or weight for a final design.

Meta-characteristics of the compliant joints, such as manufacturability, would inform the user about potential downfalls or limitations of the compliant joints within the

repository. Since an assumption had to be made for the thickness of the joints to achieve good ranges of motion, this assumption could be relaxed on a case by case basis based on the potential manufacturability of the parts. This could accommodate for new design or manufacture methods that were not developed at the time of this research. A visual representation of all results (the CAD renderings) would also help the user understand the context of the compliant joints and additional quantitative information about the construction and design of the joints.

Additional parallel testing of the current algorithm could lead to further streamlining. Running parallel with topological optimization methods or pseudo-rigid-body design of compliant mechanisms would quantify the potential of this research as a selection method for use in design, and see if all methods return the same final result.

CHAPTER 6: REFERENCES

- [1] L. L. Howell, *Compliant Mechanisms*. 2001.
- [2] S. Shuib, M. I. Z. Ridzwan, and A. H. Kadarman, “Methodology of Compliant Mechanisms and its Current Developments in Applications : A Review,” *Am. J. Appl. Sci.*, vol. 4, no. 3, pp. 160–167, 2007.
- [3] N. Lobontiu, *Compliant Mechanisms: Design of Flexure Hinges*. 2002.
- [4] S. Zelenika and F. De Bona, “Design of Microsystems Based on Compliant Structures and Devices,” in *International Design Conference*, 2006, pp. 1033–1040.
- [5] B. P. Trease, Y.-M. Moon, and S. Kota, “Design of Large-Displacement Compliant Joints,” *J. Mech. Des.*, vol. 127, no. July, pp. 788–798, 2005.
- [6] L. L. Howell, “A generalized loop-closure theory for the analysis and synthesis of compliant mechanisms,” 1993.
- [7] L. L. Howell and A. Midha, “Parametric Deflection Approximations for End-Loaded , Large-Deflection Beams in Compliant Mechanisms,” *Trans. ASME*, vol. 117, no. March, pp. 156–165, 1995.
- [8] B. T. Edwards, B. D. Jensen, and L. L. Howell, “A Pseudo-Rigid-Body Model for Functionally Binary Pinned-Pinned Segments Used in Compliant Mechanisms,” in *Proceedings of the 1999 ASME Design Engineering Technical Conferences*, 1999, pp. 1–12.
- [9] D. A. Espinosa, “Moment-dependent pseudo-rigid-body models for beam deflection and stiffness kinematics and elasticity,” 2009.
- [10] X. Tang, I.-M. Chen, and Q. Li, “Design and nonlinear modeling of a large-displacement XYZ flexure parallel mechanism with decoupled kinematic structure,” *Rev. Sci. Instrum.*, vol. 77, no. 11, p. 115101, 2006.
- [11] M. Frecker, G. K. Ananthasuresh, S. Nishiwaki, S. Kota, and N. Kikuchi, “Topological Synthesis of Compliant Mechanisms Using Multi-Criteria Optimization,” *J. Mech. Des.*, vol. 119, no. June, pp. 238–245, 1997.
- [12] M. Frecker, N. Kikuchi, and S. Kota, “Topology optimization of compliant mechanisms with multiple outputs,” *Struct. Optim.*, no. 17, pp. 269–278, 1999.

- [13] K. Lu and S. Kota, “Design of Compliant Mechanisms for Morphing Structural Shapes,” *J. Intell. Mater. Syst. Struct.*, vol. 14, no. June, pp. 379–391, 2003.
- [14] O. Sigmund, “On the Design of Compliant Mechanisms Using Topology Optimization,” *Mech. Struct. Mach. An Int. J.*, vol. 25, no. 4, pp. 493–524, Jan. 1997.
- [15] C. B. W. Pedersen, T. Buhl, and O. Sigmund, “Topology synthesis of large-displacement compliant mechanisms,” *Int. J. Numer. Methods Eng.*, vol. 50, no. January, pp. 2683–2705, 2001.
- [16] N. D. Pavlović, D. Petković, and N. T. Pavlović, “Optimal selection of the compliant mechanism synthesis method,” in *The International Conference: Mechanical Engineering in XXI Century*, 2010, pp. 1–4.
- [17] P. Bernardoni, P. Bidard, C. Bidard, and F. Gosselin, “A new compliant mechanism design methodology based on flexible building blocks,” 2004.
- [18] N. Jhavar and G. K. Ananthasuresh, “Cataloguing and Selection of Displacement-Amplifying Compliant Mechanisms,” in *Research into Design: Supporting Multiple Facets of Product Development*, 2009, pp. 26–34.
- [19] S. Mahler, “Compliant pediatric prosthetic knee,” University of South Florida, 2007.
- [20] A. Kyusojin, D. Sagawa, and A. Toyama, “Development of linear and rotary movement mechanism by using flexible strips,” *Bull. Japan Soc. Precis. Eng.*, 1988.
- [21] X. Pei, J. Yu, G. Zong, and S. Bi, “Design of compliant straight-line mechanisms using flexural joints,” *Chinese J. Mech. Eng.*, vol. 27, no. 1, pp. 146–153, Feb. 2014.
- [22] Y. Tian, B. Shirinzadeh, and D. Zhang, “Closed-form compliance equations of filleted V-shaped flexure hinges for compliant mechanism design,” *Precis. Eng.*, vol. 34, no. 3, pp. 408–418, Jul. 2010.
- [23] M. Goldfarb and J. E. Speich, “A well-behaved revolute flexure joint for compliant mechanism design,” *Trans. ASME*, vol. 121, no. September, pp. 424–429, 1999.
- [24] A. E. Guérinot, S. P. Magleby, L. L. Howell, and R. H. Todd, “Compliant Joint Design Principles for High Compressive Load Situations,” *J. Mech. Des.*, vol. 127, no. 4, pp. 774–781, 2005.

- [25] S. Smith, *Flexures, Elements of Elastic Mechanisms*. 2002.
- [26] B. D. Jensen and L. L. Howell, “The modeling of cross-axis flexural pivots,” *Mech. Mach. Theory*, vol. 37, no. 5, pp. 461–476, May 2002.
- [27] Q. Xu and Y. Li, “A novel design of a 3-PRC translational compliant parallel micromanipulator for nanomanipulation,” *Robotica*, vol. 24, no. 04, p. 527, Jan. 2006.
- [28] A. Kyusojin, D. Sagawa, and A. Toyama, “Development of Linear and Rotary Movement Mechanisms by Using Leaf Springs,” *J. Jpn. Soc. Prec. Eng.*, vol. 4, no. 22, pp. 1092–1096, 1988.
- [29] G. Mocko, R. J. Malak, C. J. J. Paredis, and R. Peak, “A Knowledge Repository for Behavioral Models in Engineering Design,” in *Proceedings of DETC '04*, 2004, pp. 1–10.

APPENDICES

APPENDIX A. COMPLIANT JOINT REPOSITORY

Located in this appendix are all of the compliant joints that were used in the compliant joint repository. Each individual compliant joint section will contain the compliant joint's name, its geometric parameters, additional notes, a 3-D CAD solid model representation, the original equations found in the literature describing its motion, and the MATLAB function parametric model used in the selection algorithm.

A.1 Translational Compliant Joints

1. Name: Smith Rectilinear

Geometric Parameters: w (thickness), d (depth), L (length), D (distance between leaf springs)
leaf springs)

Additional Notes:

Accuracy increases as $L \gg w$

Same as Kyusojin Parallel Strip – calculated differently

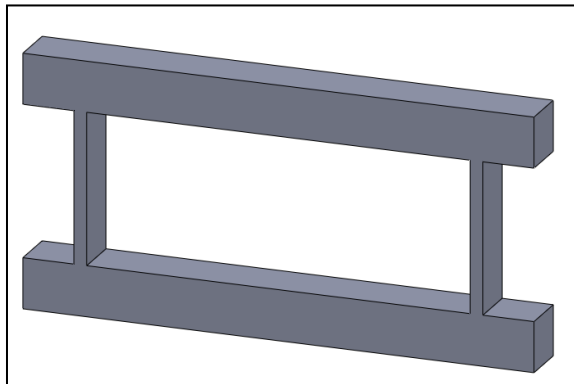


Figure A 1. Solid model representation of the Smith Rectilinear compliant joint.

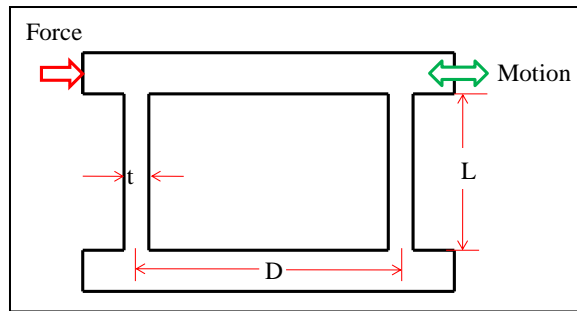


Figure A 2. Geometric characteristics of the Smith Rectilinear compliant joint.

The linear stiffness of the compliant joint is described by

$$k = 2Eb \left(\frac{t}{L} \right)^3,$$

where E is the Young's Modulus of the material, b is the depth of the material, L is the height of the leaf springs, and t is the thickness of the leaf springs.

```
function [ Disp,K ] = SmithRectilinearSpringFun( E,t,b,L,F )
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----%%
% w = thickness of leaf springs
%mm
% d = Depth of joint
% L = Height of leaf springs
% D = Distance between leaf springs
% F = Force Applied

%%----Solution of Joint----%%
%Stiffness
K=2*E*b*(t/L)^3;
%In plane displacement
Disp=F/K;
end
```

Figure A 3. MATLAB function model of the Smith Rectilinear compliant joint.

2. Name: Kyusojin Parallel Strip

Geometric Parameters: w (thickness), h (depth), L (length)

Additional Notes:

Accuracy increases as $L \gg w$

Same as Smith Rectilinear – calculated differently

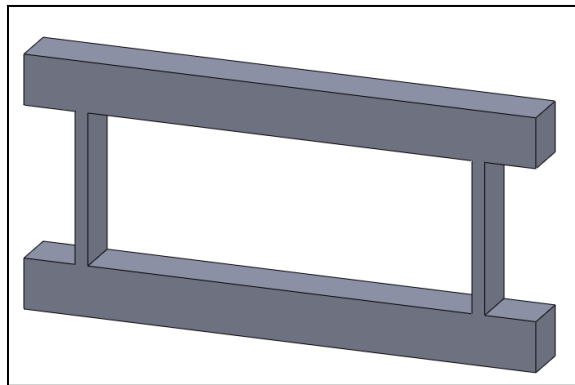


Figure A 4. Solid model representation of the Kyusojin Parallel Strip compliant joint.

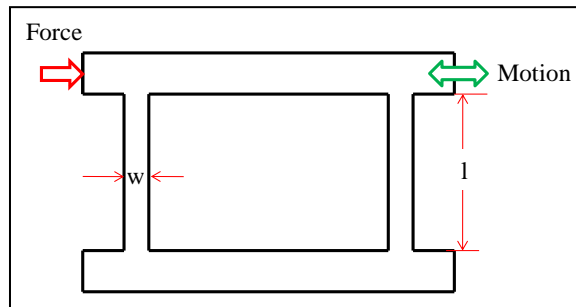


Figure A 5 Geometric characteristics of the Kyusojin Parallel Strip compliant joint.

The displacement of the upper table in the x-direction can be described by

$$x = \sqrt{\frac{5F^2 l^6}{3(240(EI)^2)'}}$$

where F is the applied force, l is the height of the leaf springs, E is the Young's modulus of the material, and I is the moment of inertia of the leaf springs about their center.

```
function [ DispX,K ] = ParallelStripFun( E,h,w,l,F )
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----%%
% w = thickness of leaf springs
%mm
% h = Depth of joint
% l = Height of leaf springs
% F = Force Applied

%%----Solution of Joint----%%
w=w/5;
I=h*w^3/12;
%In plane deflection
DispX=sqrt((5*F^2*l^6)/(3*(240*(E*I)^2)));

K=F/DispX;
%Only represents one spring, so
DispX=DispX/2;
K=K*2;
%Out of plane deflection
Dispy=F^2*l^5/(240*(E*I)^2);

end
```

Figure A 6. MATLAB function model of the Kyusojin Parallel Strip compliant joint.

3. Name: Kyusojin Linear 6L1

Geometric Parameters: t (thickness), w (width of leaf spring), l (length of leaf spring)

Additional Notes: Center table length is irrelevant

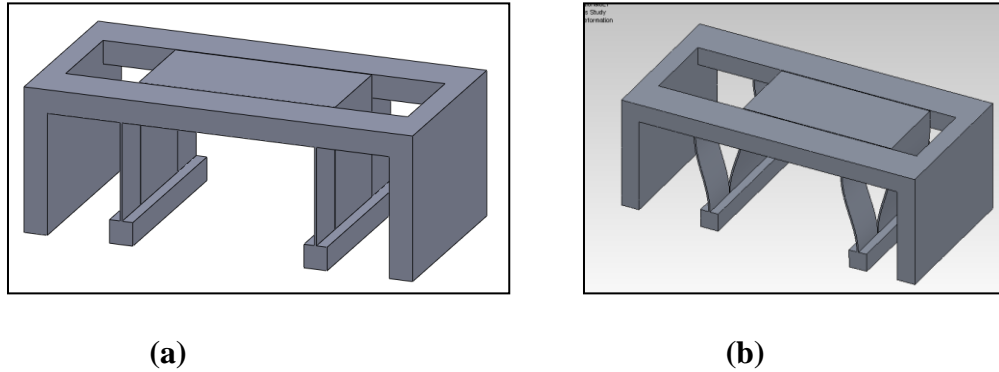


Figure A 7. Solid model representation of the Kyusojin Linear 6L1 compliant joint, (a) original position and (b) deflected position.

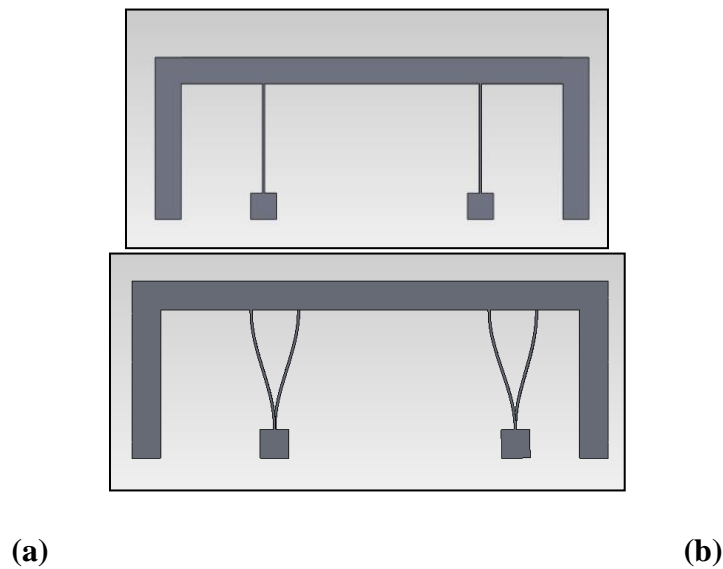


Figure A 8. Side view of solid model representation of the Kyusojin Linear 6L1 compliant joint, (a) original position and (b) deflected position.

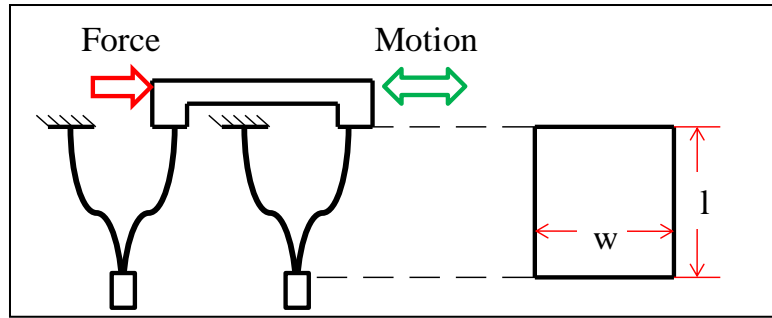


Figure A 9. Geometric characteristics of the Kyusojin Linear 6L1 compliant joint.

The displacement of the upper table in the x-direction can be described by

$$x = \sqrt{\frac{5F^2l^5}{60(EI)^2}}$$

where F is the force applied, l is the height of the leaf springs, E is the Young's Modulus of the material, and I is the moment of inertia of the leaf springs about their center.


```

function [ Disp,K ] = KyusojinLinear6L1Fun( E,t,w,l,F )
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----%%
% w = Width of leaf springs
%mm
% t = Thickness of leaf springs
% l = Height of leaf springs
% F = Force Applied

%Number of spring-pairs
N=4;

%%----Solution of Joint----%%
t=t/5;
I=(t*w^3)/12;
a=sqrt(N/(E*I));

%Out of plane deflection
dely=(F^2*l^5)/(60*(E*I)^2)*(1/(cos(a*l))-1);

%In plane deflection (linear translation)
Disp=sqrt((5*F^2*l^6)/(3*(60*(E*I)^2)));
%Needed to fix scale
Disp=Disp*1000;
K=F/Disp;
end

```

Figure A 10. MATLAB function model of the Kyusojin Linear 6L1 compliant joint.

4. Name: Trease Translational

Geometric Parameters: t (thickness), w (depth), L_B (length)

Additional Notes: This is the planar configuration – this joint also has a spatial configuration with two more sets of leaf springs on the other non translational axis

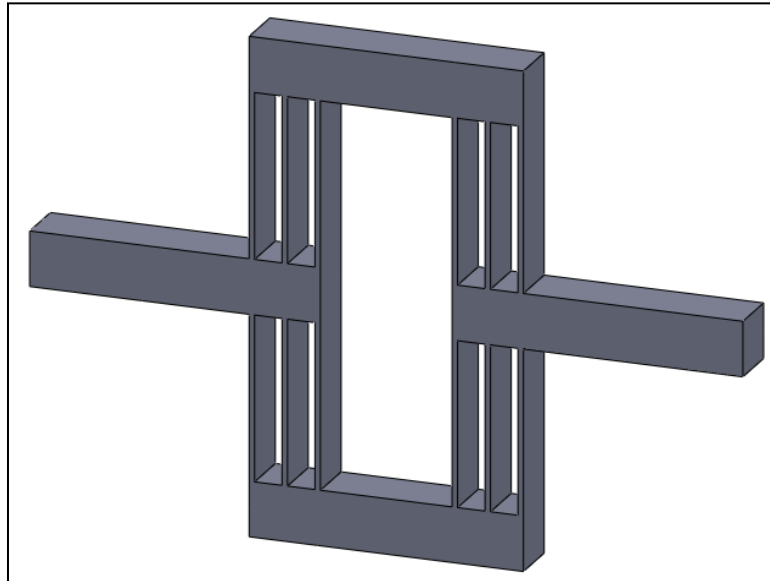


Figure A 11. Solid model representation of the Trease Translational compliant joint.

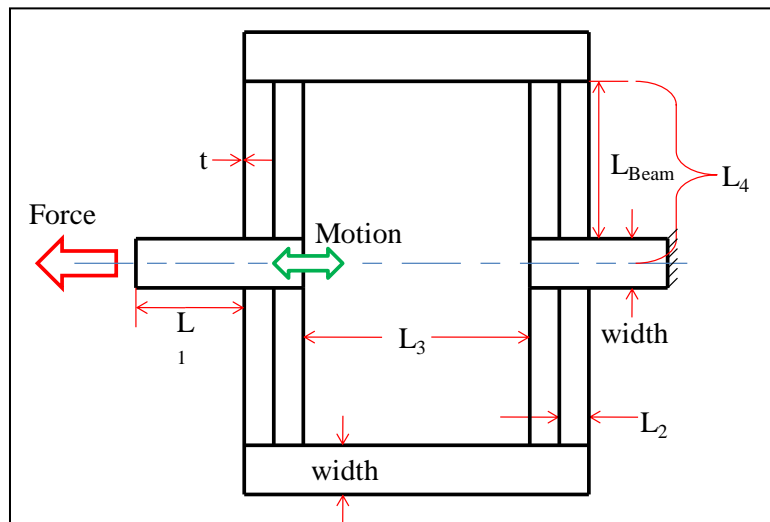


Figure A 12. Geometric characteristics of the Trease Translational compliant joint.

The linear stiffness of the compliant joint can be described by

$$k = \frac{3(Et^3w)}{LB^3},$$

where E is the Young's Modulus of the material, t is the thickness of the leaf springs, w is the depth of the leaf springs, and LB is the length of the leaf springs.

```
function [ Disp,kaxialp ] = TreaseTranslationalFun( E,t,w,LB,F )
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----%%
% w = Depth of leaf springs
%mm
% t = Thickness of leaf springs
% LB = Length of leaf springs
% F = Force Applied

%%----Solution of Joint----%%
t=t/5;
%Note: Planar is 2 sets of 6 beams, spatial is 4 sets of 6 beams
%%Only planar is used as a joint in the solution
%K for planar (2 sides)
kaxialp = 3*(E*t^3*w)/(LB^3);
%K for spatial (4 sides)
kaxials = 6*(E*t^3*w)/(LB^3);
%Displacement
Disp= F/kaxialp;

end
```

Figure A 13. MATLAB function model of the Trease Translational compliant joint.

5. Name: Xu Translational

Geometric Parameters: t (thickness), ϕ (angle), H_p (total length), H (length to fixed ends), h_f (length to force applied)

Additional Notes: Very precise for small translational distances

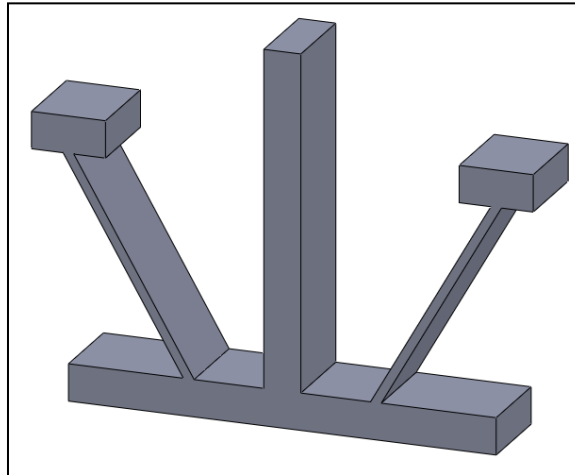


Figure A 14. Solid model representation of the Xu Translational compliant joint.

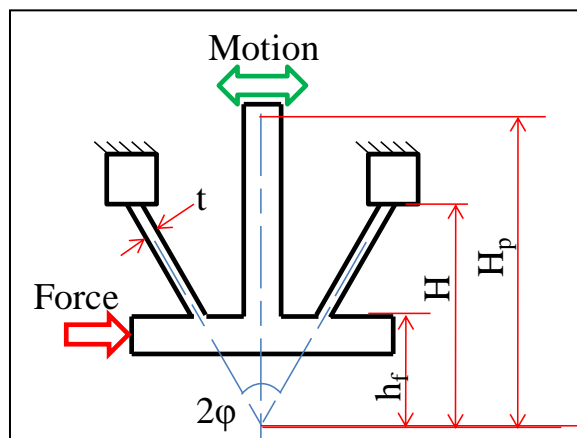


Figure A 15. Geometric characteristics of the Xu Translational compliant joint.

The linear displacement of this compliant joint can be described by

$$x = H_p * \sin(\theta),$$

where H_p is the height to the movement point from the origin point, and θ is the angle of movement of the center bar. This angle can be found from

$$\theta = \frac{H * d_c}{E * t * (3n - 1) * n * \cos(\varphi)}$$

where H is the height from the origin to the fixed locations, d_c is the depth of the compliant joint, E is the Young's Modulus of the material, t is the thickness of the leaf springs, φ is the angle of the leaf springs, and n is a geometric relation. This relation is described by

$$n = \frac{H}{(H - h_f)}$$

where h_f is the height to the platform where the force is applied. Additionally, H_p can be calculated using

$$H_p = \frac{n - \gamma}{\gamma * \cos^2(\varphi)} H,$$

where γ is a geometric relation described by

$$\gamma = \frac{15 * n^2}{(2 - 3n + 18n^2)}$$

```

function [ dmax,K ] = XuTranslationalFun( E,t,dc,hf,H,F )
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----%%
% t = Thickness of leaf springs
%mm
% phi = Angle (from vertical) to each fixed point,
%%set to simplify optimization
phi = 30;
% Hf = Point to force application
% H = Height to fixed points
% F = Force Applied

%%----Solution of Joint----%%
t=t/5;

n=H/(H-hf);
gamma =15*n^2/(2-3*n+18*n^2);
Hp=(n-gamma)/(gamma*cosd(phi)^2)*H;
theta = (H*dc)/(E*t*(3*n-1)*n*cosd(phi));
dx=Hp*theta;

%Deflection in x direction
dmax=Hp*sin(theta);
K=F/dmax;

end

```

Figure A 16. MATLAB function model of the Xu Translational compliant joint.

6. Name: Smith Notch Hinge

Geometric Parameters: t (thickness), R (radius of circles), L* (distance between flexures)
flexures)

Additional Notes: More accurate when $R \gg L$ and $t \gg L$

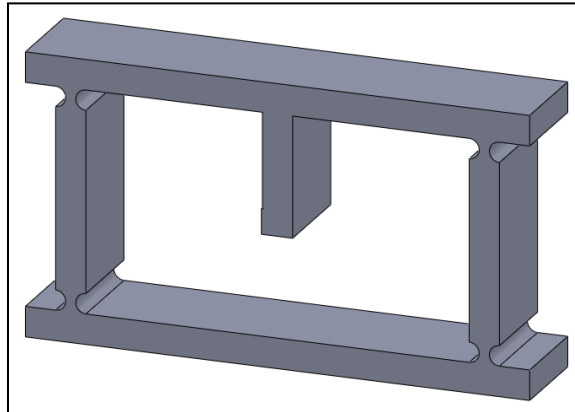


Figure A 17. Solid model representation of the Smith Notch Hinge compliant joint.

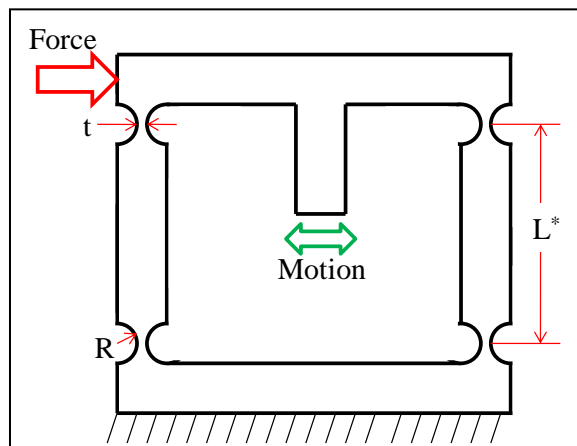


Figure A 18. Geometric characteristics for the Smith Notch Hinge compliant joint.

The linear stiffness of this compliant joint can be described by

$$K = \frac{8Ebt^{\frac{5}{2}}}{9\pi R^{\frac{1}{2}}L^{*2}}$$

where E is the Young's Modulus of the material, b is the depth of the compliant joint, t is the thickness of the circular flexures, R is the radius of the circular flexures, and L^* is the distance between the two circular flexures' centers.

```
function [ Disp,K ] = SmithNotchHingeFun( E,R,t,b,L,F )
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----%%
% R = Radius of circular notches
%mm
% t = Thickness between circular notches
% b = Depth of Flexures
% L = Length between circular notches
% F = Force Applied

%%----Solution of Joint----%%
R=R/2;
t=t/5;
Lstar=L+2*R;
%Stiffness calculation
K=(8*E*b*t^(5/2))/(9*pi()*R^.5*Lstar^2);
%Deflection
Disp=F/K;

end
```

Figure A 19. MATLAB function model of the Smith Notch Hinge compliant joint.

A.2 Rotational Compliant Joints

1. **Name:** Lobontiu Symmetric Notch

Geometric Parameters: t (thickness), w (depth), l (length)

Additional Notes:

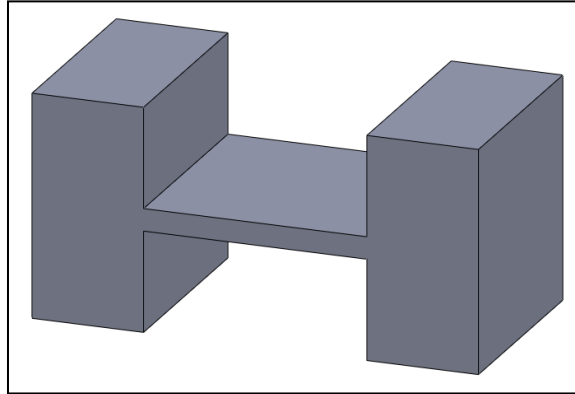


Figure A 20. Solid model representation of the Lobontiu Symmetric Notch compliant joint.

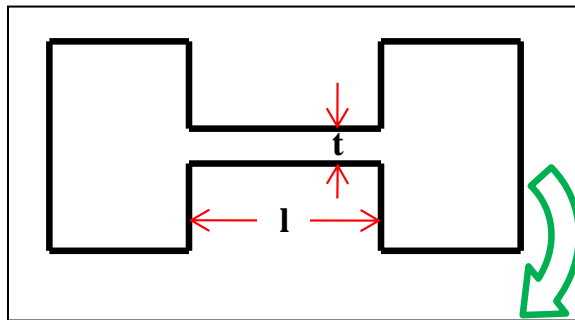


Figure A 21. Geometric characteristics of the Lobontiu Symmetric Notch compliant joint.

The linear compliance of this compliant joint is defined as

$$C = \frac{4l^3}{Ewt^3},$$

where E is the Young's modulus of the material, l is the length of the flexure, w is the depth of the flexure, and t is thickness of the flexure.

```
function [Disp,K] =
LobontiuSymmetricNotchFun(E,t,l,w,F)
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional
Characteristics----
% t = Thickness of flexure
% w = Depth of flexure
% l = Length of flexure
% F = Force applied

%%----Solution of Joint----%%
t=t/5;

%Compliance
C = (4*l^3)/(E*w*t^3);
%Stiffness
K=1/C;
%Linear Displacement
Disp = F/K;
%Angular Displacement
Disp = asin(Disp/l)*180/pi();
%Rotational Stiffness
K=(F*l/2)/Disp;

end
```

Figure A 22. MATLAB function model of the Lobontiu Symmetric Notch compliant joint.

2. Name: Lobontiu Corner Filleted

Geometric Parameters: r (radius of fillets), t (thickness), w (depth), l (length)

Additional Notes:

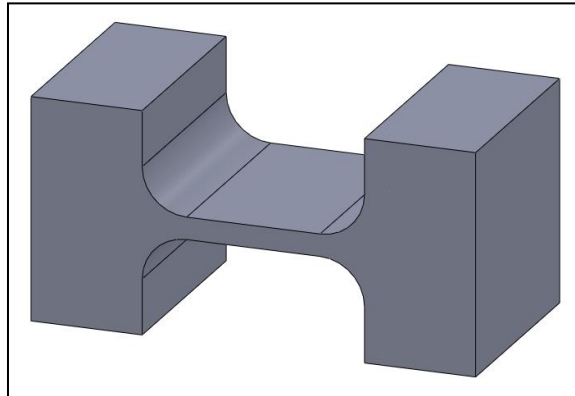


Figure A 23. Solid model representation of the Lobontiu Corner Filleted compliant joint.

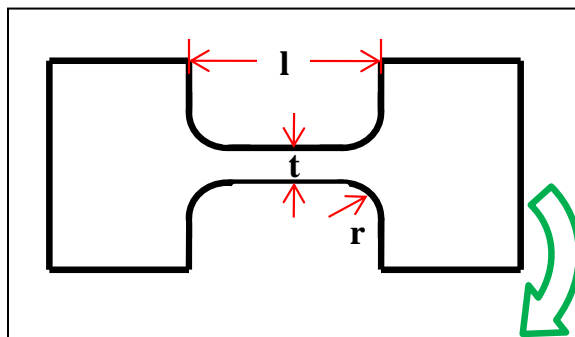


Figure A 24. Geometric characteristics of the Lobontiu Corner Filleted compliant joint.

The linear compliance of the flexure is defined as

$$\begin{aligned}
& C_{1,y-F_y} \\
&= \frac{3}{Ew} \left[\frac{4(l-2r)(l^2-lr+r^2)}{3t^3} \right. \\
&+ \frac{\sqrt{t(4r+t)}[-80r^4+24r^3t+8(3+2\pi)r^2t^2+4(l+2\pi)rt^3+\pi^4]}{4\sqrt{t^5(4r+t)^5}} \\
&+ \frac{(2r+t)^3(6r^2-4rt-t^2)\arctan\sqrt{1+\frac{4r}{t}}}{\sqrt{t^5(4r+t)^5}} \\
&+ \frac{-40r^4+8lr^2(2r-t)+12r^3t+4(3+2\pi)r^2t^2+2(1+2\pi)rt^3+\frac{\pi t^4}{2}}{2t^2(4r+t)^2} \\
&+ \frac{4l^2r(6r^2+4rt+t^2)}{t^2(2r+t)(4r+t)^2} \\
&- \left. \frac{(2r+t)[-24(l-r)^2r^2-8r^3t+14r^2t^2+8rt^3+t^4]}{\sqrt{t^5(4r+t)^5}} \arctan\sqrt{1+\frac{4r}{t}} \right],
\end{aligned}$$

where E is the Young's modulus of the material, w is the depth of the flexure, l is the length of the flexure, r is the radius of the fillets, and t is the thickness of the flexure.

```

function [ Disp,K ] = LobontiuCornerFillettedFun( E,t,r,l,w,F )
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----
% r = Radius of filets
%%mm
% w = Depth of flexure
% l = Length of flexure
% t = Thickness of flexure
% F = Force applied

%%----Solution of Joint----%%
r=r/2;
t=t/5;
%Compliance
C=(3/(E*w))*(((4*(1-2*r)*(1^2-1*r+r^2))/(3*t^3))...
+(sqrt(t*(4*r+t))*(-
80*r^4+24*r^3*t+8*(3+2*pi())*r^2*t^2+4*(1+2*pi())*r*t^3+pi()*t^4))/(4*s
qrt(t^5*(4*r+t)^5))...
+((2*r+t)^3*(6*r^2-4*r*t-
t^2)*atan(sqrt(1+4*r/t)))/sqrt(t^5*(4*r+t)^5)...
+(-40*r^4+8*1*r^2*(2*r-
t)+12*r^3*t+4*(3+2*pi())*r^2*t^2+2*(1+2*pi())*r*t^3+(pi()*t^4)/2)/(2*t^
2*(4*r+t)^2)...
+(4*1^2*r*(6*r^2+4*r*t+t^2))/(t^2*(2*r+t)*(4*r+t)^2)...
-((2*r+t)*(-24*(1-r)^2*r^2-
8*r^3*t+14*r^2*t^2+8*r*t^3+t^4)/(t^5*(4*r+t)^5)^0.5)*atan(1+4*r/t));
%Stiffness
K=1/C;
%Displacement
Disp=C*F;
%Angular Displacement
Disp = asin(Disp/l)*180/pi();
%Rotational Stiffness
K=(F*l/2)/Disp;

end

```

Figure A 25. MATLAB function model of the Lobontiu Corner Filletted compliant joint.

3. Name: Lobontiu Symmetric Circular

Geometric Parameters: r (radius of circles), t (thickness), w (depth), l (length)

Additional Notes: Same as Tang Symmetric Circle, but calculated differently

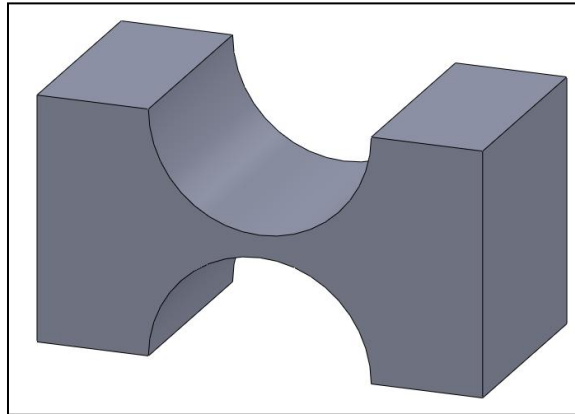


Figure A 26. Solid model representation of the Lobontiu Symmetric Circle compliant joint.

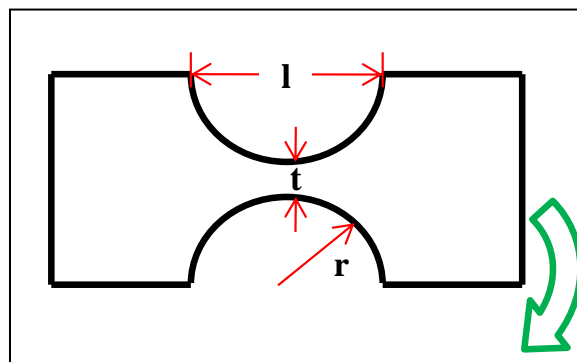


Figure A 27. Geometric characteristics of the Lobontiu Symmetric Circle compliant joint.

The compliance of the flexure is given by

$$\begin{aligned}
& C_{1,y-F_y} \\
&= \frac{3}{4Ew(2r+t)} \left\{ 2(2+\pi)r + \pi t + \frac{8r^3(44r^2 + 28rt + 5t^2)}{t^2(4r+t)^2} \right. \\
&+ \frac{(2r+t)\sqrt{t(4r+t)}[-80r^4 + 24r^3t + 8(3+2\pi)r^2t^2 + 4(1+2\pi)rt^3 + \pi t^4]}{\sqrt{t^5(4r+t)^5}} \\
&\left. - \frac{8(2r+t)^4(-6r^2 + 4rt + t^2)}{\sqrt{t^5(4r+t)^5}} \arctan \sqrt{1 + \frac{4r}{t}} \right\},
\end{aligned}$$

where E is the Young's modulus of the material, r is the radius of the flexure, l is the length of the flexure, and t is the thickness of the flexure.

```

function [ Disp,K ] = LobontiuSymmetricCircularFun( E,t,r,w,F )
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----
% r = Radius of circles
r=r/2;
%%mm
% t = Distance between circles
t=t/5;
% w = Depth of Flexure
% F = Force applied

%%----Solution of Joint----%%
%Compliance
C = (3/(4*E*w*(2*r+t)))*...

(2*(2+pi())*r+pi()*t+(8*r^3*(44*r^2+28*r*t+5*t^2))/(t^2*(4*r+t)^2)...
+((2*r+t)*sqrt(t*(4*r+t))*(-
80*r^4+24*r^3*t+8*(3+2*pi())*r^2*t^2+4*(1+2*pi())*r*t^3+pi()*t^4))/sqrt
(t^5*(4*r+t)^5)...
-((8*(2*r+t)^4*(-
6*r^2+4*r*t+t^2))/sqrt(t^5*(4*r+t)^5))*atan((1+(4*r)/t)^0.5));
% C=24*r^2/(E*w*t^3*(2*r+t)*(4*r+t)^3)*...
% (t*(4*r+t)*(6*r^2+4*r*t+t^2)...
% +6*r*(2*r+t)^2*sqrt(t*(4*r+t))*atan(sqrt(1+4*r/t)));
%Stiffness
K=1/C;
%Linear Displacement
Disp = F/K;
%Angular Displacement
Disp = asin(Disp/(r))*180/pi();
%Rotational Stiffness
K=(F*r)/Disp;

end

```

Figure A 28. MATLAB function model of the Lobontiu Symmetric Circle compliant joint.

4. Name: Tian V Shape Flexure

Geometric Parameters: R (radius of circles), t (thickness), b (depth), 2l (length), c (height of slope), h (height of flexure), θ (angle of flexure)

Additional Notes:

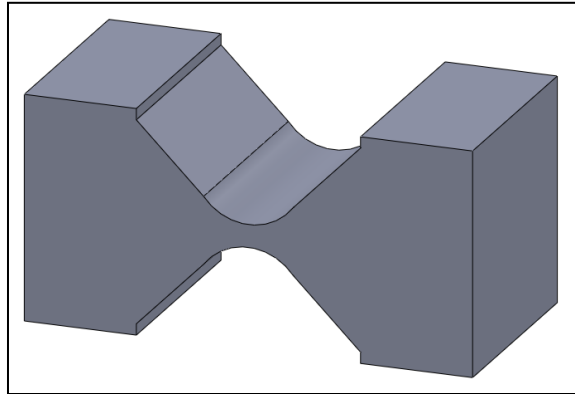


Figure A 29. Solid model representation of the Tian V Shape Flexure compliant joint.

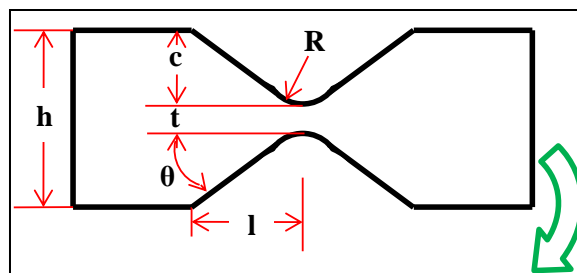


Figure A 30. Geometric characteristics of the Tian V Shape Flexure compliant joint.

The deflection of the flexure around its center, α_z , for a given moment, M_z ,

$$\frac{\alpha_z}{M_z} = \frac{3}{2EbR^2} \left\{ \frac{1}{2\beta + \beta^2} \left[\frac{(1 + \beta) \sin \theta}{(1 + \beta - \cos \theta)^2} + \frac{(3 + 2\beta + \beta^2) \sin \theta}{(2\beta + \beta^2)(1 + \beta - \cos \theta)} \right. \right. \\ \left. \left. + \frac{6(1 + \beta)}{(2\beta + \beta^2)^{3/2}} \arctan \left(\sqrt{\frac{2 + \beta}{\beta}} \tan \frac{\theta}{2} \right) \right] - \frac{\gamma^2 \cot \theta}{\beta^2(1 + \gamma)^2} \right. \\ \left. + \frac{\cot \theta}{(1 + \beta - \cos \theta)^2} \right\}$$

where E is the Young's modulus, b is the depth of the flexure, R is the radius of the center circles, θ is the angle of the slopes, and β and γ are geometric relations. B can be described by

$$\beta = \frac{t}{2R},$$

where t is the thickness of the flexure at the center, and R is the radius of the center circles. γ can be described by

$$\gamma = \frac{t}{2c},$$

where t is the thickness of the flexure at the center, and c is the height of the sloped sections.

```
function [alpha,K] = VShapeFun(E,t,h,l,R,b,F)
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----
% t = Thickness of flexure at smallest point
t=t/5;
%%mm
% R = Circles at middle of flexure
%This ensures radius is not larger than half of height of flexure
R=(R/2-t/2)/2;
% l = Half length of flexure
%Since l is half length,
l=l/2;
```

```

% h = Total height of flexure
%theta = Slope of flexure sides
theta = 20;%maintained
theta=theta*2*pi()/360;
% c = Height of sloped portion
%This logic ensures flexure has solution
if h==b
    %min def
    c=h/2-t;
else
    c=0.1;
end
% F = Force applied

%%----Solution of Joint----%%
B=t/(2*R);
y=t/(2*c);
M=F*(l);
%Compliance
C=(3/(2*E*b*R^2))*((1/(2*B+B^2))*((1+B)*sin(theta)/(1+B-
cos(theta))^2)...
    +((3+2*B+B^2)*sin(theta)/((2*B+B^2)*(1+B-cos(theta)))))...
+(6*(1+B)/((2*B+B^2)^(3/2))*atan(sqrt((2+B)/B)*tan(theta/2)))...
-y^2*cot(theta)/(B^2*(1+y)^2)...
+cot(theta)/(1+B-cos(theta))^2);

%Alternative compliance, for force applied rather than moment
% C=(3*y+3*(B-y)*cos(theta))/(2*E*b*R*y*sin(theta))*...
% ((1/(2*B+B^2))*((1+B)*sin(theta))/(1+B-cos(theta))^2)...
% +((3+2*B+B^2)*sin(theta)/((2*B+B^2)*(1+B-cos(theta))))...
%
% +(6*(1+B)/((2*B+B^2)^(3/2))*atan(sqrt((2+B)/B)*tan(theta/2)))...
% -(y^2*cot(theta))/(B^2*(1+y)^2)...
% +cot(theta)/(1+B-cos(theta))^2);
%Stiffness
K=1/C;

%Angular Displacement
alpha=M*C*180/pi();
%Linear Displacement
Disp=sin(alpha)*(l);
%Transform stiffness to Nm/deg
K=K*pi()/180;

end

```

Figure A 31. MATLAB function model of the Tian V Shape Flexure compliant joint.

5. Name: Tang Symmetric Circular

Geometric Parameters: t (thickness), b (depth), R (radius)

Additional Notes: Same as Lobontiu Symmetric Circle, but with a simplified calculation for stiffness.

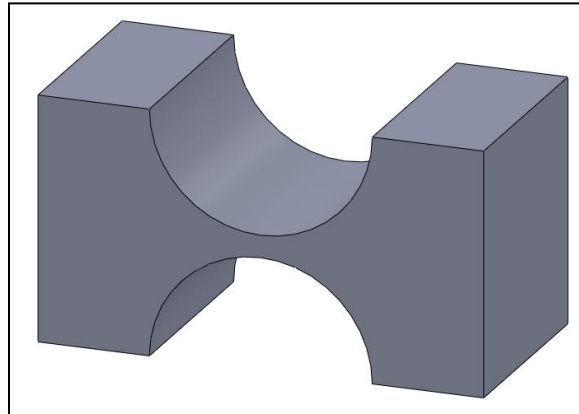


Figure A 32. Solid model representation of the Tang Symmetric Circular compliant joint.

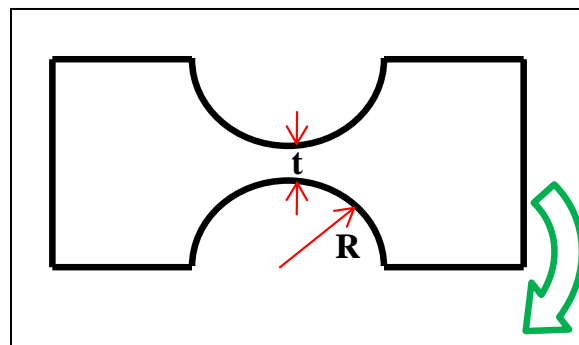


Figure A 33. Geometric characteristics of the Tang Symmetric Circular compliant joint.

The angular stiffness about the center of the flexure is described as

$$K = \frac{2Ebt^{5/2}}{9\pi R^{1/2}},$$

where E is the Young's modulus of the material, b is the depth of the flexure, t is the thickness of the flexure at the center, and R is the radius of the circles.

```
function [Disp,K] = TangSymmetricCircularFun(E,R,t,b,F)
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----
% t = Thickness of flexure at smallest point
t=t/5;
%%mm
% R = Radius of circles
R=R/2;
% b = Depth of Flexure
% F = Force applied

%%----Solution of Joint----%%
M= F*R;
%Stiffness
K=(2*E*b*t^(5/2))/(9*pi()*R^.5);
%Angular Displacement
Disp = M/K*180/pi();
%Linear Displacement
% Disp=sind(Disp)*R;
%Transform stiffness to Nm/deg
K=K*pi()/180;

end
```

Figure A 34. MATLAB function model of the Tang Symmetric Circular compliant joint.

6. Name: Smith Two Axis

Geometric Parameters: r (radius of circles), t (thickness)

Additional Notes:

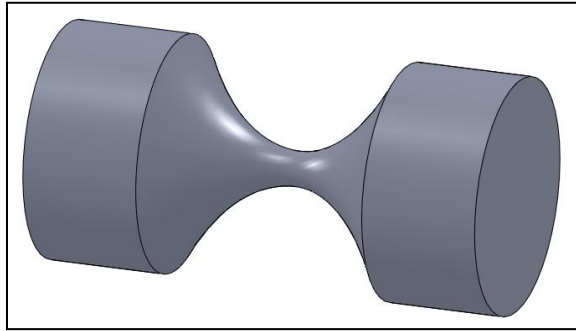


Figure A 35. Solid model representation of the Smith Two Axis compliant joint.

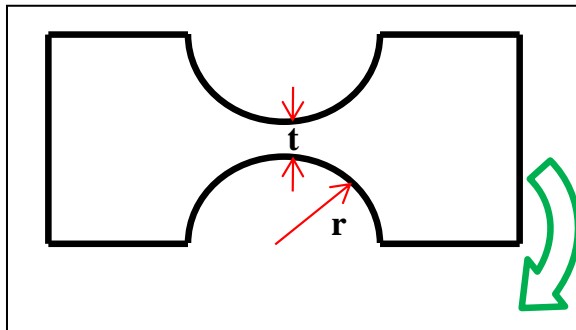


Figure A 36. Geometric characteristics of the Smith Two Axis compliant joint.

The angular stiffness of the flexure can be described by

$$K = \frac{Et^{7/2}}{20R^{3/2}},$$

where E is the Young's Modulus of the material, t is the thickness at the center of the flexure, and R is the radius of the circles.

```

function [Disp,K] = SmithTwoAxisFun(E,R,t,F)
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----
% t = Thickness of flexure at smallest point
t=t/5;
%%mm
% R = Radius of circles
R=R/2;
% F = Applied force

%%----Solution of Joint----%%
%Stiffness
K=(E*t^(7/2))/(20*R^(3/2));
%Angular Displacement for a force applied at the end
Disp=F/K*180/pi();
%Linear Displacement
% Disp=sin(Disptheta)*(R);
%Angular Stiffness
K=F*R/Disp;

end

```

Figure A 37. MATLAB function model of the Smith Two Axis compliant joint.

7. Name: Smith Annulus

Geometric Parameters: t (thickness), r2 (inner radius), r1 (outer radius)

Additional Notes:

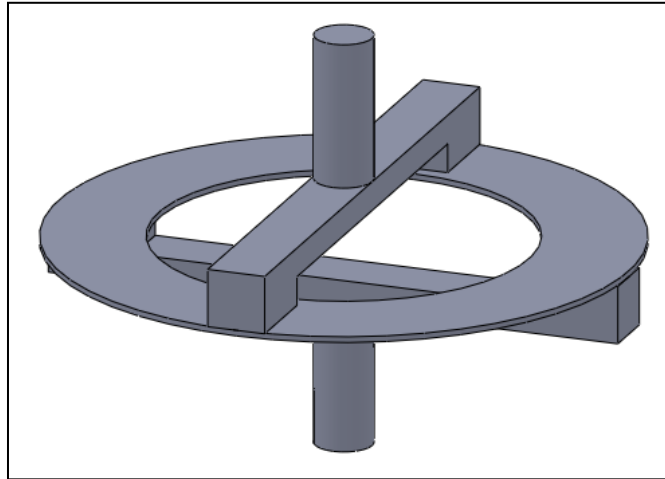


Figure A 38. Solid model representation of the Smith Annulus compliant joint.

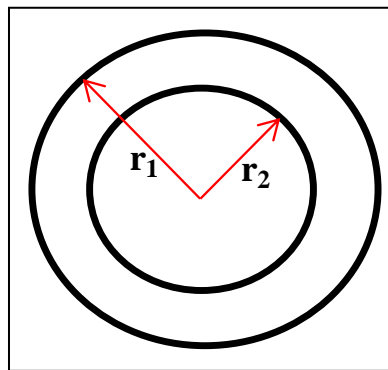


Figure A 39. Geometric characteristics of the Smith Annulus compliant joint.

This compliant joint's angular stiffness can be described by

$$K = \frac{16EI}{r_1} fm,$$

where E is the Young's modulus of the material, I is the moment of inertia of the annulus, r_i is the outer radius of the annulus, and fm is a geometric relation. This relation can be described by

$$fm = \frac{(\pi + 4) + \lambda(8 - \pi)}{(2\pi^2 - 4\pi - 4) + \lambda(8\pi^2 - 18\pi - 16) + \lambda^2(6\pi^2 - 14\pi - 12)},$$

where λ is an additional geometric relation. This is described by

$$\lambda = \frac{EI}{GJ},$$

where E is the Young's modulus of the material, G is the shear modulus of the material, I is the moment of inertia of the annulus, and J is the polar moment of inertia of the annulus.

```

function [ Disp,K ] = SmithAnnulusFun( E,G,r1,t,F )
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----
% t = Thickness of central disk
t=t/5;
%%mm
% r1 = Radius of outer circle
% r2 = Radius of inner circle
r1=r1/2;
r2=r1-(r1*.2);
% F = Applied force

%%----Solution of Joint----%%
M=F*r1;
b=r1-r2;

J=(b*t^3)/3;

I=pi()/4*(r1^4-r2^4);

lamda=E*I/(G*J);
fm1=((pi()+4)+lamda*(8-pi()));
fm2=((2*pi())^2-4*pi()-4)+lamda*(8*pi()^2-18*pi()-16)+lamda^2*(6*pi()^2-
14*pi()-12));
fm=fm1/fm2;
%Stiffness
K=((16*E*I)/r1)*fm;
%Angular displacement
Disp=M/K *180/pi();
%Transform stiffness to Nm/deg
K=K*pi()/180;

end

```

Figure A 40. MATLAB function model of the Smith Annulus compliant joint.

8. Name: Smith Cartwheel

Geometric Parameters: t (thickness), R (radius), b (depth)

Additional Notes:

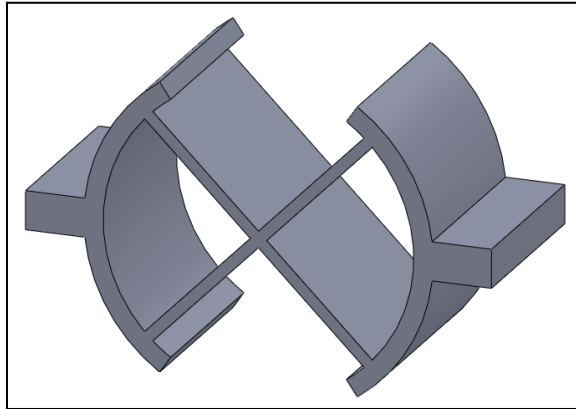


Figure A 41. Solid model representation of the Smith Cartwheel compliant joint.

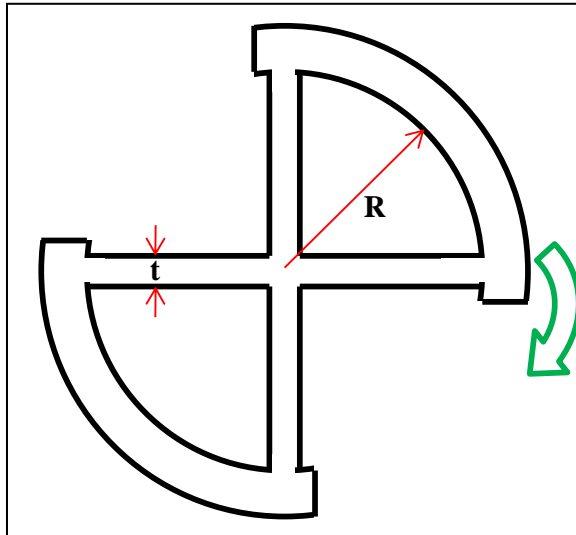


Figure A 42. Geometric characteristics of the Smith Cartwheel compliant joint.

The angular stiffness of the compliant joint can be described by

$$K = \frac{4EI}{R},$$

where E is the Young's Modulus of the material, I is the moment of inertia of the beams, and R is the radius of the circle that contains the flexure.

```
function [thetadeg,K] = SmithCartwheelFun(E,t,R,b,F)
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----
% t = Thickness of flexure beams
t=t/5;
%%mm
% R = Radius of circle
R=R/2;
% b = Depth of flexure
% F = Applied force

%%----Solution of Joint----%%
I=(b*t^3)/12;

%Stiffness
K=4*E*I/R;
M=F*(R);
theta=M/K;
%Angular Displacement
thetadeg=theta*180/pi();
%Linear Displacement
Disp=sin(theta)*(2*R);
%Transform stiffness to Nm/deg
K=K*pi()/180;

end
```

Figure A 43. MATLAB function model of the Smith Cartwheel compliant joint.

9. Name: Smith Cruciform

Geometric Parameters: t (thickness), w (width), l (depth)

Additional Notes:

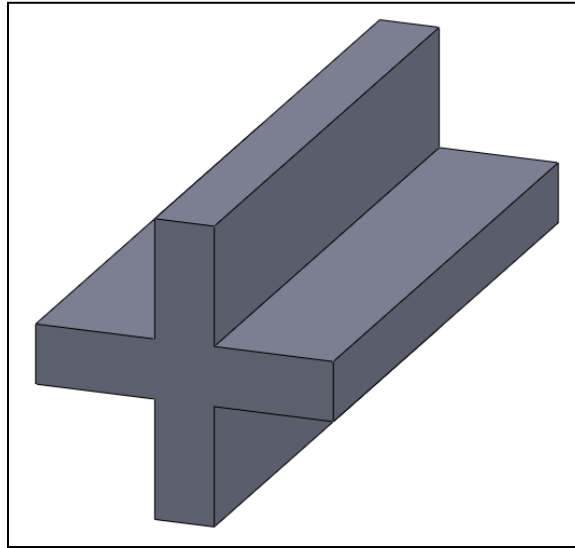


Figure A 44. Solid model representation of the Smith Cruciform compliant joint.

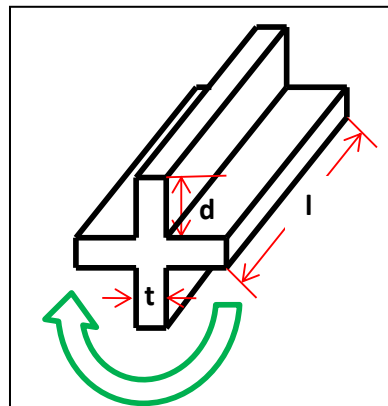


Figure A 45. Geometric characteristics of the Smith Cruciform compliant joint.

The angular stiffness of the compliant joint can be described by

$$K = \left(\frac{d}{t} - 0.373 \right) \frac{2Gt^4}{3L},$$

where d is the height of the cruciform arms, t is the thickness of the cruciform arms, G is the shear modulus of the material, and L is the length of the cruciform.

```
function [theta,K] = SmithCruciformFun(G,t,L,d,F)
% G = Shear Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----
% t = Thickness of cruciform
t=t/5;
%%mm
% l = length of cruciform
% d = height of cruciform arms
d=d/2-t/2;
% F = Applied force

%%----Solution of Joint----%%
M=F*(d);
%Stiffness
K=(d/t-0.373)*(2*G*t^4)/(3*L);
%Angular Displacement
theta=M/K*180/pi();
%Linear Displacement
Disp=sind(theta)*(d);
%Transform stiffness to Nm/deg
K=K*pi()/180;

end
```

Figure A 46. MATLAB function model of the Smith Cruciform compliant joint.

10. Name: Jensen Cross Axis

Geometric Parameters: t (thickness), r (length), L (Length of lever), l (length of cross springs), w (height of flexure), D (depth)

Additional Notes:

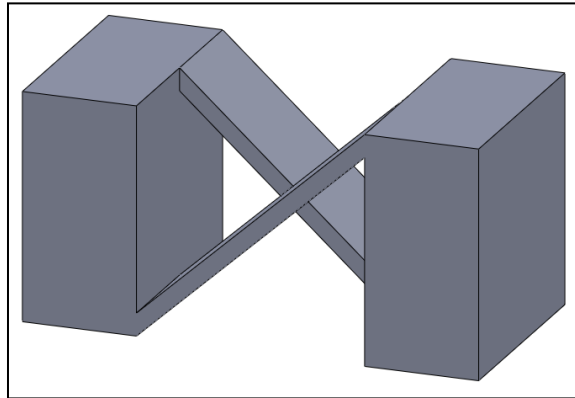


Figure A 47. Solid model representation of the Jensen Cross Axis compliant joint.

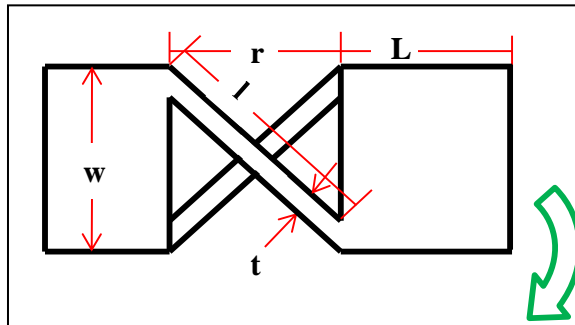


Figure A 48. Geometric characteristics of the Jensen Cross Axis compliant joint.

The angular stiffness of the compliant joint can be described by

$$K = \frac{K_{\theta}EI}{2 * l'}$$

where E is the Young's Modulus, I is the moment of inertia of the leaf springs, l is the length of the leaf springs, and K_{θ} is a polynomial curve fit. This curve fit can be described by

$$K_{\theta} = 5.300185 - 1.6866n + 0.885356n^2 - 0.2094n^3 + 0.018385n^4,$$

where n is the geometric relation

$$n = \frac{r}{w},$$

which is defined by the height of the joint, w and the length of the joint, r.

```
function [Disp,K] = JensenCrossAxisFun( E,t,w,r,D,F)
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----%%
% r = Total length of gap
%mm
% l = Length of cross spring
% w = Height of joint
% t = Thickness of Leaf Spring
% L = Lever arm
% D = Depth of joint
% F = Force Applied

%%----Solution of Joint----%%
L=0.1; %
t=t/5;
x = 1.404*t; %estimation, can be determined geometrically

l=sqrt((w-x)^2+r^2);
n=r/w;

KTheta=5.300185-1.6866*n+0.885356*n^2-0.2094*n^3+0.018385*n^4;
I=(1/12)*D*t^3;
%Stiffness
K=(KTheta*E*I)/(2*l);
M=F*(L+r);
%Angular Displacement
Theta=M/(K);
%Linear Displacement
Disp=sin(Theta)*(L+r);
%Transform stiffness to Nm/deg
K=K*pi()/180;

end
```

Figure A 49. MATLAB function model of the Jensen Cross Axis compliant joint.

11. Name: Smith Rotationally Symmetric Leaf Hinge

Geometric Parameters: d (thickness of cut), R_i (inner radius), R_o (outer radius)

Additional Notes:

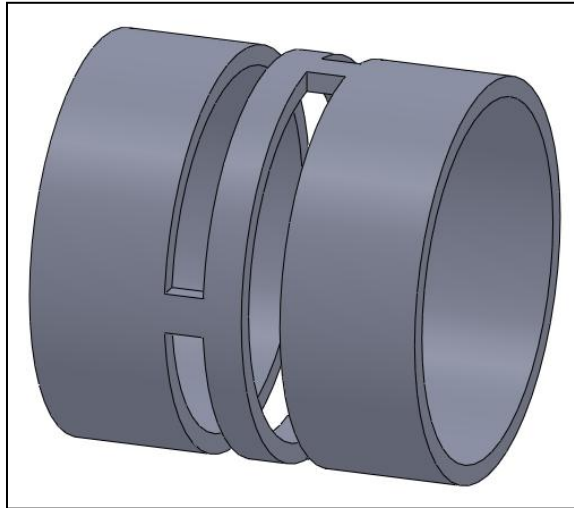


Figure A 50. Solid model representation of the Smith Rotationally Symmetric Leaf Hinge compliant joint.

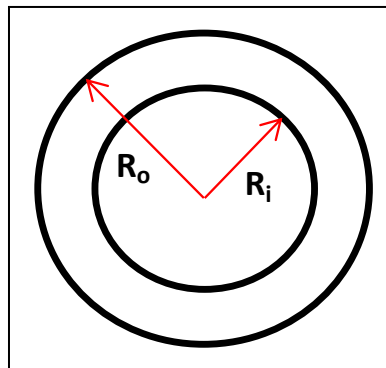


Figure A 51. Geometric characteristics of the Smith Rotationally Symmetric Leaf Hinge compliant joint.

The angular stiffness of this compliant joint can be described by

$$K = \frac{E * d}{2\theta^3 \left(\frac{\varepsilon}{\gamma}\right)^2 (1 - \gamma^2)},$$

where E is the Young's Modulus of the material, d is the thickness of the material between the cuts, θ is the deflection of the center beams, and ε and γ are geometric relations. These relations are described by

$$\gamma = \frac{Ri}{Ro},$$

where Ri is the inner radius and Ro is the outer radius, and

$$\varepsilon = \frac{d}{Ro}.$$

```

function [ Theta,K ] = RotationallySymmetricLeafHingeFun(E,d,Ro,F )
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----
% d = Thickness of flexure at smallest point
d=d/5;
%%mm
% Ro = Outer radius of circle
Ro=Ro/2;
% Ri = Inner radius of circle
Ri=Ro-Ro/5;
% F = Applied force

%%----Solution of Joint----%%
theta=75;%Simplification - deflection of
%thin member when force is applied
thetar=theta*pi()/180;
epsilon=d/Ro;
gamma=Ri/Ro;

%Stiffness
K=E*d/(2*thetar^3)*(epsilon/gamma)^2*(1-gamma^2);
%Linear Displacement
Disp=F/K;
%Angular Displacement
Theta=asin(Disp/Ro)*180/pi();
%Rotational Stiffness
K=(F*Ro)/Theta;

end

```

Figure A 52. MATLAB function model of the Smith Rotationally Symmetric Leaf Hinge compliant joint.

12. Name: Trease Rotational

Geometric Parameters: t (thickness), w (width), l (cruciform length)

Additional Notes: Uses two Smith Cruciform flexures to achieve movement, but calculated from different geometric characteristics

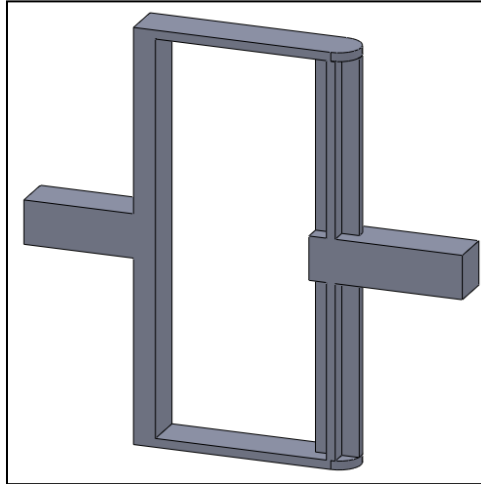


Figure A 53. Solid model representation of the Trease Rotational compliant joint.

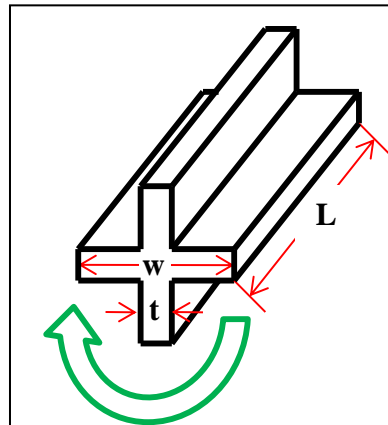


Figure A 54. Geometric characteristics of the Trease Rotational compliant joint.

The angular stiffness of the compliant joint is described by

$$K = \left(\frac{w}{t} - 0.373 \right) \left(\frac{4Gt^4}{3L} \right),$$

where w is the width of the cruciform, t is the thickness of the cruciform, G is the shear modulus of the material, and L is the length of the cruciforms.

```
function [Disptheta,KTheta] = TreaseConceptFun(G,t,L,w,F)
% G = Shear Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----
% t = Thickness of flexure at smallest point
t=t/5;
%%mm
% L = Length of cruciforms
L=L/2; %B/C equations are both sides
% w = Total width of cruciform
% F = Applied force

%%----Solution of Joint----%%
T=F*(w/2);
Q=(w^2*t^2)/(3*w+1.8*t);
Tmax=T/Q;
%Stiffness
KTheta=(w/t-0.373)*(4*G*t^4)/(3*L);
%Angular Displacement
Disptheta=T/KTheta*180/pi();
%Linear Displacement
Disp=sind(Disptheta)*(w/2);
%Transform stiffness to Nm/deg
KTheta=KTheta*pi()/180;

end
```

Figure A 55. MATLAB function model of the Trease Rotational compliant joint.

13. Name: Kyusojin Rotational 6R2

Geometric Parameters: t (thickness of leaf spring), w (leaf spring width), l (leaf spring length), r (radius of circle)

Additional Notes: The two spring leaf points are at a 90 degree angle

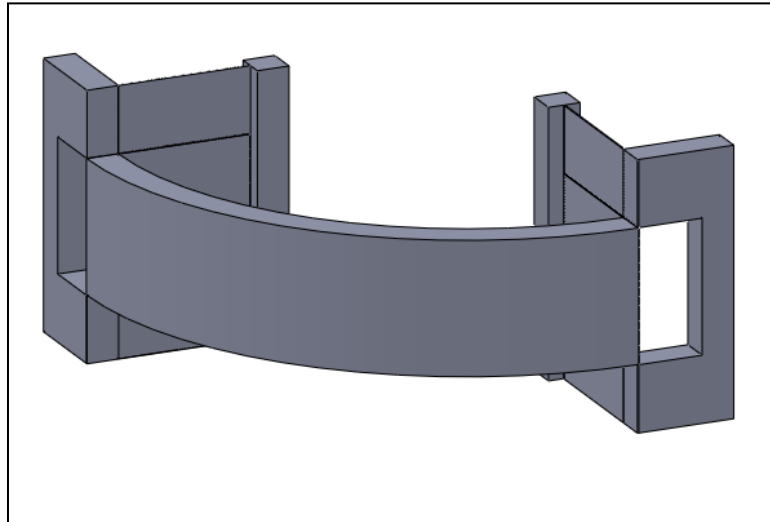


Figure A 56. Solid model representation of the Kyusojin Rotational 6R2 compliant joint.

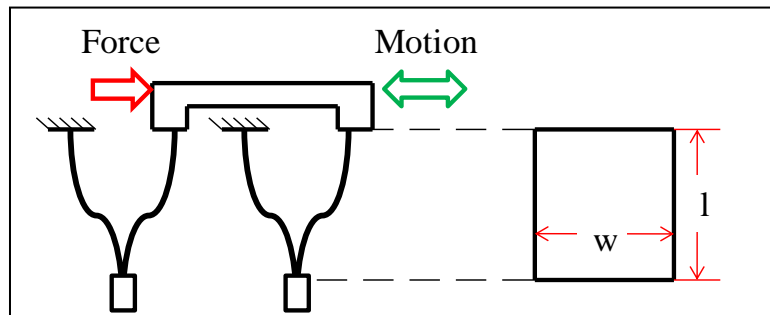


Figure A 57. Geometric characteristics of the Kyusojin Rotational 6R2 compliant joint.

The linear displacement of the upper table of this compliant joint can be defined by

$$i = \sqrt{\frac{5l^6}{3(60(EI)^2)'}}$$

where P is the applied force, l is the length of the leaf springs, E is the Young's Modulus, and I is the moment of inertia of the leaf springs about their center. A simple trigonometric relation,

$$\varphi = \text{asin}\left(\frac{i}{r}\right),$$

where i is the linear displacement, and r is the radius of the circle that makes up the table, can be used to determine the angular rotation of the center point of the table.

```

function [phi,K] = KyusojinRotational6R2Fun(E,t,l,w,P)
% E = Young's Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----
% t = Thickness of leaf spring
t=t/5;
%%mm
% l = Length of leaf spring
% w = width of leaf spring
% P = Applied force
N=4; %# of springs (constant for this geometry)
r= w*2; %radius of circle (center platform)
%Note - the size of the center platform
% (that moves) does not matter

%%----Solution of Joint----%%
I=t*w^3/12;
a=sqrt(N/(E*I));
%Out of plane deflection
dell=(P^2*l^5)/(60*(E*I)^2)*(1/(cos(a*l))-1);
% i=sqrt((5*P^2*l^6)/(3*(60*(E*I)^2)));
%Linear displacement
i=sqrt((5*l^6)/(3*(60*(E*I)^2)));
%Stiffness
K=P/i;
P=kx
%Angular Displacement
phi=asin(i/r)*180/pi();
%Rotational Stiffness
K=(P*r)/phi;

end

```

Figure A 58. MATLAB function model of the Kyusojin Rotational 6R2 compliant joint.

14. Name: Goldfarb Conventional Split Tube

Geometric Parameters: t (thickness), w (leaf spring width), l (leaf spring length), r (radius of circle)

Additional Notes:

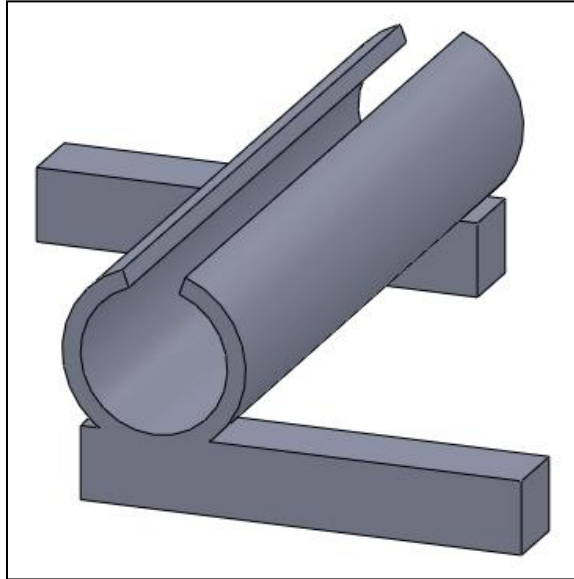


Figure A 59. Solid model representation of the Goldfarb Conventional Split Tube compliant joint.

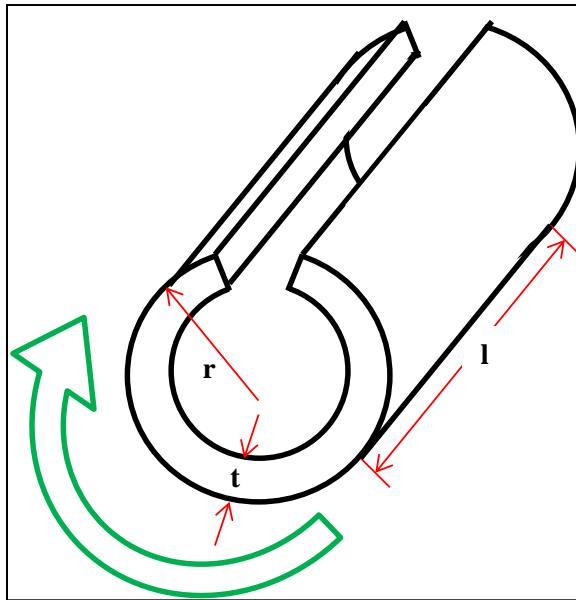


Figure A 60. Geometric characteristics of the Goldfarb Conventional Split Tube compliant joint.

The angular stiffness of the compliant joint can be described by

$$K = \frac{2\pi GRt^3}{3L},$$

where G is the shear modulus, R is the outer radius of the tube, t is the thickness of the tube, and L is the length of the tube.

```

function [Disptheta,K] = ConventionalSplitTubeFun(G,t,L,R,F)
% G = Shear Modulus
%N/mm^2
%%----Joint Dimensional Characteristics----
% t = Thickness of tube
t=t/5;
%%mm
% R = Outer radius of tube
R=R/2;
% L = Tube length
% F = Applied force

%%----Solution of Joint----%%
%Stiffness
K=(2*pi()*G*R*t^3)/(3*L);
%Angular Displacement
Disptheta=(F*R)/K*180/pi();
%Linear Displacement
Disp=sind(Disptheta)*R;
%Rotational Stiffness
K=(F*R)/Disptheta;

end

```

Figure A 61. MATLAB function model of the Goldfarb Conventional Split Tube compliant joint.

APPENDIX B. COMPLETE SELECTION ALGORITHM MATLAB CODE

```

function [ a ] = DecisiontreeFunction( ty,rom,k,s1,s2,F,E,G )
    %type,range of motion,stiffness,sizevalue1,sizevalue2,E,G,load

    %Used if no F value is specified - can be changed by user
    Fset=[.1,1,10];

    %a is used as a counter
    a=0;
    nfig=1;
    %%-----%%
    %%Conversion of strings (user input from gui) to type double
    s1 = str2double(s1);
    s2 = str2double(s2);

    romc=isequal('na',rom);
    if romc==0
        rom = str2double(rom);
    end

    kc=isequal('na',k);
    if kc==0
        k = str2double(k);
    end

    fc=isequal('na',F);
    if fc==0
        F=str2double(F);
    end

    %%-----%%
    %%Logic Tree%%
    %%-----%%

    if ty==1
        fprintf('Translational Joints are selected\n')
        if romc==0
            fprintf('RoM has a value, %2.2f mm.\n',rom)
            if kc==0
                fprintf('K has a value, %2.2f N/mm.\n',k)
                if fc==0
                    fprintf('F has a value, %2.2f N.\n',F)
                    %This tree - Y,Y,Y -> Search using Load as F
                    applied,
                    %find geometry that results in rom and k.
                    %-----
                    %n=# of joints in function
                    n=6;
                    Disp=zeros(n,5);
                    for i=1:n

```

```

        Disp(i,5)=i;
    end

    %Find the minimum and maximum displacement for each
joint - leave thickness
    %constant, vary the depth of the joint from each
iteration

[Disp(1,1),Disp(1,3)]=SmithRectilinearSpringFun(E,s1,s1,s2,F);%max disp
[Disp(1,2),Disp(1,4)]=SmithRectilinearSpringFun(E,s1,s2,s2,F);%min disp

[Disp(2,1),Disp(2,3)]=ParallelStripFun(E,s1,s1,s2,F);
[Disp(2,2),Disp(2,4)]=ParallelStripFun(E,s2,s1,s2,F);

[Disp(3,1),Disp(3,3)]=KyusojinLinear6L1Fun(
E,s1,s1,s2,F);
[Disp(3,2),Disp(3,4)]=KyusojinLinear6L1Fun(
E,s1,s2,s2,F);

[Disp(4,1),Disp(4,3)]=TreaseTranslationalFun(
E,s1,s2,s2,F);
[Disp(4,2),Disp(4,4)]=TreaseTranslationalFun(
E,s1,s2,s1,F);

[Disp(5,1),Disp(5,3)]=XuTranslationalFun(
E,s1,s2,s1,s2,F);
[Disp(5,2),Disp(5,4)]=XuTranslationalFun(
E,s1,s1,s1,s2,F);

[Disp(6,1),Disp(6,3)]=SmithNotchHingeFun(
E,s2,s1,s2,s2,F);
[Disp(6,2),Disp(6,4)]=SmithNotchHingeFun(
E,s1,s1,s2,s1,F);

    Check=zeros(n,2);

    for i=1:n

Check(i,1)=CheckRangeFun(Disp(i,2),Disp(i,1),rom);%min max value
Check(i,2)=CheckRangeFun(Disp(i,3),Disp(i,4),k);
    end

names=char('SmithRectilinear','ParallelStrip','KyusojinLinear6L1',...
'TreaseTranslational','XuTranslational','SmithNotchHinge');
    b=1;

```

```

for i=1:n
    if (Check(i,1)==0 || Check(i,2)==0)
        a(b,1)=i;
        b=b+1;
    end
end

%-----

% Inverse of above for pareto curve production %
d=1;
e=0;
for i=1:n
    if (Check(i,1)==1 && Check(i,2)==1)
        e(d,1)=i;
        d=d+1;
    end
end

z = ParetoFunT_YYY( e,E,s1,s2,F,nfig);
%-----

sa=size(a);
c=sa(1);
for i=1:sa(1)
    names(a(c),:)=[];
    c=c-1;
end
fprintf('WARNING! Displacements larger than 30%% of
\n the total length of the compliant joint may \n
be be inaccurate.')
fprintf('\nPossible joints for this user input
are:\n')

disp(names)
%-----

---%

%%END OF Y,Y,Y TREE
%-----

---%

else
    fprintf('F does not have a value (na)\n')
    %This tree - Y,Y,N -> Search using generic F set
    %find geometry that results in rom and k.
    %-----

---%

%n=# of joints in function
n=6;
Disp=zeros(n,5);
for i=1:n
    Disp(i,5)=i;
end

```

```

                                %Find the minimum and maximum displacement for each
joint - leave thickness
                                %constant, vary the depth of the joint from each
iteration
                                sizeofset=size(Fset);
                                nfset=sizeofset(2);
                                %This is a loop over the user defined forces, Fset
(current 3
                                %forces a factor of 10 apart, starting with .1)
                                for j=1:nfset
                                    F=Fset(j);

[Disp(1,1),Disp(1,3)]=SmithRectilinearSpringFun(E,s1,s1,s2,F);%max disp
[Disp(1,2),Disp(1,4)]=SmithRectilinearSpringFun(E,s1,s2,s2,F);%min disp

[Disp(2,1),Disp(2,3)]=ParallelStripFun(E,s1,s1,s2,F);
[Disp(2,2),Disp(2,4)]=ParallelStripFun(E,s2,s1,s2,F);

                                [Disp(3,1),Disp(3,3)]=KyusojinLinear6L1Fun(
E,s1,s1,s2,F );
                                [Disp(3,2),Disp(3,4)]=KyusojinLinear6L1Fun(
E,s1,s2,s2,F );

                                [Disp(4,1),Disp(4,3)]=TreaseTranslationalFun(
E,s1,s2,s2,F );
                                [Disp(4,2),Disp(4,4)]=TreaseTranslationalFun(
E,s1,s2,s1,F );

                                [Disp(5,1),Disp(5,3)]=XuTranslationalFun(
E,s1,s2,s1,s2,F );
                                [Disp(5,2),Disp(5,4)]=XuTranslationalFun(
E,s1,s1,s1,s2,F );

                                [Disp(6,1),Disp(6,3)]=SmithNotchHingeFun(
E,s2,s1,s2,s2,F );
                                [Disp(6,2),Disp(6,4)]=SmithNotchHingeFun(
E,s1,s1,s2,s1,F );

                                Check=zeros(n,2);

                                for i=1:n
Check(i,1)=CheckRangeFun(Disp(i,2),Disp(i,1),rom);
Check(i,2)=CheckRangeFun(Disp(i,3),Disp(i,4),k);
                                end

```

```

names=char('SmithRectilinear','ParallelStrip','KyusojinLinear6L1',...
'TreaseTranslational','XuTranslational','SmithNotchHinge');
    b=1;
    for i=1:n
        if (Check(i,1)==0 || Check(i,2)==0)
            a(b,1)=i;
            b=b+1;
        end
    end

%-----
% Inverse of above for pareto curve production
%
    d=1;
    e=0;
    for i=1:n
        if (Check(i,1)==1 && Check(i,2)==1)
            e(d,1)=i;
            d=d+1;
        end
    end

    z = ParetoFunT_YYY( e,E,s1,s2,F,nfig);
    %-----
    sa=size(a);
    c=sa(1);
    for i=1:sa(1)
        names(a(c),:)=[];
        c=c-1;
    end
    fprintf('WARNING! Displacements larger than
30%% of \n the total length of the compliant joint may \n be
inaccurate.')
        fprintf('\nPossible joints for this user input
using a force of %f are:\n',Fset(j))
        disp(names)
        a=0;
    end
%-----
---%
%%END OF Y,Y,N TREE
%-----
---%
    end
else
    fprintf('K does not have a value (na)\n')
    if fc==0
        fprintf('F has a value, %2.2f N.\n',F)
        %This tree - Y,N,Y -> Set F as load applied, check
for

```



```

%any joints that can result in rom.
%-----
---%
%n=# of joints in function
n=6;
Disp=zeros(n,5);
for i=1:n
    Disp(i,5)=i;
end

%Find the minimum and maximum displacement for each
joint - leave thickness
%constant, vary the depth of the joint from each
iteration

[Disp(1,1),Disp(1,3)]=SmithRectilinearSpringFun(E,s1,s1,s2,F);%max disp
[Disp(1,2),Disp(1,4)]=SmithRectilinearSpringFun(E,s1,s2,s2,F);%min disp

[Disp(2,1),Disp(2,3)]=ParallelStripFun(E,s1,s1,s2,F);
[Disp(2,2),Disp(2,4)]=ParallelStripFun(E,s2,s1,s2,F);

[Disp(3,1),Disp(3,3)]=KyusojinLinear6L1Fun(
E,s1,s1,s2,F);
[Disp(3,2),Disp(3,4)]=KyusojinLinear6L1Fun(
E,s1,s2,s2,F);

[Disp(4,1),Disp(4,3)]=TreaseTranslationalFun(
E,s1,s2,s2,F);
[Disp(4,2),Disp(4,4)]=TreaseTranslationalFun(
E,s1,s2,s1,F);

[Disp(5,1),Disp(5,3)]=XuTranslationalFun(
E,s1,s2,s1,s2,F);
[Disp(5,2),Disp(5,4)]=XuTranslationalFun(
E,s1,s1,s1,s2,F);

[Disp(6,1),Disp(6,3)]=SmithNotchHingeFun(
E,s2,s1,s2,s2,F);
[Disp(6,2),Disp(6,4)]=SmithNotchHingeFun(
E,s1,s1,s2,s1,F);

Check=zeros(n,1);

for i=1:n
Check(i)=CheckRangeFun(Disp(i,2),Disp(i,1),rom);
end

```

```

names=char('SmithRectilinear','ParallelStrip','KyusojinLinear6L1',...
'TreaseTranslational','XuTranslational','SmithNotchHinge');
    b=1;
    for i=1:n
        if Check(i)==0
            a(b,1)=i;
            b=b+1;
        end
    end

%-----

% Inverse of above for pareto curve production %
d=1;
e=0;
for i=1:n
    if Check(i,1)==1
        e(d,1)=i;
        d=d+1;
    end
end

z = ParetoFunT_YYY( e,E,s1,s2,F,nfig);
%-----
sa=size(a);
c=sa(1);
for i=1:sa(1)
    names(a(c),:)=[];
    c=c-1;
end
fprintf('WARNING! Displacements larger than 30%% of
\n the total length of the compliant joint may \n be be inaccurate.')

fprintf('\nPossible joints for this user input
are:\n')

disp(names)
%-----

%%END OF Y,N,Y TREE
%-----

else
    fprintf('F does not have a value (na)\n')
    %This tree - Y,N,N -> Search using generic F set
    %find joints that can result in rom.
    %-----

    %n=# of joints in function
    n=6;

```

```

Disp=zeros(n,5);
for i=1:n
    Disp(i,5)=i;
end

%Find the minimum and maximum displacement for each
joint - leave thickness
%constant, vary the depth of the joint from each
iteration

sizeofset=size(Fset);
nfset=sizeofset(2);
%This is a loop over the user defined forces, Fset
(current 3

%forces a factor of 10 apart, starting with .1)
for j=1:nfset
    F=Fset(j);

[Disp(1,1),Disp(1,3)]=SmithRectilinearSpringFun(E,s1,s1,s2,F);%max disp
[Disp(1,2),Disp(1,4)]=SmithRectilinearSpringFun(E,s1,s2,s2,F);%min disp

[Disp(2,1),Disp(2,3)]=ParallelStripFun(E,s1,s1,s2,F);
[Disp(2,2),Disp(2,4)]=ParallelStripFun(E,s2,s1,s2,F);

[Disp(3,1),Disp(3,3)]=KyusojinLinear6L1Fun(
E,s1,s1,s2,F );
[Disp(3,2),Disp(3,4)]=KyusojinLinear6L1Fun(
E,s1,s2,s2,F );

[Disp(4,1),Disp(4,3)]=TreaseTranslationalFun(
E,s1,s2,s2,F );
[Disp(4,2),Disp(4,4)]=TreaseTranslationalFun(
E,s1,s2,s1,F );

[Disp(5,1),Disp(5,3)]=XuTranslationalFun(
E,s1,s2,s1,s2,F );
[Disp(5,2),Disp(5,4)]=XuTranslationalFun(
E,s1,s1,s1,s2,F );

[Disp(6,1),Disp(6,3)]=SmithNotchHingeFun(
E,s2,s1,s2,s2,F );
[Disp(6,2),Disp(6,4)]=SmithNotchHingeFun(
E,s1,s1,s2,s1,F );

Check=zeros(n,1);

for i=1:n
Check(i)=CheckRangeFun(Disp(i,2),Disp(i,1),rom);
end

```

```

names=char('SmithRectilinear','ParallelStrip','KyusojinLinear6L1',...
'TreaseTranslational','XuTranslational','SmithNotchHinge');
    b=1;
    for i=1:n
        if Check(i)==0
            a(b,1)=i;
            b=b+1;
        end
    end

%-----
% Inverse of above for pareto curve production
%
    d=1;
    e=0;
    for i=1:n
        if Check(i,1)==1
            e(d,1)=i;
            d=d+1;
        end
    end

    z = ParetoFunT_YYY( e,E,s1,s2,F,nfig);
    %-----
    sa=size(a);
    c=sa(1);
    for i=1:sa(1)
        names(a(c),:)=[];
        c=c-1;
    end
    fprintf('WARNING! Displacements larger than
30%% of \n the total length of the compliant joint may \n be be
inaccurate.')

    fprintf('\nPossible joints for this user input
using a force of %f are:\n',Fset(j))
    disp(names)
    a=0;
    nfig=nfig+1;
end
%-----
%END OF Y,N,N TREE
%-----
end
else
    fprintf('RoM does not have a value (na)\n')

```

```

        if kc==0
            fprintf('K has a value, %2.2f N/mm.\n',k)
            if fc==0
                fprintf('F has a value, %2.2f N.\n',F)
                %This tree - N,Y,Y -> Search using Load as F
applied,
                %find geometry that results in k.
                %-----%
----%
                %n=# of joints in function
                n=6;
                Disp=zeros(n,5);
                for i=1:n
                    Disp(i,5)=i;
                end

                %Find the minimum and maximum displacement for each
joint - leave thickness
                %constant, vary the depth of the joint from each
iteration

[Disp(1,1),Disp(1,3)]=SmithRectilinearSpringFun(E,s1,s1,s2,F);%max disp
[Disp(1,2),Disp(1,4)]=SmithRectilinearSpringFun(E,s1,s2,s2,F);%min disp

[Disp(2,1),Disp(2,3)]=ParallelStripFun(E,s1,s1,s2,F);
[Disp(2,2),Disp(2,4)]=ParallelStripFun(E,s2,s1,s2,F);

                [Disp(3,1),Disp(3,3)]=KyusojinLinear6L1Fun(
E,s1,s1,s2,F);
                [Disp(3,2),Disp(3,4)]=KyusojinLinear6L1Fun(
E,s1,s2,s2,F);

                [Disp(4,1),Disp(4,3)]=TreaseTranslationalFun(
E,s1,s2,s2,F);
                [Disp(4,2),Disp(4,4)]=TreaseTranslationalFun(
E,s1,s2,s1,F);

                [Disp(5,1),Disp(5,3)]=XuTranslationalFun(
E,s1,s2,s1,s2,F);
                [Disp(5,2),Disp(5,4)]=XuTranslationalFun(
E,s1,s1,s1,s2,F);

                [Disp(6,1),Disp(6,3)]=SmithNotchHingeFun(
E,s2,s1,s2,s2,F);
                [Disp(6,2),Disp(6,4)]=SmithNotchHingeFun(
E,s1,s1,s2,s1,F);

                Check=zeros(n,1);

```

```

        for i=1:n
            Check(i)=CheckRangeFun(Disp(i,3),Disp(i,4),k);
        end

names=char('SmithRectilinear','ParallelStrip','KyusojinLinear6L1',...
'TreaseTranslational','XuTranslational','SmithNotchHinge');
        b=1;
        for i=1:n
            if Check(i)==0
                a(b,1)=i;
                b=b+1;
            end
        end

        %-----
-----%
        % Inverse of above for pareto curve production %
        d=1;
        e=0;
        for i=1:n
            if Check(i,1)==1
                e(d,1)=i;
                d=d+1;
            end
        end

        z = ParetoFunT_YYY( e,E,s1,s2,F,nfig);
        %-----
        sa=size(a);
        c=sa(1);
        for i=1:sa(1)
            names(a(c),:)=[];
            c=c-1;
        end
        fprintf('WARNING! Displacements larger than 30%% of
\n the total length of the compliant joint may \n be be inaccurate.')

        fprintf('\nPossible joints for this user input
are:\n')

        disp(names)
        %-----
-----%
        %%END OF N,Y,Y TREE
        %-----
-----%
        else
            fprintf('F does not have a value (na)\n')
            %This tree - N,Y,N -> Search using generic F set
            applied,
            %find geometry that results in k.
            %-----

```

```

---%
                                %n=# of joints in function
                                n=6;
                                Disp=zeros(n,5);
                                for i=1:n
                                    Disp(i,5)=i;
                                end

                                %Find the minimum and maximum displacement for each
joint - leave thickness
                                %constant, vary the depth of the joint from each
iteration
                                sizeofset=size(Fset);
                                nfset=sizeofset(2);
                                %This is a loop over the user defined forces, Fset
(current 3
                                %forces a factor of 10 apart, starting with .1)
                                for j=1:nfset
                                    F=Fset(j);

[Disp(1,1),Disp(1,3)]=SmithRectilinearSpringFun(E,s1,s1,s2,F);%max disp
[Disp(1,2),Disp(1,4)]=SmithRectilinearSpringFun(E,s1,s2,s2,F);%min disp

[Disp(2,1),Disp(2,3)]=ParallelStripFun(E,s1,s1,s2,F);
[Disp(2,2),Disp(2,4)]=ParallelStripFun(E,s2,s1,s2,F);

                                [Disp(3,1),Disp(3,3)]=KyusojinLinear6L1Fun(
E,s1,s1,s2,F );
                                [Disp(3,2),Disp(3,4)]=KyusojinLinear6L1Fun(
E,s1,s2,s2,F );

                                [Disp(4,1),Disp(4,3)]=TreaseTranslationalFun(
E,s1,s2,s2,F );
                                [Disp(4,2),Disp(4,4)]=TreaseTranslationalFun(
E,s1,s2,s1,F );

                                [Disp(5,1),Disp(5,3)]=XuTranslationalFun(
E,s1,s2,s1,s2,F );
                                [Disp(5,2),Disp(5,4)]=XuTranslationalFun(
E,s1,s1,s1,s2,F );

                                [Disp(6,1),Disp(6,3)]=SmithNotchHingeFun(
E,s2,s1,s2,s2,F );
                                [Disp(6,2),Disp(6,4)]=SmithNotchHingeFun(
E,s1,s1,s2,s1,F );

                                Check=zeros(n,1);

                                for i=1:n

```

```

Check(i)=CheckRangeFun(Disp(i,3),Disp(i,4),k);
    end

names=char('SmithRectilinear','ParallelStrip','KyusojinLinear6L1',...
'TreaseTranslational','XuTranslational','SmithNotchHinge');
    b=1;
    for i=1:n
        if Check(i)==0
            a(b,1)=i;
            b=b+1;
        end
    end

-----
% Inverse of above for pareto curve production
%
d=1;
e=0;
for i=1:n
    if Check(i,1)==1
        e(d,1)=i;
        d=d+1;
    end
end

z = ParetoFunT_YYY( e,E,s1,s2,F,nfig);
-----
sa=size(a);
c=sa(1);
for i=1:sa(1)
    names(a(c),:)=[];
    c=c-1;
end
fprintf('WARNING! Displacements larger than
30%% of \n the total length of the compliant joint may \n be
inaccurate.')

    fprintf('\nPossible joints for this user input
using a force of %f are:\n',Fset(j))
    disp(names)
    a=0;
    nfig=nfig+1;
end
-----
%%END OF N,Y,N TREE
-----
end

```



```

else
    fprintf('K does not have a value (na)\n')
    if fc==0
        fprintf('F has a value, %2.2f N.\n',F)
        %This tree - N,N,Y -> Set F as load applied,
present user
        %all joints, organize by rom.
        %-----%
----%
        n=6;
        Disp=zeros(n,5);
        for i=1:n
            Disp(i,5)=i;
        end

[Disp(1,1),Disp(1,3)]=SmithRectilinearSpringFun(E,s1,s1,s2,F);%max disp
[Disp(1,2),Disp(1,4)]=SmithRectilinearSpringFun(E,s1,s2,s2,F);%min disp

[Disp(2,1),Disp(2,3)]=ParallelStripFun(E,s1,s1,s2,F);
[Disp(2,2),Disp(2,4)]=ParallelStripFun(E,s2,s1,s2,F);

        [Disp(3,1),Disp(3,3)]=KyusojinLinear6L1Fun(
E,s1,s1,s2,F );
        [Disp(3,2),Disp(3,4)]=KyusojinLinear6L1Fun(
E,s1,s2,s2,F );

        [Disp(4,1),Disp(4,3)]=TreaseTranslationalFun(
E,s1,s2,s2,F );
        [Disp(4,2),Disp(4,4)]=TreaseTranslationalFun(
E,s1,s2,s1,F );

        [Disp(5,1),Disp(5,3)]=XuTranslationalFun(
E,s1,s2,s1,s2,F );
        [Disp(5,2),Disp(5,4)]=XuTranslationalFun(
E,s1,s1,s1,s2,F );

        [Disp(6,1),Disp(6,3)]=SmithNotchHingeFun(
E,s2,s1,s2,s2,F );
        [Disp(6,2),Disp(6,4)]=SmithNotchHingeFun(
E,s1,s1,s2,s1,F );

        Check=zeros(n,1);

        Disp=sortrows(Disp,2);
        fprintf('\n\nThe following joints are organized by
RoM,\nLargest to smallest, for the given force, %f N :\n',F)
namest=char('SmithRectilinear','ParallelStrip','KyusojinLinear6L1',...

```

```

'TreaseTranslational','XuTranslational','SmithNotchHinge');
names=cellstr(namest);
for i=n:-1:1
    name=names{Disp(i,5)};
    fprintf('%s with %2.4f mm displacement.
\n',name,Disp(i,2))
end

-----
%
% Inverse of above for creating pareto curves
%
d=1;
e=[1; 2; 3; 4; 5; 6];

z = ParetoFunT_YYY( e,E,s1,s2,F,nfig);
-----
%
-----%
%%END OF N,N,Y TREE
-----%

else
    fprintf('F does not have a value (na)\n')
    %This tree - N,N,N -> Will result in all results
being
    %returned, cannot specify with no information
    -----%
    -----%
    n=6;
    Disp=zeros(n,5);
    for i=1:n
        Disp(i,5)=i;
    end

    sizeofset=size(Fset);
    nfset=sizeofset(2);
    %This is a loop over the user defined forces, Fset
    for j=1:nfset
        F=Fset(j);

[Disp(1,1),Disp(1,3)]=SmithRectilinearSpringFun(E,s1,s1,s2,F);%max disp
[Disp(1,2),Disp(1,4)]=SmithRectilinearSpringFun(E,s1,s2,s2,F);%min disp

[Disp(2,1),Disp(2,3)]=ParallelStripFun(E,s1,s1,s2,F);
[Disp(2,2),Disp(2,4)]=ParallelStripFun(E,s2,s1,s2,F);

```

```

E,s1,s1,s2,F );           [Disp(3,1),Disp(3,3)]=KyusojinLinear6L1Fun(
E,s1,s2,s2,F );           [Disp(3,2),Disp(3,4)]=KyusojinLinear6L1Fun(

E,s1,s2,s2,F );           [Disp(4,1),Disp(4,3)]=TreaseTranslationalFun(
E,s1,s2,s1,F );           [Disp(4,2),Disp(4,4)]=TreaseTranslationalFun(

E,s1,s1,s1,s2,F );         [Disp(5,1),Disp(5,3)]=XuTranslationalFun(
E,s1,s2,s1,s2,F );         [Disp(5,2),Disp(5,4)]=XuTranslationalFun(

E,s2,s1,s2,s2,F );         [Disp(6,1),Disp(6,3)]=SmithNotchHingeFun(
E,s1,s1,s2,s1,F );         [Disp(6,2),Disp(6,4)]=SmithNotchHingeFun(

                                Check=zeros(n,1);

                                Disp=sortrows(Disp,2);

namest=char('SmithRectilinear','ParallelStrip','KyusojinLinear6L1',...
'TreaseTranslational','XuTranslational','SmithNotchHinge');
                                names=cellstr(namest);
                                fprintf('WARNING! Displacements larger than
30%% of \n the total length of the compliant joint may \n be be
inaccurate.')

                                fprintf('\nPossible joints for this user input
using a force of %f N,\n organized by RoM are:\n',Fset(j))
                                for i=n:-1:1
                                    name=names{Disp(i,5)};
                                    fprintf('%s with %2.4f mm displacement.
\n',name,Disp(i,2))

                                end
                                a=0;
                                Disp=zeros(n,5);
                                for i=1:n
                                    Disp(i,5)=i;
                                end
                                end
                                %-----
                                %%END OF N,N,N TREE
                                %-----
                                end
                                end

```

```

end
else
fprintf('Rotational Joints are selected.\n')

%ty,rom,k,s1,s2,E,p,G,maxf,fos
%ty,rom,k,sizevalue1,sizevalue2,E,ro,G,maxload,fos

namest=char('SmithRectilinear','ParallelStrip','KyusojinLinear6L1',...
'TreaseTranslational','XuTranslational','SmithNotchHinge');

namesr=char('JensenCrossAxis','LobontiuCornerFilletted','LobontiuSymmetr
icCircular',...

'LobontiuSymmetricNotch','RotationallySymmetric','SmithAnnulus','SmithC
artwheel',...

'SmithCruciform','SmithTwoAxis','TangSymmetricCircular','TreaseConcept'
,....
'VShape','Kyusojin6R2','ConventionalSplitTube');
%n=# of joints in function
n=14;
Disp=zeros(n,5);
for i=1:n
    Disp(i,5)=i;
end
if romc==0
    fprintf('RoM has a value, %2.2f degrees.\n',rom)
    if kc==0
        fprintf('K has a value, %2.2f Nmm/degree.\n',k)
        if fc==0
            fprintf('F has a value, %2.2f N.\n',F)
            %This tree - Y,Y,Y -> Search using Load as F
            %find geometry that results in rom and k.
            %-----%
            %Find the minimum and maximum displacement for each
joint - leave thickness
            %constant, vary the depth of the joint from each
iteration

            %Joint 1
            [Disp(1,1),Disp(1,3)] =
JensenCrossAxisFun(E,s1,s1,s1,s2,F);%maxK
            [Disp(1,2),Disp(1,4)] =
JensenCrossAxisFun(E,s1,s2,s2,s2,F);%max disp
            %Joint 2
            [Disp(2,1),Disp(2,3)] =
LobontiuCornerFillettedFun(E,s1,s1,s1,s2,F);%maxK
            [Disp(2,2),Disp(2,4)] =

```

```

LobontiuCornerFilletedFun(E,s2,s2,s2,s2,F);%max disp
    %Joint 3
    [Disp(3,1),Disp(3,3)] =
LobontiuSymmetricCircularFun(E,s1,s1,s2,F);%maxK
    [Disp(3,2),Disp(3,4)] =
LobontiuSymmetricCircularFun(E,s1,s2,s2,F);%max disp
    %Joint 4
    [Disp(4,1),Disp(4,3)] =
LobontiuSymmetricNotchFun(E,s1,s1,s2,F);%maxK
    [Disp(4,2),Disp(4,4)] =
LobontiuSymmetricNotchFun(E,s1,s2,s2,F);%max disp
    %Joint 5
    [Disp(5,1),Disp(5,3)] =
RotationallySymmetricLeafHingeFun(E,s1,s1,F);%maxK
    [Disp(5,2),Disp(5,4)] =
RotationallySymmetricLeafHingeFun(E,s1,s2,F);%max disp
    %Joint 6
    [Disp(6,1),Disp(6,3)] =
SmithAnnulusFun(E,G,s2,s2,F);%maxK
    [Disp(6,2),Disp(6,4)] =
SmithAnnulusFun(E,G,s1,s1,F);%max disp
    %Joint 7
    [Disp(7,1),Disp(7,3)] =
SmithCartwheelFun(E,s1,s1,s2,F);%maxK
    [Disp(7,2),Disp(7,4)] =
SmithCartwheelFun(E,s1,s2,s2,F);%max disp
    %Joint 8
    [Disp(8,1),Disp(8,3)] =
SmithCruciformFun(G,s1,s1,s1,F);%maxK
    [Disp(8,2),Disp(8,4)] =
SmithCruciformFun(G,s1,s2,s1,F);%max disp
    %Joint 9
    [Disp(9,1),Disp(9,3)] =
SmithTwoAxisFun(E,s1,s1,F);%maxK
    [Disp(9,2),Disp(9,4)] =
SmithTwoAxisFun(E,s2,s1,F);%max disp
    %Joint 10
    [Disp(10,1),Disp(10,3)] =
TangSymmetricCircularFun(E,s1,s1,s2,F);%maxK
    [Disp(10,2),Disp(10,4)] =
TangSymmetricCircularFun(E,s2,s1,s2,F);%max disp
    %Joint 11
    [Disp(11,1),Disp(11,3)] =
TreaseConceptFun(G,s1,s1,s1,F);%maxK
    [Disp(11,2),Disp(11,4)] =
TreaseConceptFun(G,s1,s2,s1,F);%max disp
    %Joint 12
    [Disp(12,1),Disp(12,3)] =
VShapeFun(E,s1,s1,s1,s1,s2,F);%maxK
    [Disp(12,2),Disp(12,4)] =
VShapeFun(E,s1,s2,s2,s2,s2,F);%max disp
    %Joint 13
    [Disp(13,1),Disp(13,3)] =

```

```

KyusojinRotational6R2Fun(E,s1,s1,s2,F);%maxK
    [Disp(13,2),Disp(13,4)] =
KyusojinRotational6R2Fun(E,s1,s2,s2,F);%max disp
    %Joint 14
    [Disp(14,1),Disp(14,3)] =
ConventionalSplitTubeFun(G,s1,s1,s2,F);%maxK
    [Disp(14,2),Disp(14,4)] =
ConventionalSplitTubeFun(G,s1,s2,s2,F);%max disp

    Check=zeros(n,2);

    for i=1:n

Check(i,1)=CheckRangeFun(Disp(i,1),Disp(i,2),rom);

Check(i,2)=CheckRangeFun(Disp(i,4),Disp(i,3),k);
    end

    b=1;
    for i=1:n
        if (Check(i,1)==0 || Check(i,2)==0)
            a(b,1)=i;
            b=b+1;
        end
    end

    end

    %-----
-----
    % Inverse of above for pareto curve production %
    d=1;
    e=0;
    for i=1:n
        if (Check(i,1)==1 && Check(i,2)==1)
            e(d,1)=i;
            d=d+1;
        end
    end

    z = ParetoFunR_YYY( e,E,G,s1,s2,F,nfig);
    %-----

    sa=size(a);
    c=sa(1);
    for i=1:sa(1)
        namesr(a(c),:)=[];
        c=c-1;
    end
    fprintf('WARNING! Displacements larger than 30%% of
\n the total length of the compliant joint may \n be be inaccurate.')

    fprintf('\nPossible joints for this user input
are:\n')

    disp(namesr)

```

```

-----%
%-----
---%
%%END OF Y,Y,Y TREE
%-----
---%

else
    fprintf('F does not have a value (na)\n')
    %This tree - Y,Y,N -> Search using generic F set
applied,
    %find geometry that results in rom and k.
%-----
---%

    %Find the minimum and maximum displacement for each
joint - leave thickness
    %constant, vary the depth of the joint from each
iteration

    sizeofset=size(Fset);
    nfset=sizeofset(2);
    %This is a loop over the user defined forces, Fset
(current 3

    %forces a factor of 10 apart, starting with .1)
    for j=1:nfset
        F=Fset(j);
        %Joint 1
            [Disp(1,1),Disp(1,3)] =
JensenCrossAxisFun(E,s1,s1,s1,s2,F);%maxK
            [Disp(1,2),Disp(1,4)] =
JensenCrossAxisFun(E,s1,s2,s2,s2,F);%max disp
        %Joint 2
            [Disp(2,1),Disp(2,3)] =
LobontiuCornerFillettedFun(E,s1,s1,s1,s2,F);%maxK
            [Disp(2,2),Disp(2,4)] =
LobontiuCornerFillettedFun(E,s2,s2,s2,s2,F);%max disp
        %Joint 3
            [Disp(3,1),Disp(3,3)] =
LobontiuSymmetricCircularFun(E,s1,s1,s2,F);%maxK
            [Disp(3,2),Disp(3,4)] =
LobontiuSymmetricCircularFun(E,s1,s2,s2,F);%max disp
        %Joint 4
            [Disp(4,1),Disp(4,3)] =
LobontiuSymmetricNotchFun(E,s1,s1,s2,F);%maxK
            [Disp(4,2),Disp(4,4)] =
LobontiuSymmetricNotchFun(E,s1,s2,s2,F);%max disp
        %Joint 5
            [Disp(5,1),Disp(5,3)] =
RotationallySymmetricLeafHingeFun(E,s1,s1,F);%maxK
            [Disp(5,2),Disp(5,4)] =
RotationallySymmetricLeafHingeFun(E,s1,s2,F);%max disp
        %Joint 6
            [Disp(6,1),Disp(6,3)] =
SmithAnnulusFun(E,G,s2,s2,F);%maxK

```

```

[Disp(6,2),Disp(6,4)] =
SmithAnnulusFun(E,G,s1,s1,F);%max disp
%Joint 7
[Disp(7,1),Disp(7,3)] =
SmithCartwheelFun(E,s1,s1,s2,F);%maxK
[Disp(7,2),Disp(7,4)] =
SmithCartwheelFun(E,s1,s2,s2,F);%max disp
%Joint 8
[Disp(8,1),Disp(8,3)] =
SmithCruciformFun(G,s1,s1,s1,F);%maxK
[Disp(8,2),Disp(8,4)] =
SmithCruciformFun(G,s1,s2,s1,F);%max disp
%Joint 9
[Disp(9,1),Disp(9,3)] =
SmithTwoAxisFun(E,s1,s1,F);%maxK
[Disp(9,2),Disp(9,4)] =
SmithTwoAxisFun(E,s2,s1,F);%max disp
%Joint 10
[Disp(10,1),Disp(10,3)] =
TangSymmetricCircularFun(E,s1,s1,s2,F);%maxK
[Disp(10,2),Disp(10,4)] =
TangSymmetricCircularFun(E,s2,s1,s2,F);%max disp
%Joint 11
[Disp(11,1),Disp(11,3)] =
TreaseConceptFun(G,s1,s1,s1,F);%maxK
[Disp(11,2),Disp(11,4)] =
TreaseConceptFun(G,s1,s2,s1,F);%max disp
%Joint 12
[Disp(12,1),Disp(12,3)] =
VShapeFun(E,s1,s1,s1,s1,s2,F);%maxK
[Disp(12,2),Disp(12,4)] =
VShapeFun(E,s1,s2,s2,s2,s2,F);%max disp
%Joint 13
[Disp(13,1),Disp(13,3)] =
KyusojinRotational6R2Fun(E,s1,s1,s2,F);%maxK
[Disp(13,2),Disp(13,4)] =
KyusojinRotational6R2Fun(E,s1,s2,s2,F);%max disp
%Joint 14
[Disp(14,1),Disp(14,3)] =
ConventionalSplitTubeFun(G,s1,s1,s2,F);%maxK
[Disp(14,2),Disp(14,4)] =
ConventionalSplitTubeFun(G,s1,s2,s2,F);%max disp

Check=zeros(n,2);

namesr=char('JensenCrossAxis','LobontiuCornerFilletted','LobontiuSymmetr
icCircular',...

'LobontiuSymmetricNotch','RotationallySymmetric','SmithAnnulus','SmithC
artwheel',...

'SmithCruciform','SmithTwoAxis','TangSymmetricCircular','TreaseConcept'
,....

```



```

'VShape','Kyusojin6R2','ConventionalSplitTube');

        for i=1:n

Check(i,1)=CheckRangeFun(Disp(i,1),Disp(i,2),rom);

Check(i,2)=CheckRangeFun(Disp(i,4),Disp(i,3),k);
        end

        b=1;
        for i=1:n
            if (Check(i,1)==0 || Check(i,2)==0)
                a(b,1)=i;
                b=b+1;
            end
        end

%-----
% Inverse of above for pareto curve production
%
d=1;
e=0;
for i=1:n
    if (Check(i,1)==1 && Check(i,2)==1)
        e(d,1)=i;
        d=d+1;
    end
end

z = ParetoFunR_YYY( e,E,G,s1,s2,F,nfig);
%-----
sa=size(a);
c=sa(1);
for i=1:sa(1)
    namesr(a(c),:)=[];
    c=c-1;
end
fprintf('WARNING! Displacements larger than
30%% of \n the total length of the compliant joint may \n be
inaccurate.')

        fprintf('\nPossible joints for this user input
using a force of %f N are:\n',Fset(j))
        disp(namesr)
        a=0;
        nfig=nfig+1;
    end
%-----
%-----%
%%END OF Y,Y,N TREE
%-----

```

```

----%
        end
    else
        fprintf('K does not have a value (na)\n')
        if fc==0
            fprintf('F has a value, %2.2f N.\n',F)
            %This tree - Y,N,Y -> Set F as load applied, check
for
            %any joints that can result in rom.
            %-----
----%

            %Find the minimum and maximum displacement for each
joint - leave thickness
            %constant, vary the depth of the joint from each
iteration

            %Joint 1
            [Disp(1,1),Disp(1,3)] =
JensenCrossAxisFun(E,s1,s1,s1,s2,F);%maxK
            [Disp(1,2),Disp(1,4)] =
JensenCrossAxisFun(E,s1,s2,s2,s2,F);%max disp
            %Joint 2
            [Disp(2,1),Disp(2,3)] =
LobontiuCornerFillettedFun(E,s1,s1,s1,s2,F);%maxK
            [Disp(2,2),Disp(2,4)] =
LobontiuCornerFillettedFun(E,s2,s2,s2,s2,F);%max disp
            %Joint 3
            [Disp(3,1),Disp(3,3)] =
LobontiuSymmetricCircularFun(E,s1,s1,s2,F);%maxK
            [Disp(3,2),Disp(3,4)] =
LobontiuSymmetricCircularFun(E,s1,s2,s2,F);%max disp
            %Joint 4
            [Disp(4,1),Disp(4,3)] =
LobontiuSymmetricNotchFun(E,s1,s1,s2,F);%maxK
            [Disp(4,2),Disp(4,4)] =
LobontiuSymmetricNotchFun(E,s1,s2,s2,F);%max disp
            %Joint 5
            [Disp(5,1),Disp(5,3)] =
RotationallySymmetricLeafHingeFun(E,s1,s1,F);%maxK
            [Disp(5,2),Disp(5,4)] =
RotationallySymmetricLeafHingeFun(E,s1,s2,F);%max disp
            %Joint 6
            [Disp(6,1),Disp(6,3)] =
SmithAnnulusFun(E,G,s2,s2,F);%maxK
            [Disp(6,2),Disp(6,4)] =
SmithAnnulusFun(E,G,s1,s1,F);%max disp
            %Joint 7
            [Disp(7,1),Disp(7,3)] =
SmithCartwheelFun(E,s1,s1,s2,F);%maxK
            [Disp(7,2),Disp(7,4)] =
SmithCartwheelFun(E,s1,s2,s2,F);%max disp
            %Joint 8

```

```

        [Disp(8,1),Disp(8,3)] =
SmithCruciformFun(G,s1,s1,s1,F);%maxK
        [Disp(8,2),Disp(8,4)] =
SmithCruciformFun(G,s1,s2,s1,F);%max disp
        %Joint 9
        [Disp(9,1),Disp(9,3)] =
SmithTwoAxisFun(E,s1,s1,F);%maxK
        [Disp(9,2),Disp(9,4)] =
SmithTwoAxisFun(E,s2,s1,F);%max disp
        %Joint 10
        [Disp(10,1),Disp(10,3)] =
TangSymmetricCircularFun(E,s1,s1,s2,F);%maxK
        [Disp(10,2),Disp(10,4)] =
TangSymmetricCircularFun(E,s2,s1,s2,F);%max disp
        %Joint 11
        [Disp(11,1),Disp(11,3)] =
TreaseConceptFun(G,s1,s1,s1,F);%maxK
        [Disp(11,2),Disp(11,4)] =
TreaseConceptFun(G,s1,s2,s1,F);%max disp
        %Joint 12
        [Disp(12,1),Disp(12,3)] =
VShapeFun(E,s1,s1,s1,s1,s2,F);%maxK
        [Disp(12,2),Disp(12,4)] =
VShapeFun(E,s1,s2,s2,s2,s2,F);%max disp
        %Joint 13
        [Disp(13,1),Disp(13,3)] =
KyusojinRotational6R2Fun(E,s1,s1,s2,F);%maxK
        [Disp(13,2),Disp(13,4)] =
KyusojinRotational6R2Fun(E,s1,s2,s2,F);%max disp
        %Joint 14
        [Disp(14,1),Disp(14,3)] =
ConventionalSplitTubeFun(G,s1,s1,s2,F);%maxK
        [Disp(14,2),Disp(14,4)] =
ConventionalSplitTubeFun(G,s1,s2,s2,F);%max disp

        Check=zeros(n,1);

        for i=1:n

Check(i)=CheckRangeFun(Disp(i,1),Disp(i,2),rom);
        end

        b=1;
        for i=1:n
            if Check(i)==0
                a(b,1)=i;
                b=b+1;
            end
        end

        %-----
-----

```

```

% Inverse of above for pareto curve production %
d=1;
e=0;
for i=1:n
    if Check(i,1)==1
        e(d,1)=i;
        d=d+1;
    end
end

z = ParetoFunR_YYY( e,E,G,s1,s2,F,nfig);
%-----
sa=size(a);
c=sa(1);
for i=1:sa(1)
    namesr(a(c),:)=[];
    c=c-1;
end
fprintf('WARNING! Displacements larger than 30%% of
\n the total length of the compliant joint may \n be be inaccurate.')

fprintf('\nPossible joints for this user input
are:\n')

disp(namesr)
%-----
---%
%%END OF Y,N,Y TREE
%-----
---%

else
    fprintf('F does not have a value (na)\n')
    %This tree - Y,N,N -> Search using generic F set
    %find joints that can result in rom.
    %-----
    ---%

    %Find the minimum and maximum displacement for each
joint - leave thickness
    %constant, vary the depth of the joint from each
iteration

    sizeofset=size(Fset);
    nfset=sizeofset(2);
    %This is a loop over the user defined forces, Fset
(current 3

    %forces a factor of 10 apart, starting with .1)
    for j=1:nfset
        F=Fset(j);
        %Joint 1
        [Disp(1,1),Disp(1,3)] =
JensenCrossAxisFun(E,s1,s1,s1,s2,F);%maxK
        [Disp(1,2),Disp(1,4)] =

```

```

JensenCrossAxisFun(E, s1, s2, s2, s2, F); %max disp
    %Joint 2
    [Disp(2,1), Disp(2,3)] =
LobontiuCornerFillettedFun(E, s1, s1, s1, s2, F); %maxK
    [Disp(2,2), Disp(2,4)] =
LobontiuCornerFillettedFun(E, s2, s2, s2, s2, F); %max disp
    %Joint 3
    [Disp(3,1), Disp(3,3)] =
LobontiuSymmetricCircularFun(E, s1, s1, s2, F); %maxK
    [Disp(3,2), Disp(3,4)] =
LobontiuSymmetricCircularFun(E, s1, s2, s2, F); %max disp
    %Joint 4
    [Disp(4,1), Disp(4,3)] =
LobontiuSymmetricNotchFun(E, s1, s1, s2, F); %maxK
    [Disp(4,2), Disp(4,4)] =
LobontiuSymmetricNotchFun(E, s1, s2, s2, F); %max disp
    %Joint 5
    [Disp(5,1), Disp(5,3)] =
RotationallySymmetricLeafHingeFun(E, s1, s1, F); %maxK
    [Disp(5,2), Disp(5,4)] =
RotationallySymmetricLeafHingeFun(E, s1, s2, F); %max disp
    %Joint 6
    [Disp(6,1), Disp(6,3)] =
SmithAnnulusFun(E, G, s2, s2, F); %maxK
    [Disp(6,2), Disp(6,4)] =
SmithAnnulusFun(E, G, s1, s1, F); %max disp
    %Joint 7
    [Disp(7,1), Disp(7,3)] =
SmithCartwheelFun(E, s1, s1, s2, F); %maxK
    [Disp(7,2), Disp(7,4)] =
SmithCartwheelFun(E, s1, s2, s2, F); %max disp
    %Joint 8
    [Disp(8,1), Disp(8,3)] =
SmithCruciformFun(G, s1, s1, s1, F); %maxK
    [Disp(8,2), Disp(8,4)] =
SmithCruciformFun(G, s1, s2, s1, F); %max disp
    %Joint 9
    [Disp(9,1), Disp(9,3)] =
SmithTwoAxisFun(E, s1, s1, F); %maxK
    [Disp(9,2), Disp(9,4)] =
SmithTwoAxisFun(E, s2, s1, F); %max disp
    %Joint 10
    [Disp(10,1), Disp(10,3)] =
TangSymmetricCircularFun(E, s1, s1, s2, F); %maxK
    [Disp(10,2), Disp(10,4)] =
TangSymmetricCircularFun(E, s2, s1, s2, F); %max disp
    %Joint 11
    [Disp(11,1), Disp(11,3)] =
TreaseConceptFun(G, s1, s1, s1, F); %maxK
    [Disp(11,2), Disp(11,4)] =
TreaseConceptFun(G, s1, s2, s1, F); %max disp
    %Joint 12
    [Disp(12,1), Disp(12,3)] =

```

```

VShapeFun(E,s1,s1,s1,s1,s2,F);%maxK
    [Disp(12,2),Disp(12,4)] =
VShapeFun(E,s1,s2,s2,s2,s2,F);%max disp
    %Joint 13
    [Disp(13,1),Disp(13,3)] =
KyusojinRotational6R2Fun(E,s1,s1,s2,F);%maxK
    [Disp(13,2),Disp(13,4)] =
KyusojinRotational6R2Fun(E,s1,s2,s2,F);%max disp
    %Joint 14
    [Disp(14,1),Disp(14,3)] =
ConventionalSplitTubeFun(G,s1,s1,s2,F);%maxK
    [Disp(14,2),Disp(14,4)] =
ConventionalSplitTubeFun(G,s1,s2,s2,F);%max disp

    Check=zeros(n,1);

namesr=char('JensenCrossAxis','LobontiuCornerFilleted','LobontiuSymmetr
icCircular',...

'LobontiuSymmetricNotch','RotationallySymmetric','SmithAnnulus','SmithC
artwheel',...

'SmithCruciform','SmithTwoAxis','TangSymmetricCircular','TreaseConcept'
,...

'VShape','Kyusojin6R2','ConventionalSplitTube');

    for i=1:n

Check(i)=CheckRangeFun(Disp(i,1),Disp(i,2),rom);
    end

    b=1;
    for i=1:n
        if Check(i)==0
            a(b,1)=i;
            b=b+1;
        end
    end

    %-----
-----
% Inverse of above for pareto curve production
%

    d=1;
    e=0;
    for i=1:n
        if Check(i,1)==1
            e(d,1)=i;
            d=d+1;
        end
    end

```

```

end

z = ParetoFunR_YYY( e,E,G,s1,s2,F,nfig);
%-----
sa=size(a);
c=sa(1);
for i=1:sa(1)
    namesr(a(c),:)=[];
    c=c-1;
end
fprintf('WARNING! Displacements larger than
30%% of \n the total length of the compliant joint may \n be
inaccurate.')

fprintf('\nPossible joints for this user input
using a force of %f N are:\n',Fset(j))
disp(namesr)
a=0;
nfig=nfig+1;
end
%-----
---%
%%END OF Y,N,N TREE
%-----
---%

end
end
else
fprintf('RoM does not have a value (na)\n')
if kc==0
    fprintf('K has a value, %2.2f Nmm/degree.\n',k)
    if fc==0
        fprintf('F has a value, %2.2f N.\n',F)
        %This tree - N,Y,Y -> Search using Load as F
        applied,
        %find geometry that results in k.
        %-----
        %Find the minimum and maximum displacement for each
joint - leave thickness
        %constant, vary the depth of the joint from each
iteration
        %Joint 1
        [Disp(1,1),Disp(1,3)] =
JensenCrossAxisFun(E,s1,s1,s1,s2,F);%maxK
        [Disp(1,2),Disp(1,4)] =
JensenCrossAxisFun(E,s1,s2,s2,s2,F);%max disp
        %Joint 2
        [Disp(2,1),Disp(2,3)] =
LobontiuCornerFilletedFun(E,s1,s1,s1,s2,F);%maxK
        [Disp(2,2),Disp(2,4)] =
LobontiuCornerFilletedFun(E,s2,s2,s2,s2,F);%max disp

```

```

%Joint 3
[Disp(3,1),Disp(3,3)] =
LobontiuSymmetricCircularFun(E,s1,s1,s2,F);%maxK
[Disp(3,2),Disp(3,4)] =
LobontiuSymmetricCircularFun(E,s1,s2,s2,F);%max disp
%Joint 4
[Disp(4,1),Disp(4,3)] =
LobontiuSymmetricNotchFun(E,s1,s1,s2,F);%maxK
[Disp(4,2),Disp(4,4)] =
LobontiuSymmetricNotchFun(E,s1,s2,s2,F);%max disp
%Joint 5
[Disp(5,1),Disp(5,3)] =
RotationallySymmetricLeafHingeFun(E,s1,s1,F);%maxK
[Disp(5,2),Disp(5,4)] =
RotationallySymmetricLeafHingeFun(E,s1,s2,F);%max disp
%Joint 6
[Disp(6,1),Disp(6,3)] =
SmithAnnulusFun(E,G,s2,s2,F);%maxK
[Disp(6,2),Disp(6,4)] =
SmithAnnulusFun(E,G,s1,s1,F);%max disp
%Joint 7
[Disp(7,1),Disp(7,3)] =
SmithCartwheelFun(E,s1,s1,s2,F);%maxK
[Disp(7,2),Disp(7,4)] =
SmithCartwheelFun(E,s1,s2,s2,F);%max disp
%Joint 8
[Disp(8,1),Disp(8,3)] =
SmithCruciformFun(G,s1,s1,s1,F);%maxK
[Disp(8,2),Disp(8,4)] =
SmithCruciformFun(G,s1,s2,s1,F);%max disp
%Joint 9
[Disp(9,1),Disp(9,3)] =
SmithTwoAxisFun(E,s1,s1,F);%maxK
[Disp(9,2),Disp(9,4)] =
SmithTwoAxisFun(E,s2,s1,F);%max disp
%Joint 10
[Disp(10,1),Disp(10,3)] =
TangSymmetricCircularFun(E,s1,s1,s2,F);%maxK
[Disp(10,2),Disp(10,4)] =
TangSymmetricCircularFun(E,s2,s1,s2,F);%max disp
%Joint 11
[Disp(11,1),Disp(11,3)] =
TreaseConceptFun(G,s1,s1,s1,F);%maxK
[Disp(11,2),Disp(11,4)] =
TreaseConceptFun(G,s1,s2,s1,F);%max disp
%Joint 12
[Disp(12,1),Disp(12,3)] =
VShapeFun(E,s1,s1,s1,s1,s2,F);%maxK
[Disp(12,2),Disp(12,4)] =
VShapeFun(E,s1,s2,s2,s2,s2,F);%max disp
%Joint 13
[Disp(13,1),Disp(13,3)] =
KyusojinRotational6R2Fun(E,s1,s1,s2,F);%maxK

```



```

[Disp(13,2),Disp(13,4)] =
KyusojinRotational6R2Fun(E,s1,s2,s2,F);%max disp
    %Joint 14
[Disp(14,1),Disp(14,3)] =
ConventionalSplitTubeFun(G,s1,s1,s2,F);%maxK
[Disp(14,2),Disp(14,4)] =
ConventionalSplitTubeFun(G,s1,s2,s2,F);%max disp

Check=zeros(n,1);

for i=1:n
    Check(i)=CheckRangeFun(Disp(i,4),Disp(i,3),k);
end

b=1;
for i=1:n
    if Check(i)==0
        a(b,1)=i;
        b=b+1;
    end
end

%-----

% Inverse of above for pareto curve production %
d=1;
e=0;
for i=1:n
    if Check(i,1)==1
        e(d,1)=i;
        d=d+1;
    end
end

z = ParetoFunR_YYY( e,E,G,s1,s2,F,nfig);
%-----
sa=size(a);
c=sa(1);
for i=1:sa(1)
    namesr(a(c),:)=[];
    c=c-1;
end
fprintf('WARNING! Displacements larger than 30%% of
\n the total length of the compliant joint may \n be be inaccurate.')

fprintf('\nPossible joints for this user input
are:\n')

disp(namesr)
%-----

----%

%%END OF N,Y,Y TREE
%-----

```

```

----%
        else
            fprintf('F does not have a value (na)\n')
            %This tree - N,Y,N -> Search using generic F set
applied,
            %find geometry that results in k.
            %-----%
----%

            %Find the minimum and maximum displacement for each
joint - leave thickness
            %constant, vary the depth of the joint from each
iteration

            sizeofset=size(Fset);
            nfset=sizeofset(2);
            %This is a loop over the user defined forces, Fset
(current 3

            %forces a factor of 10 apart, starting with .1)
            for j=1:nfset
                F=Fset(j);

                    %Joint 1
                    [Disp(1,1),Disp(1,3)] =
JensenCrossAxisFun(E,s1,s1,s1,s2,F);%maxK
                    [Disp(1,2),Disp(1,4)] =
JensenCrossAxisFun(E,s1,s2,s2,s2,F);%max disp
                    %Joint 2
                    [Disp(2,1),Disp(2,3)] =
LobontiuCornerFillettedFun(E,s1,s1,s1,s2,F);%maxK
                    [Disp(2,2),Disp(2,4)] =
LobontiuCornerFillettedFun(E,s2,s2,s2,s2,F);%max disp
                    %Joint 3
                    [Disp(3,1),Disp(3,3)] =
LobontiuSymmetricCircularFun(E,s1,s1,s2,F);%maxK
                    [Disp(3,2),Disp(3,4)] =
LobontiuSymmetricCircularFun(E,s1,s2,s2,F);%max disp
                    %Joint 4
                    [Disp(4,1),Disp(4,3)] =
LobontiuSymmetricNotchFun(E,s1,s1,s2,F);%maxK
                    [Disp(4,2),Disp(4,4)] =
LobontiuSymmetricNotchFun(E,s1,s2,s2,F);%max disp
                    %Joint 5
                    [Disp(5,1),Disp(5,3)] =
RotationallySymmetricLeafHingeFun(E,s1,s1,F);%maxK
                    [Disp(5,2),Disp(5,4)] =
RotationallySymmetricLeafHingeFun(E,s1,s2,F);%max disp
                    %Joint 6
                    [Disp(6,1),Disp(6,3)] =
SmithAnnulusFun(E,G,s2,s2,F);%maxK
                    [Disp(6,2),Disp(6,4)] =
SmithAnnulusFun(E,G,s1,s1,F);%max disp
                    %Joint 7

```

```

        [Disp(7,1),Disp(7,3)] =
SmithCartwheelFun(E,s1,s1,s2,F);%maxK
        [Disp(7,2),Disp(7,4)] =
SmithCartwheelFun(E,s1,s2,s2,F);%max disp
        %Joint 8
        [Disp(8,1),Disp(8,3)] =
SmithCruciformFun(G,s1,s1,s1,F);%maxK
        [Disp(8,2),Disp(8,4)] =
SmithCruciformFun(G,s1,s2,s1,F);%max disp
        %Joint 9
        [Disp(9,1),Disp(9,3)] =
SmithTwoAxisFun(E,s1,s1,F);%maxK
        [Disp(9,2),Disp(9,4)] =
SmithTwoAxisFun(E,s2,s1,F);%max disp
        %Joint 10
        [Disp(10,1),Disp(10,3)] =
TangSymmetricCircularFun(E,s1,s1,s2,F);%maxK
        [Disp(10,2),Disp(10,4)] =
TangSymmetricCircularFun(E,s2,s1,s2,F);%max disp
        %Joint 11
        [Disp(11,1),Disp(11,3)] =
TreaseConceptFun(G,s1,s1,s1,s1,F);%maxK
        [Disp(11,2),Disp(11,4)] =
TreaseConceptFun(G,s1,s2,s1,F);%max disp
        %Joint 12
        [Disp(12,1),Disp(12,3)] =
VShapeFun(E,s1,s1,s1,s1,s2,F);%maxK
        [Disp(12,2),Disp(12,4)] =
VShapeFun(E,s1,s2,s2,s2,s2,F);%max disp
        %Joint 13
        [Disp(13,1),Disp(13,3)] =
KyusojinRotational6R2Fun(E,s1,s1,s2,F);%maxK
        [Disp(13,2),Disp(13,4)] =
KyusojinRotational6R2Fun(E,s1,s2,s2,F);%max disp
        %Joint 14
        [Disp(14,1),Disp(14,3)] =
ConventionalSplitTubeFun(G,s1,s1,s2,F);%maxK
        [Disp(14,2),Disp(14,4)] =
ConventionalSplitTubeFun(G,s1,s2,s2,F);%max disp

        Check=zeros(n,1);

namesr=char('JensenCrossAxis','LobontiuCornerFilletted','LobontiuSymmetr
icCircular',...

'LobontiuSymmetricNotch','RotationallySymmetric','SmithAnnulus','SmithC
artwheel',...

'SmithCruciform','SmithTwoAxis','TangSymmetricCircular','TreaseConcept'
,....

'VShape','Kyusojin6R2','ConventionalSplitTube');

```

```

        for i=1:n
Check(i)=CheckRangeFun(Disp(i,4),Disp(i,3),k);
        end

        b=1;
        for i=1:n
            if Check(i)==0
                a(b,1)=i;
                b=b+1;
            end
        end

        %-----
%
% Inverse of above for pareto curve production

        d=1;
        e=0;
        for i=1:n
            if Check(i,1)==1
                e(d,1)=i;
                d=d+1;
            end
        end

        z = ParetoFunR_YYY( e,E,G,s1,s2,F,nfig);
        %-----
        sa=size(a);
        c=sa(1);
        for i=1:sa(1)
            namesr(a(c),:)=[];
            c=c-1;
        end
        fprintf('WARNING! Displacements larger than
30%% of \n the total length of the compliant joint may \n be be
inaccurate.')

        fprintf('\nPossible joints for this user input
using a force of %f N are:\n',Fset(j))
        disp(namesr)
        a=0;
        nfig=nfig+1;
    end
    %-----
    %%END OF N,Y,N TREE
    %-----
end
else
    fprintf('K does not have a value (na)\n')

```

```

if fc==0
    fprintf('F has a value, %2.2f N.\n',F)
    %This tree - N,N,Y -> Set F as load applied,
present user
    %all joints, organize by rom.
    %-----%
----%

    %Joint 1
    [Disp(1,1),Disp(1,3)] =
JensenCrossAxisFun(E,s1,s1,s1,s2,F);%maxK
    [Disp(1,2),Disp(1,4)] =
JensenCrossAxisFun(E,s1,s2,s2,s2,F);%max disp
    %Joint 2
    [Disp(2,1),Disp(2,3)] =
LobontiuCornerFillettedFun(E,s1,s1,s1,s2,F);%maxK
    [Disp(2,2),Disp(2,4)] =
LobontiuCornerFillettedFun(E,s2,s2,s2,s2,F);%max disp
    %Joint 3
    [Disp(3,1),Disp(3,3)] =
LobontiuSymmetricCircularFun(E,s1,s1,s2,F);%maxK
    [Disp(3,2),Disp(3,4)] =
LobontiuSymmetricCircularFun(E,s1,s2,s2,F);%max disp
    %Joint 4
    [Disp(4,1),Disp(4,3)] =
LobontiuSymmetricNotchFun(E,s1,s1,s2,F);%maxK
    [Disp(4,2),Disp(4,4)] =
LobontiuSymmetricNotchFun(E,s1,s2,s2,F);%max disp
    %Joint 5
    [Disp(5,1),Disp(5,3)] =
RotationallySymmetricLeafHingeFun(E,s1,s1,F);%maxK
    [Disp(5,2),Disp(5,4)] =
RotationallySymmetricLeafHingeFun(E,s1,s2,F);%max disp
    %Joint 6
    [Disp(6,1),Disp(6,3)] =
SmithAnnulusFun(E,G,s2,s2,F);%maxK
    [Disp(6,2),Disp(6,4)] =
SmithAnnulusFun(E,G,s1,s1,F);%max disp
    %Joint 7
    [Disp(7,1),Disp(7,3)] =
SmithCartwheelFun(E,s1,s1,s2,F);%maxK
    [Disp(7,2),Disp(7,4)] =
SmithCartwheelFun(E,s1,s2,s2,F);%max disp
    %Joint 8
    [Disp(8,1),Disp(8,3)] =
SmithCruciformFun(G,s1,s1,s1,F);%maxK
    [Disp(8,2),Disp(8,4)] =
SmithCruciformFun(G,s1,s2,s1,F);%max disp
    %Joint 9
    [Disp(9,1),Disp(9,3)] =
SmithTwoAxisFun(E,s1,s1,F);%maxK
    [Disp(9,2),Disp(9,4)] =
SmithTwoAxisFun(E,s2,s1,F);%max disp

```

```

        %Joint 10
        [Disp(10,1),Disp(10,3)] =
TangSymmetricCircularFun(E,s1,s1,s2,F);%maxK
        [Disp(10,2),Disp(10,4)] =
TangSymmetricCircularFun(E,s2,s1,s2,F);%max disp
        %Joint 11
        [Disp(11,1),Disp(11,3)] =
TreaseConceptFun(G,s1,s1,s1,F);%maxK
        [Disp(11,2),Disp(11,4)] =
TreaseConceptFun(G,s1,s2,s1,F);%max disp
        %Joint 12
        [Disp(12,1),Disp(12,3)] =
VShapeFun(E,s1,s1,s1,s1,s2,F);%maxK
        [Disp(12,2),Disp(12,4)] =
VShapeFun(E,s1,s2,s2,s2,s2,F);%max disp
        %Joint 13
        [Disp(13,1),Disp(13,3)] =
KyusojinRotational6R2Fun(E,s1,s1,s2,F);%maxK
        [Disp(13,2),Disp(13,4)] =
KyusojinRotational6R2Fun(E,s1,s2,s2,F);%max disp
        %Joint 14
        [Disp(14,1),Disp(14,3)] =
ConventionalSplitTubeFun(G,s1,s1,s2,F);%maxK
        [Disp(14,2),Disp(14,4)] =
ConventionalSplitTubeFun(G,s1,s2,s2,F);%max disp

        Check=zeros(n,1);

        Disp=sortrows(Disp,2);
        fprintf('WARNING! Displacements larger than 30%% of
\n the total length of the compliant joint may \n be be inaccurate.')

        fprintf('\n\nThe following joints are organized by
RoM,\nLargest to smallest, for the given force, %f N :\n',F)

        names=cellstr(namesr);
        for i=n:-1:1
            name=names{Disp(i,5)};
            fprintf('%s with %2.4f mm displacement.
\n',name,Disp(i,2))

        end

        %-----
        % Inverse of above for pareto curve production %
        d=1;
        e=0;
        e=[1;2;3;4;5;6;7;8;9;10;11;12;13;14];

        z = ParetoFunR_YYY( e,E,G,s1,s2,F,nfig);
        %-----

```

```

-----%
---%
%%END OF N,N,Y TREE
-----%
---%
else
    fprintf('F does not have a value (na)\n')
    %This tree - N,N,N -> Will result in all results
being
    %returned, cannot specify with no information
    %-----%
---%
    %Find the minimum and maximum displacement for each
joint - leave thickness
    %constant, vary the depth of the joint from each
iteration

    sizeofset=size(Fset);
    nfset=sizeofset(2);
    %This is a loop over the user defined forces, Fset
(current 3
    %forces a factor of 10 apart, starting with .1)
    for j=1:nfset
        F=Fset(j);
        %Joint 1
        [Disp(1,1),Disp(1,3)] =
JensenCrossAxisFun(E,s1,s1,s1,s2,F);%maxK
        [Disp(1,2),Disp(1,4)] =
JensenCrossAxisFun(E,s1,s2,s2,s2,F);%max disp
        %Joint 2
        [Disp(2,1),Disp(2,3)] =
LobontiuCornerFilletedFun(E,s1,s1,s1,s2,F);%maxK
        [Disp(2,2),Disp(2,4)] =
LobontiuCornerFilletedFun(E,s2,s2,s2,s2,F);%max disp
        %Joint 3
        [Disp(3,1),Disp(3,3)] =
LobontiuSymmetricCircularFun(E,s1,s1,s2,F);%maxK
        [Disp(3,2),Disp(3,4)] =
LobontiuSymmetricCircularFun(E,s1,s2,s2,F);%max disp
        %Joint 4
        [Disp(4,1),Disp(4,3)] =
LobontiuSymmetricNotchFun(E,s1,s1,s2,F);%maxK
        [Disp(4,2),Disp(4,4)] =
LobontiuSymmetricNotchFun(E,s1,s2,s2,F);%max disp
        %Joint 5
        [Disp(5,1),Disp(5,3)] =
RotationallySymmetricLeafHingeFun(E,s1,s1,F);%maxK
        [Disp(5,2),Disp(5,4)] =
RotationallySymmetricLeafHingeFun(E,s1,s2,F);%max disp
        %Joint 6
        [Disp(6,1),Disp(6,3)] =
SmithAnnulusFun(E,G,s2,s2,F);%maxK

```

```

        [Disp(6,2),Disp(6,4)] =
SmithAnnulusFun(E,G,s1,s1,F);%max disp
        %Joint 7
        [Disp(7,1),Disp(7,3)] =
SmithCartwheelFun(E,s1,s1,s2,F);%maxK
        [Disp(7,2),Disp(7,4)] =
SmithCartwheelFun(E,s1,s2,s2,F);%max disp
        %Joint 8
        [Disp(8,1),Disp(8,3)] =
SmithCruciformFun(G,s1,s1,s1,F);%maxK
        [Disp(8,2),Disp(8,4)] =
SmithCruciformFun(G,s1,s2,s1,F);%max disp
        %Joint 9
        [Disp(9,1),Disp(9,3)] =
SmithTwoAxisFun(E,s1,s1,F);%maxK
        [Disp(9,2),Disp(9,4)] =
SmithTwoAxisFun(E,s2,s1,F);%max disp
        %Joint 10
        [Disp(10,1),Disp(10,3)] =
TangSymmetricCircularFun(E,s1,s1,s2,F);%maxK
        [Disp(10,2),Disp(10,4)] =
TangSymmetricCircularFun(E,s2,s1,s2,F);%max disp
        %Joint 11
        [Disp(11,1),Disp(11,3)] =
TreaseConceptFun(G,s1,s1,s1,F);%maxK
        [Disp(11,2),Disp(11,4)] =
TreaseConceptFun(G,s1,s2,s1,F);%max disp
        %Joint 12
        [Disp(12,1),Disp(12,3)] =
VShapeFun(E,s1,s1,s1,s1,s2,F);%maxK
        [Disp(12,2),Disp(12,4)] =
VShapeFun(E,s1,s2,s2,s2,s2,F);%max disp
        %Joint 13
        [Disp(13,1),Disp(13,3)] =
KyusojinRotational6R2Fun(E,s1,s1,s2,F);%maxK
        [Disp(13,2),Disp(13,4)] =
KyusojinRotational6R2Fun(E,s1,s2,s2,F);%max disp
        %Joint 14
        [Disp(14,1),Disp(14,3)] =
ConventionalSplitTubeFun(G,s1,s1,s2,F);%maxK
        [Disp(14,2),Disp(14,4)] =
ConventionalSplitTubeFun(G,s1,s2,s2,F);%max disp

        Check=zeros(n,1);

        Disp=sortrows(Disp,2);
        fprintf('WARNING! Displacements larger than
30%% of \n the total length of the compliant joint may \n be be
inaccurate.')

        fprintf('\nPossible joints for this user input
using a force of %2.3f N,\n organized by RoM are:\n',Fset(j))

```


APPENDIX C. MATLAB GUI CODE

```
function varargout = JointMenu(varargin)
% JOINTMENU MATLAB code for JointMenu.fig
%     JOINTMENU, by itself, creates a new JOINTMENU or raises the
existing
%     singleton*.
%
%     H = JOINTMENU returns the handle to a new JOINTMENU or the
handle to
%     the existing singleton*.
%
%     JOINTMENU('CALLBACK', hObject,eventData,handles,...) calls the
local
%     function named CALLBACK in JOINTMENU.M with the given input
arguments.
%
%     JOINTMENU('Property','Value',...) creates a new JOINTMENU or
raises the
%     existing singleton*. Starting from the left, property value
pairs are
%     applied to the GUI before JointMenu_OpeningFcn gets called. An
%     unrecognized property name or invalid value makes property
application
%     stop. All inputs are passed to JointMenu_OpeningFcn via
varargin.
%
%     *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only
one
%     instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help JointMenu

% Last Modified by GUIDE v2.5 12-Feb-2015 16:04:50

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @JointMenu_OpeningFcn, ...
                  'gui_OutputFcn',  @JointMenu_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
```

```

gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before JointMenu is made visible.
function JointMenu_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to JointMenu (see VARARGIN)

% Create the values that will populate all pre-set fields
handles.Translational=1;
handles.Rotational=0;

handles.rom='na';

handles.k='na';

handles.s1='0.5';
handles.s2='10';

handles.f='na';

handles.A1E=73000;
handles.PLAE=2800;
handles.ABSE=2587.5;
handles.A1G=28000;
handles.PLAG=875;
handles.ABSG=875;

%Set initial data value for drop down box values
handles.type = handles.Translational;
handles.matE = handles.A1E;
handles.matG = handles.A1G;

% Choose default command line output for JointMenu
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes JointMenu wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = JointMenu_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);

```

```

% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on selection change in type_popup.
function type_popup_Callback(hObject, eventdata, handles)
% hObject    handle to type_popup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Determine the selected data set.
str = get(hObject, 'String');
val = get(hObject, 'Value');
% Set current data to the selected data set.
switch str{val};
case 'Translational' % User selects Translational.
    handles.type = handles.Translational;
case 'Rotational' % User selects Rotational.
    handles.type = handles.Rotational;
end
% Save the handles structure.
guidata(hObject,handles)

% Hints: contents = cellstr(get(hObject,'String')) returns type_popup
% contents as cell array
% contents{get(hObject,'Value')} returns selected item from
% type_popup

% --- Executes during object creation, after setting all properties.
function type_popup_CreateFcn(hObject, eventdata, handles)
% hObject    handle to type_popup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: popupmenu controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function rom_edit_Callback(hObject, eventdata, handles)
% hObject    handle to rom_edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

rom = get(hObject, 'string');
handles.rom = rom;
guidata(gcbo, handles);

% --- Executes during object creation, after setting all properties.
function rom_edit_CreateFcn(hObject, ~, handles)
% hObject    handle to rom_edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUiControlBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end

function k_edit_Callback(hObject, eventdata, handles)
% hObject    handle to k_edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
k = get(hObject, 'string');
handles.k = k;
guidata(gcbo, handles);
% Hints: get(hObject, 'String') returns contents of k_edit as text
%       str2double(get(hObject, 'String')) returns contents of k_edit
as a double

% --- Executes during object creation, after setting all properties.
function k_edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to k_edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUiControlBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end

function s1_edit_Callback(hObject, eventdata, handles)
% hObject    handle to s1 edit (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
s1 = get(hObject, 'string');
handles.s1 = s1;
guidata(gcbo, handles);
% Hints: get(hObject, 'String') returns contents of s1_edit as text
%         str2double(get(hObject, 'String')) returns contents of s1_edit
as a double

% --- Executes during object creation, after setting all properties.
function s1_edit_CreateFcn(hObject, eventdata, handles)
% hObject handle to s1_edit (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUiControlBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end

function s2_edit_Callback(hObject, eventdata, handles)
% hObject handle to s2_edit (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
s2 = get(hObject, 'string');
handles.s2 = s2;
guidata(gcbo, handles);
% Hints: get(hObject, 'String') returns contents of s2_edit as text
%         str2double(get(hObject, 'String')) returns contents of s2_edit
as a double

% --- Executes during object creation, after setting all properties.
function s2_edit_CreateFcn(hObject, eventdata, handles)
% hObject handle to s2_edit (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUiControlBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end

```

```

% --- Executes on selection change in mat_popupmenu.
function mat_popupmenu_Callback(hObject, eventdata, handles)
% hObject    handle to mat_popupmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Determine the selected data set.
str = get(hObject, 'String');
val = get(hObject, 'Value');
% Set current data to the selected data set.
switch str{val};
case 'Aluminum' % User selects Aluminum.
    handles.matE = handles.A1E;
    handles.matG = handles.A1G;
case 'PLA' % User selects PLA.
    handles.matE = handles.PLAE;
    handles.matG = handles.PLAG;
case 'ABS' % User selects PLA.
    handles.matE = handles.ABSE;
    handles.matG = handles.ABSG;
end
% Save the handles structure.
guidata(hObject,handles)

% Hints: contents = cellstr(get(hObject,'String')) returns
mat_popupmenu contents as cell array
%         contents{get(hObject,'Value')} returns selected item from
mat_popupmenu

% --- Executes during object creation, after setting all properties.
function mat_popupmenu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to mat_popupmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function f_edit_Callback(hObject, eventdata, handles)
% hObject    handle to f_edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
f = get(hObject, 'string');
handles.f = f;
guidata(gcbo,handles);

```

```

% Hints: get(hObject,'String') returns contents of f_edit as text
%         str2double(get(hObject,'String')) returns contents of f_edit
as a double

% --- Executes during object creation, after setting all properties.
function f_edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to f_edit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in done_pushbutton1.
function done_pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to done_pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get user input from GUI
% disp(handles.type);
% disp(handles.rom);
% disp(handles.k);
% disp(handles.s1);
% disp(handles.s2);
% disp(handles.f);
% disp(handles.matE);
% disp(handles.matG);

[ z ] =
DecisiontreeFunction(handles.type,handles.rom,handles.k,handles.s1,hand
les.s2,handles.f,handles.matE,handles.matG);

```

Figure A 63.The MATLAB function that controls the GUI.

APPENDIX D. MATLAB PARETO CURVE GENERATING CODE

```
function [ fig ] = ParetoFunT_YYY( array,E,s1,s2,F,nfig )

    s1c=s1;
    s2c=s2;
    scount=(s2-s1)/10;
    Disp=[0];
    k=[0];

    sizea=size(array);
    nsubplot=sizea(1);
    width=ceil(nsubplot/2);
    height=2;
    num=1;

    fig=figure(nfig);
    b=1;
    a=ismember(1,array);
    if a == 1
        %pareto for joint 1
        for i=s1:scount:s2
            [Disp(b),k(b)]=SmithRectilinearSpringFun(E,s1c,i,s2c,F);
            b=b+1;
        end
        subplot(height,width,num)
        plot(Disp,k)
        xlabel('Displacement (mm)')
        ylabel('Stiffness (N/mm)')
        title('Pareto Curve for Smith Rectilinear Spring')
        num=num+1;
    end

    b=1;
    a=ismember(2,array);
    if a == 1
        %pareto for joint 2
        for i=s1:scount:s2
            [Disp(b),k(b)]=ParallelStripFun(E,i,s1c,s2c,F);
            b=b+1;
        end
        subplot(height,width,num)
        plot(Disp,k)
        xlabel('Displacement (mm)')
        ylabel('Stiffness (N/mm)')
        title('Pareto Curve for Parallel Strip')
        num=num+1;
    end

    b=1;
    a=ismember(3,array);
    if a == 1
```

```

%pareto for joint 3
for i=s1:scount:s2
    [Disp(b),k(b)]=KyusojinLinear6L1Fun(E,s1c,i,s2c,F);
    b=b+1;
end
subplot(height,width,num)
plot(Disp,k)
xlabel('Displacement (mm)')
ylabel('Stiffness (N/mm)')
title('Pareto Curve for Kyusojin Linear 6L1')
num=num+1;
end

b=1;
a=ismember(4,array);
if a == 1
    %pareto for joint 4
    j=s2;
    for i=s1:scount:s2
        [Disp(b),k(b)]=TreaseTranslationalFun(E,s1c,s2c,j,F);
        b=b+1;
        j=j-scount;
    end
    subplot(height,width,num)
    plot(Disp,k)
    xlabel('Displacement (mm)')
    ylabel('Stiffness (N/mm)')
    title('Pareto Curve for Trease Translational')
    num=num+1;
end

b=1;
a=ismember(5,array);
if a == 1
    %pareto for joint 5
    j=s2;
    for i=s1:scount:s2
        [Disp(b),k(b)]=XuTranslationalFun(E,s1c,j,s1c,s2c,F);
        b=b+1;
        j=j-scount;
    end
    subplot(height,width,num)
    plot(Disp,k)
    xlabel('Displacement (mm)')
    ylabel('Stiffness (N/mm)')
    title('Pareto Curve for Xu Translational')
    num=num+1;
end

b=1;
a=ismember(6,array);
if a == 1
    %pareto for joint 6

```

```

        j=s2;
        for i=s1:scount:s2
            [Disp(b),k(b)]=SmithNotchHingeFun(E,j,s1c,s2c,j,F);
            b=b+1;
            j=j-scount;
        end
        subplot(height,width,num)
        plot(Disp,k)
        xlabel('Displacement (mm)')
        ylabel('Stiffness (N/mm)')
        title('Pareto Curve for Smith Notch Hinge')
    end
end

```

Figure A 64. Complete MATLAB function used to build Pareto curves for a translational joint set.

```

function [ fig ] = ParetoFunR_YYY( array,E,G,s1,s2,F,nfig )

    s1c=s1;
    s2c=s2;
    scount=abs(s2-s1)/10;
    Disp=[0];
    k=[0];

    sizea=size(array);
    nsubplot=sizea(1);
    width=ceil(nsubplot/2);
    height=3;
    num=1;

    fig=figure(nfig);
    b=1;
    a=ismember(1,array);
    if a == 1
        %pareto for joint 1
        for i=s1:scount:s2
            [Disp(b),k(b)]=JensenCrossAxisFun(E,s1c,i,i,s2c,F);
            b=b+1;
        end
        subplot(height,width,num)
        plot(Disp,k)
        xlabel('Displacement (degrees)')
        ylabel('Stiffness (Nmm/degree)')
        title('Pareto Curve for Jensen Cross Axis')
        num=num+1;
    end

    b=1;
    a=ismember(2,array);
    if a == 1
        %pareto for joint 2

```

```

j=s2;
for i=s1:scount:s2
    [Disp(b),k(b)]=LobontiuCornerFillettedFun(E,i,i,i,s2c,F);
    b=b+1;
    j=j-scount;
end
subplot(height,width,num)
plot(Disp,k)
xlabel('Displacement (degrees)')
ylabel('Stiffness (Nmm/degree)')
title('Pareto Curve for Lobontiu Corner Filletted')
num=num+1;
end

b=1;
a=ismember(3,array);
if a == 1
    %pareto for joint 3
    j=s2;
    for i=s1:scount:s2
        [Disp(b),k(b)]=LobontiuSymmetricCircularFun(E,s1c,i,s2c,F);
        b=b+1;
        j=j-scount;
    end
    subplot(height,width,num)
    plot(Disp,k)
    xlabel('Displacement (degrees)')
    ylabel('Stiffness (Nmm/degree)')
    title('Pareto Curve for Lobontiu Symmetric Circular')
    num=num+1;
end

b=1;
a=ismember(4,array);
if a == 1
    %pareto for joint 4
    j=s2;
    for i=s1:scount:s2
        [Disp(b),k(b)]=LobontiuSymmetricNotchFun(E,s1c,i,s2c,F);
        b=b+1;
        j=j-scount;
    end
    subplot(height,width,num)
    plot(Disp,k)
    xlabel('Displacement (degrees)')
    ylabel('Stiffness (Nmm/degree)')
    title('Pareto Curve for Lobontiu Symmetric Notch')
    num=num+1;
end

b=1;
a=ismember(5,array);
if a == 1

```

```

    %pareto for joint 5
    j=s2;
    for i=s1:scount:s2
[Disp(b),k(b)]=RotationallySymmetricLeafHingeFun(E,s1c,i,F);
        b=b+1;
        j=j-scount;
    end
    subplot(height,width,num)
    plot(Disp,k)
    xlabel('Displacement (degrees)')
    ylabel('Stiffness (Nmm/degree)')
    title('Pareto Curve for Rotationally Symmetric Leaf Hinge')
    num=num+1;
end

b=1;
a=ismember(6,array);
if a == 1
    %pareto for joint 6
    for i=s1:scount:s2
        [Disp(b),k(b)]=SmithAnnulusFun(E,G,i,i,F);
        b=b+1;
    end
    subplot(height,width,num)
    plot(Disp,k)
    xlabel('Displacement (degrees)')
    ylabel('Stiffness (Nmm/degree)')
    title('Pareto Curve for Smith Annulus')
    num=num+1;
end

b=1;
a=ismember(7,array);
if a == 1
    %pareto for joint 7
    for i=s1:scount:s2
        [Disp(b),k(b)]=SmithCartwheelFun(E,s1c,i,s2c,F);
        b=b+1;
    end
    subplot(height,width,num)
    plot(Disp,k)
    xlabel('Displacement (degrees)')
    ylabel('Stiffness (Nmm/degree)')
    title('Pareto Curve for Smith Cartwheel')
    num=num+1;
end

b=1;
a=ismember(8,array);
if a == 1
    %pareto for joint 8
    j=s2;

```

```

    for i=s1:scount:s2
        [Disp(b), k(b)] = SmithCruciformFun(G, slc, j, slc, F);
        b=b+1;
        j=j-scount;
    end
    subplot(height,width,num)
    plot(Disp,k)
    xlabel('Displacement (degrees)')
    ylabel('Stiffness (Nmm/degree)')
    title('Pareto Curve for Smith Cruciform')
    num=num+1;
end

b=1;
a=ismember(9,array);
if a == 1
    %pareto for joint 9
    for i=s1:scount:s2
        [Disp(b), k(b)] = SmithTwoAxisFun(E, i, slc, F);
        b=b+1;
    end
    subplot(height,width,num)
    plot(Disp,k)
    xlabel('Displacement (degrees)')
    ylabel('Stiffness (Nmm/degree)')
    title('Pareto Curve for Smith Two Axis')
    num=num+1;
end

b=1;
a=ismember(10,array);
if a == 1
    %pareto for joint 10
    for i=s1:scount:s2
        [Disp(b), k(b)] = TangSymmetricCircularFun(E, i, slc, s2c, F);
        b=b+1;
    end
    subplot(height,width,num)
    plot(Disp,k)
    xlabel('Displacement (degrees)')
    ylabel('Stiffness (Nmm/degree)')
    title('Pareto Curve for Tang Symmetric Circular')
    num=num+1;
end

b=1;
a=ismember(11,array);
if a == 1
    %pareto for joint 11

    for i=s1:scount:s2
        [Disp(b), k(b)] = TreaseConceptFun(G, slc, i, slc, F);
        b=b+1;
    end
end

```

```

end
subplot(height,width,num)
plot(Disp,k)
xlabel('Displacement (degrees)')
ylabel('Stiffness (Nmm/degree)')
title('Pareto Curve for Trease Concept')
num=num+1;
end

b=1;
a=ismember(12,array);
if a == 1
    %pareto for joint 12
    j=s2;
    for i=s1:scount:s2
        [Disp(b),k(b)]=VShapeFun(E,s1c,i,i,i,s2c,F);
        b=b+1;
        j=j-scount;
    end
    subplot(height,width,num)
    plot(Disp,k)
    xlabel('Displacement (degrees)')
    ylabel('Stiffness (Nmm/degree)')
    title('Pareto Curve for V Shape Flexure')
    num=num+1;
end

b=1;
a=ismember(13,array);
if a == 1
    %pareto for joint 13
    j=s2;
    for i=s1:scount:s2
        [Disp(b),k(b)]=KyusojinRotational6R2Fun(E,s1c,i,s2c,F);
        b=b+1;
        j=j-scount;
    end
    subplot(height,width,num)
    plot(Disp,k)
    xlabel('Displacement (degrees)')
    ylabel('Stiffness (Nmm/degree)')
    title('Pareto Curve for Kyusojin Rotational 6R2')
    num=num+1;
end

b=1;
a=ismember(14,array);
if a == 1
    %pareto for joint 14
    for i=s1:scount:s2
        [Disp(b),k(b)]=ConventionalSplitTubeFun(G,s1c,i,s2c,F);
        b=b+1;
    end
end

```

```
subplot(height,width,num)
plot(Disp,k)
xlabel('Displacement (degrees)')
ylabel('Stiffness (Nmm/degree)')
title('Pareto Curve for Conventional Split Tube')
end
end
```

Figure A 65. Complete MATLAB function used to build Pareto curves for a rotational joint set.

APPENDIX E. TEST CASES USED TO VALIDATE SELECTION ALGORITHM

E.1 Test Case 1

The values used for Test Case 1 can be seen in Table A 1. The GUI showing the input values, output Pareto curves, and MATLAB text output are shown in Figure A 66, Figure A 67, and Figure A 68, respectively.

Table A 1. Input values for Test Case 1.

Test Case	Type of Joint	Range of Motion [mm or degrees]	Stiffness [N/mm or Nmm/degree]	Size Constraints [mm]	Applied Force [N]	Material
1	Translational	5	7	4.1 - 41.5	5	ABS

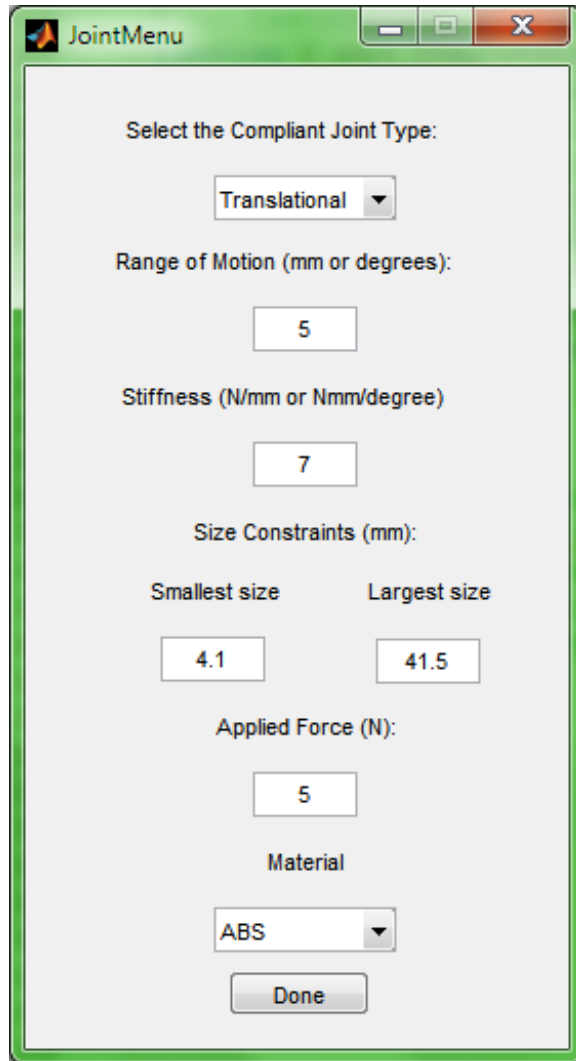


Figure A 66. GUI inputs used for Test Case 1.

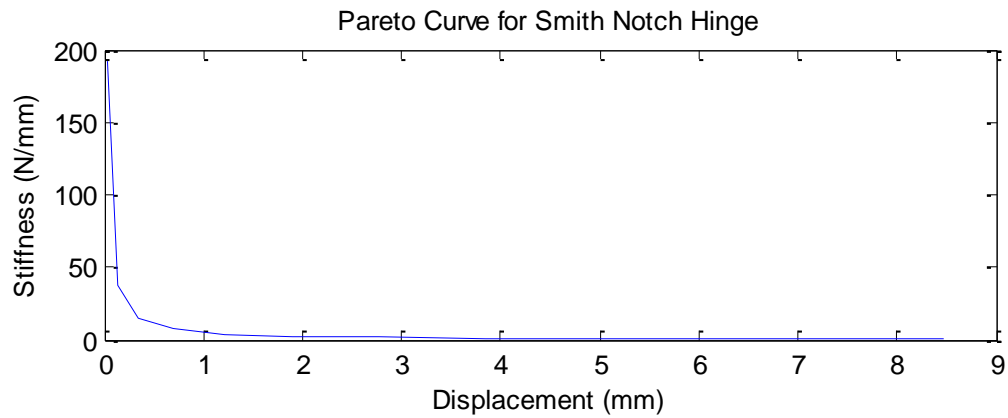


Figure A 67. Pareto curve outputs for Test Case 1.

```
>> JointMenu
Translational Joints are selected
RoM has a value, 5.00 mm.
K has a value, 7.00 N/mm.
F has a value, 5.00 N.

Possible joints for this user input are:
SmithNotchHinge
```

Figure A 68. MATLAB text output for Test Case 1.

E.2 Test Case 2

The values used for Test Case 2 can be seen in Table A 2. The GUI showing the input values is shown in Figure A 69. Since this input does not have a force applied, a generic force set of 0.1 N, 1 N, and 10 N are used to show range of motion possibilities. In this test case, no results are returned for the first two applied forces, and as such, the Pareto curve figures that would be generated are blank. They are omitted for this reason. The Pareto curves for a force applied of 10 N are shown in Figure A 70. The MATLAB text output is shown in Figure A 71.

Table A 2. Input values for Test Case 2.

Test Case	Type of Joint	Range of Motion [mm or degrees]	Stiffness [N/mm or Nmm/degree]	Size Constraints [mm]	Applied Force [N]	Material
2	Rotational	10	3	1.2 - 29.2	-	Aluminum

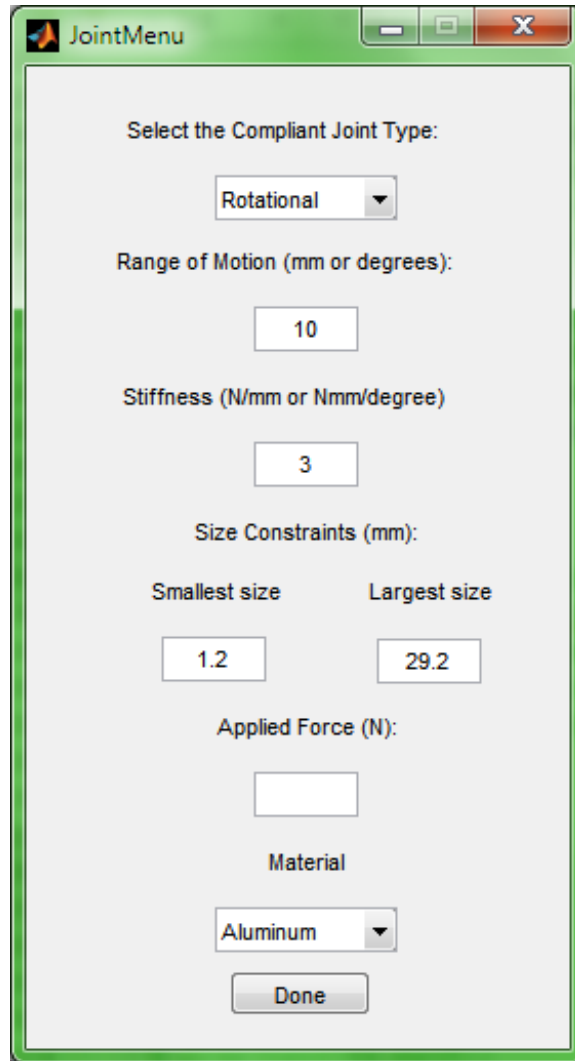


Figure A 69. GUI inputs used for Test Case 2.

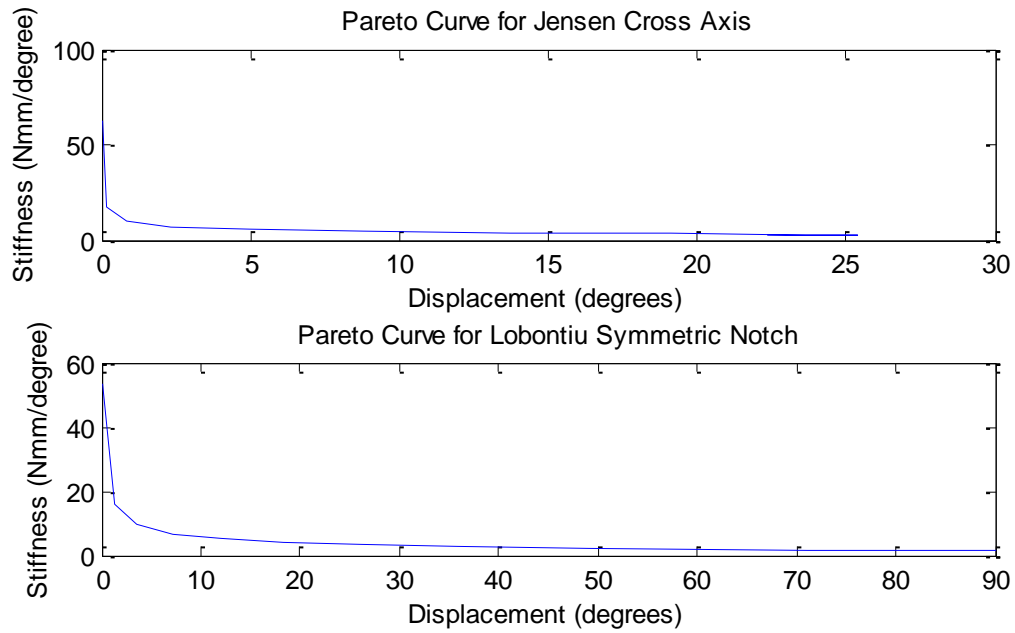


Figure A 70. Pareto curve outputs for Test Case 2.

```

>> JointMenu
Rotational Joints are selected.
RoM has a value, 10.00 degrees.
K has a value, 3.00 N/mm.
F does not have a value (na)

Possible joints for this user input using a force of 0.100000 N are:

Possible joints for this user input using a force of 1.000000 N are:

Possible joints for this user input using a force of 10.000000 N are:
JensenCrossAxis
LobontiuSymmetricNotch

```

Figure A 71. MATLAB text output for Test Case 2.

E.3 Test Case 3

The values used for Test Case 3 can be seen in Table A 3. The GUI showing the input values, output Pareto curves, and MATLAB text output are shown in Figure A 72, Figure A 73, and Figure A 74, respectively.

Table A 3. Input values for Test Case 3.

Test Case	Type of Joint	Range of Motion [mm or degrees]	Stiffness [N/mm or Nmm/degree]	Size Constraints [mm]	Applied Force [N]	Material
3	Translational	2	-	1.7 - 27.5	3	ABS

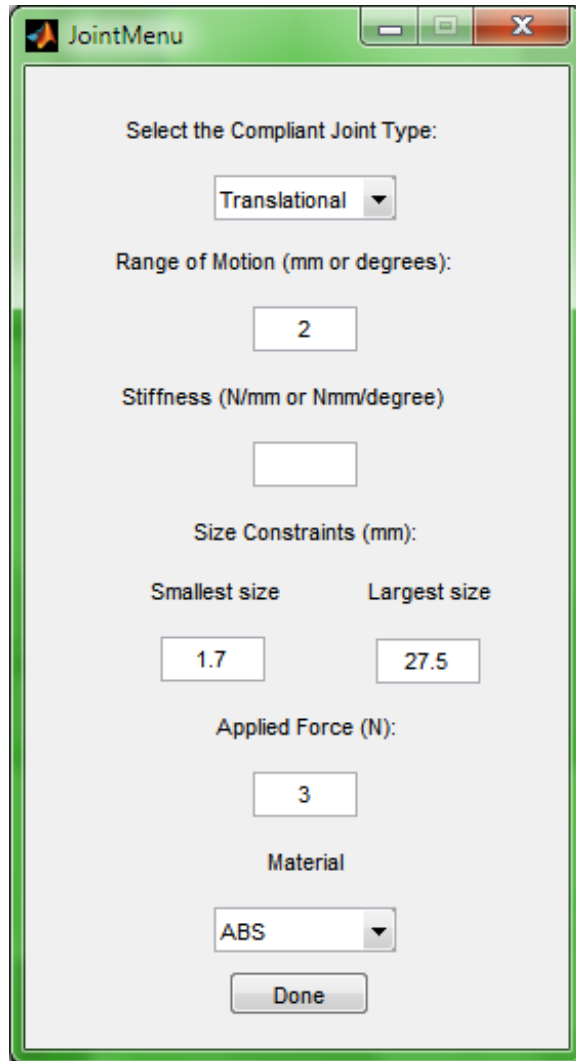


Figure A 72. GUI inputs used for Test Case 3.

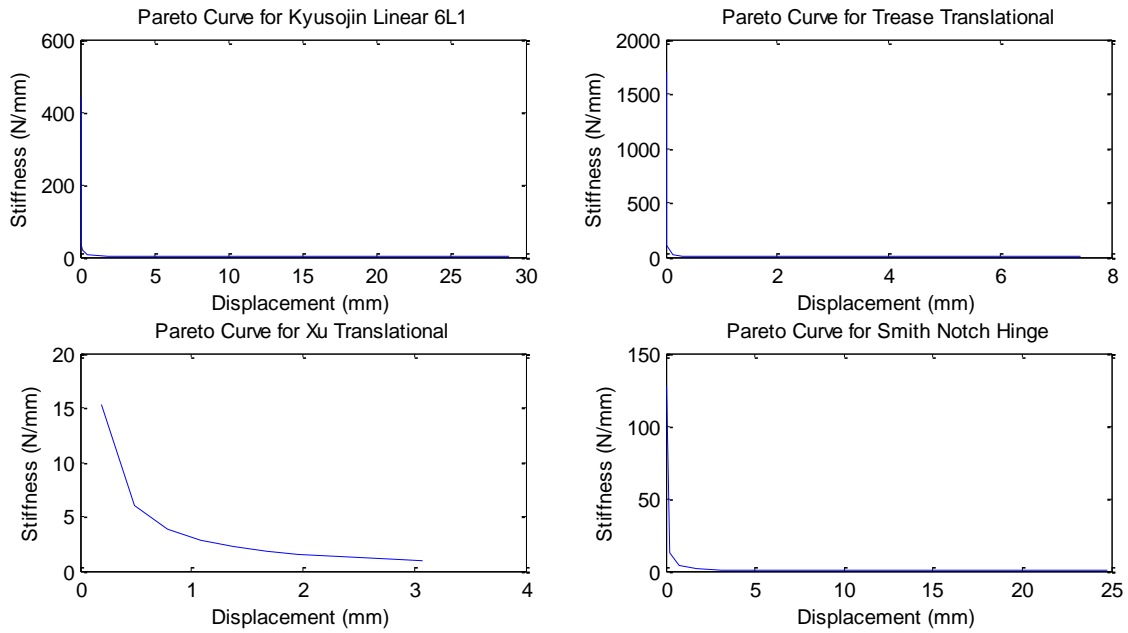


Figure A 73. Pareto curve outputs for Test Case 3.

```
>> JointMenu
Translational Joints are selected
RoM has a value, 2.00 mm.
k does not have a value (na)
F has a value, 3.00 N.

Possible joints for this user input are:
KyusojinLinear6L1
TreaseTranslational
XuTranslational
SmithNotchHinge
```

Figure A 74. MATLAB text output for Test Case 3.

E.4 Test Case 4

The values used for Test Case 4 can be seen in Table A 4. The GUI showing the input values is shown in Figure A 75. Since this input does not have a force applied, a generic force set of 0.1 N, 1 N, and 10 N are used to show range of motion possibilities. The Pareto curves for a force applied of 0.1N, 1 N, and 10 N are shown in Figure A 76, Figure A 77, and Figure A 78, respectively. The MATLAB text output is shown in Figure A 79.

Table A 4. Input values for Test Case 4.

Test Case	Type of Joint	Range of Motion [mm or degrees]	Stiffness [N/mm or Nmm/degree]	Size Constraints [mm]	Applied Force [N]	Material
4	Rotational	4	-	2.5 - 45.9	-	ABS

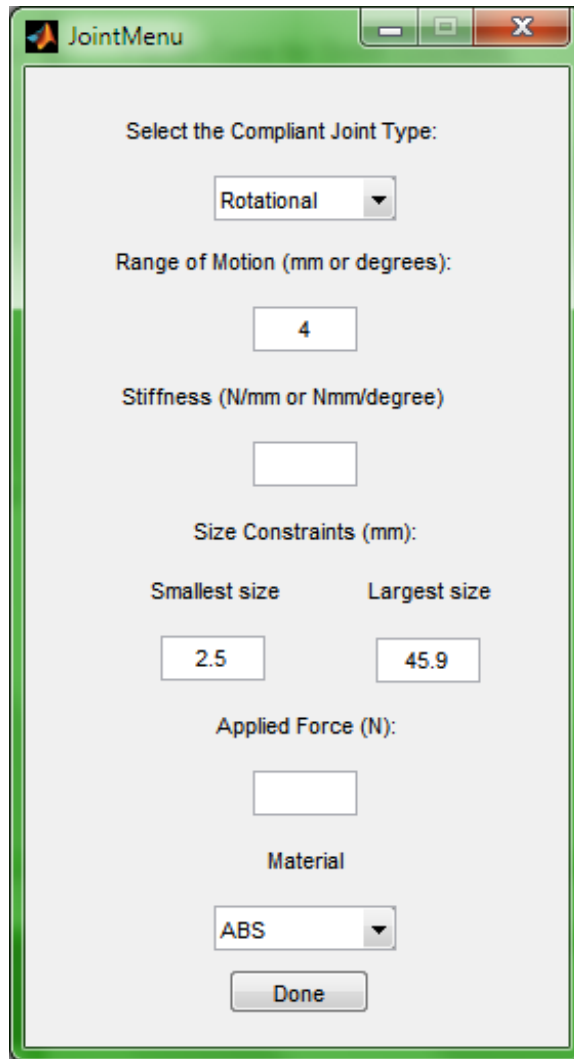


Figure A 75. GUI inputs used for Test Case 4.

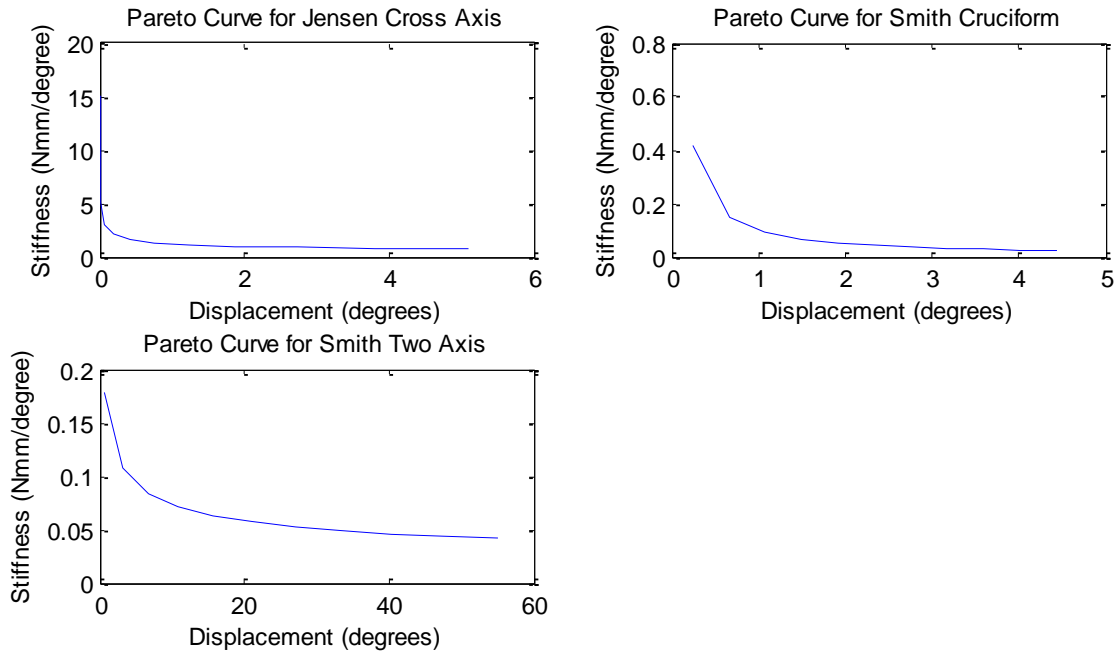


Figure A 76. Pareto curve outputs for Test Case 4, using an applied force of 0.1 N.

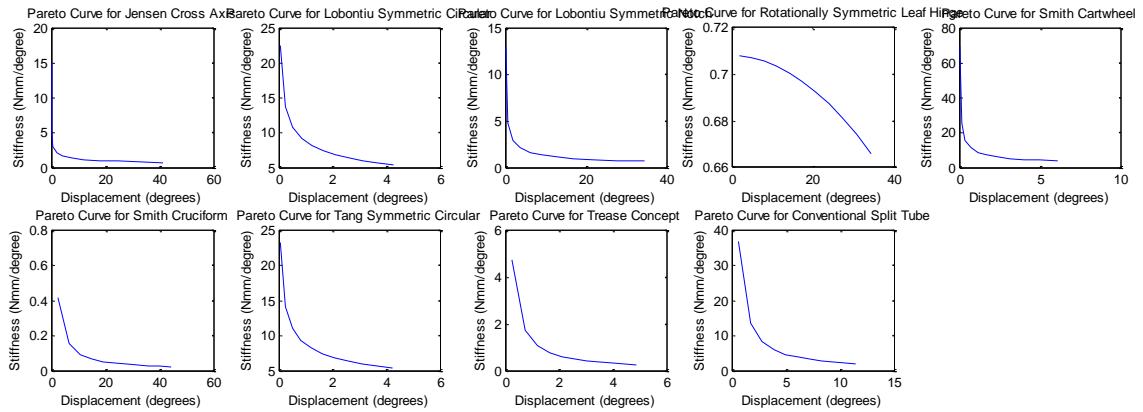


Figure A 77. Pareto curve outputs for Test Case 4, using an applied force of 1 N.

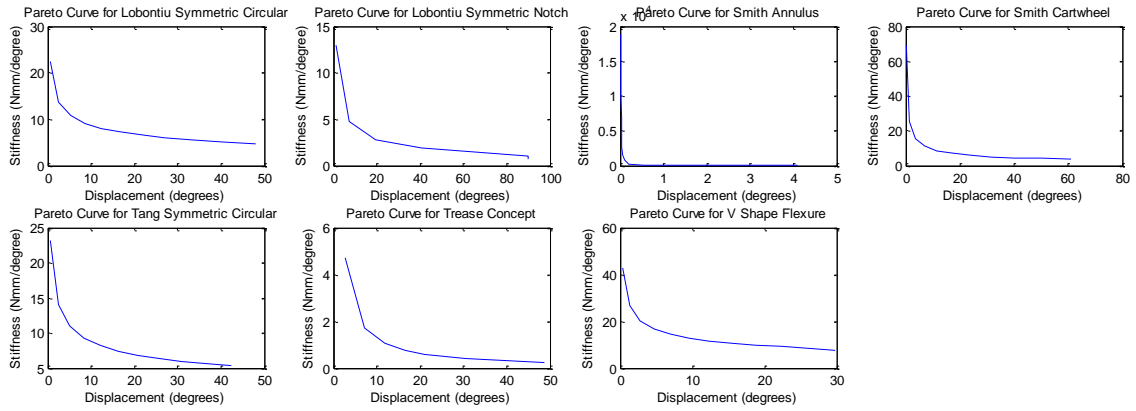


Figure A 78. Pareto curve outputs for Test Case 4, using an applied force of 10 N.

```
>> JointMenu
JointMenu
Rotational Joints are
selected.
RoM has a value, 4.00
degrees.
K does not have a value
(na)
F does not have a value
(na)

Possible joints for this
user input using a force
of 0.100000 N are:
JensenCrossAxis
SmithCruciform
SmithTwoAxis

Possible joints for this
user input using a force
of 1.000000 N are:
JensenCrossAxis
LobontiuSymmetricCircular
LobontiuSymmetricNotch
RotationallySymmetric
SmithCartwheel
SmithCruciform
TangSymmetricCircular
TreaseConcept
ConventionalSplitTube

Possible joints for this
user input using a force
of 10.000000 N are:
LobontiuSymmetricCircular
LobontiuSymmetricNotch
SmithAnnulus
SmithCartwheel
TangSymmetricCircular
TreaseConcept
VShape
```

Figure A 79. MATLAB text output for Test Case 4.

E.5 Test Case 5

The values used for Test Case 5 can be seen in Table A 5. The GUI showing the input values, output Pareto curves, and MATLAB text output are shown in Figure A 80, Figure A 81, and Figure A 82, respectively.

Table A 5. Input values for Test Case 5.

Test Case	Type of Joint	Range of Motion [mm or degrees]	Stiffness [N/mm or Nmm/degree]	Size Constraints [mm]	Applied Force [N]	Material
5	Translational	-	1	1.3 - 14.3	6	PLA

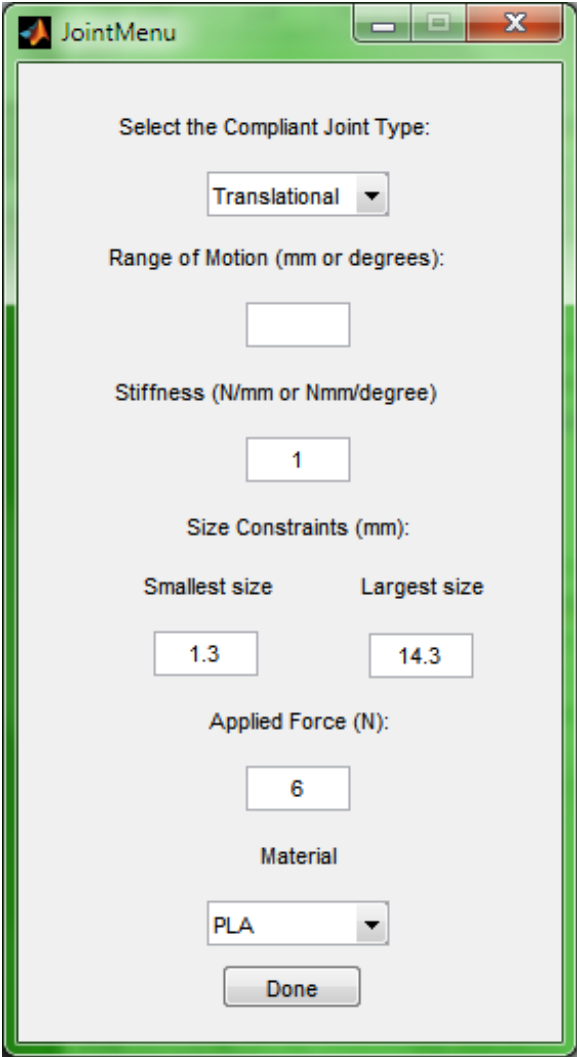


Figure A 80. GUI inputs used for Test Case 5.

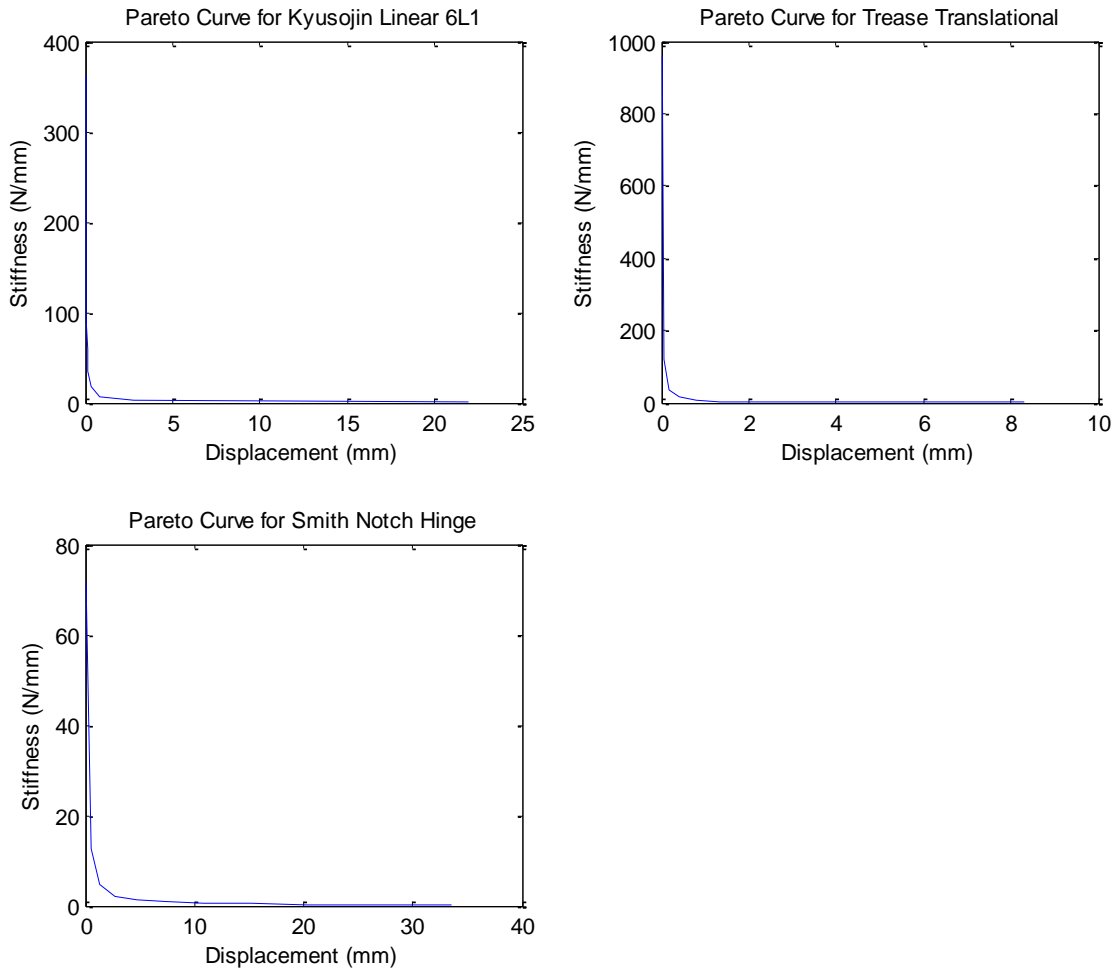


Figure A 81. Pareto curve outputs for Test Case 5.

```
>> JointMenu
Translational Joints are selected
RoM does not have a value (na)
K has a value, 1.00 N/mm.
F has a value, 6.00 N.

Possible joints for this user input are:
KyusojinLinear6L1
TreaseTranslational
SmithNotchHinge
```

Figure A 82. MATLAB text output for Test Case 5.

E.6 Test Case 6

The values used for Test Case 6 can be seen in Table A 6. The GUI showing the input values is shown in Figure A 83. Since this input does not have a force applied, a generic force set of 0.1 N, 1 N, and 10 N are used to show range of motion possibilities. The Pareto curves for a force applied of 0.1N, 1 N, and 10 N are shown in Figure A 84, Figure A 85, and Figure A 86, respectively. The MATLAB text output is shown in Figure A 87.

Table A 6. Input values for Test Case 6.

Test Case	Type of Joint	Range of Motion [mm or degrees]	Stiffness [N/mm or Nmm/degree]	Size Constraints [mm]	Applied Force [N]	Material
6	Rotational	-	2	3 - 37.9	-	Aluminum

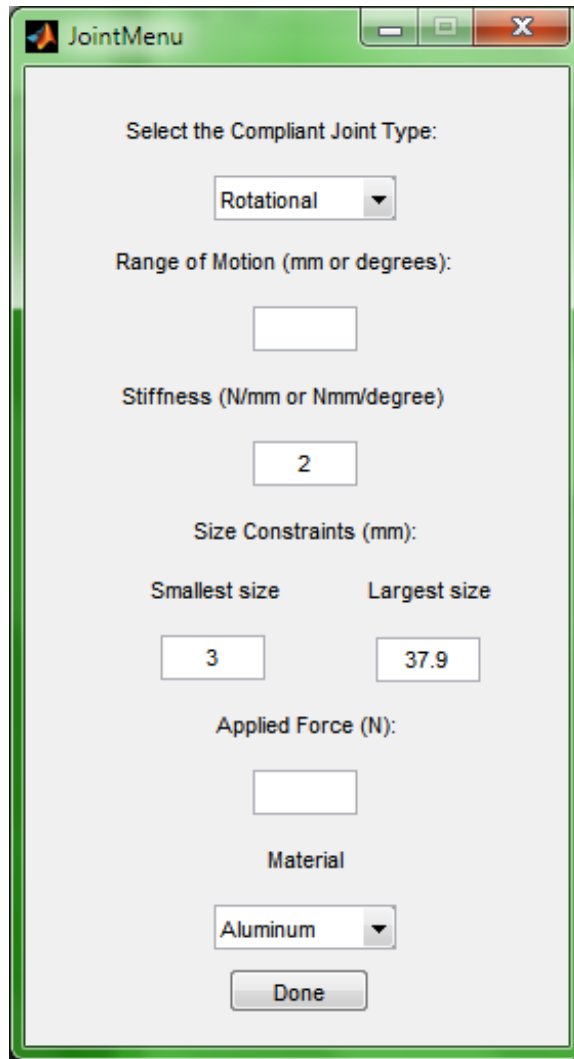


Figure A 83. GUI inputs used for Test Case 6.

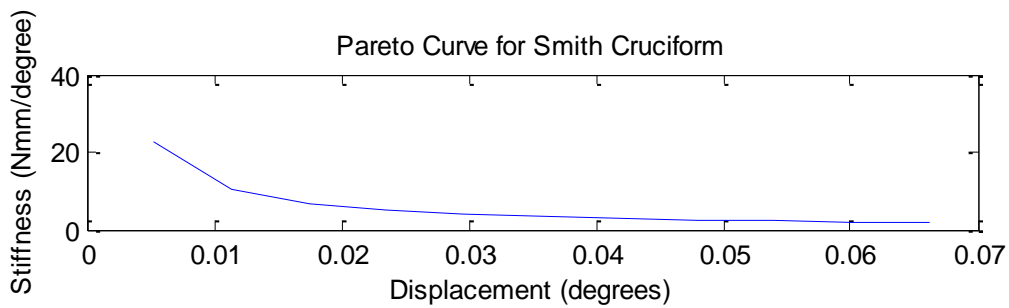


Figure A 84. Pareto curve outputs for Test Case 6, using an applied force of 0.1 N.

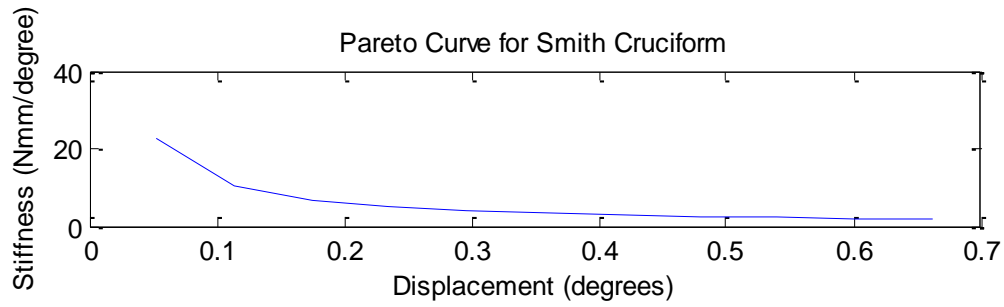


Figure A 85. Pareto curve outputs for Test Case 6, using an applied force of 1 N.

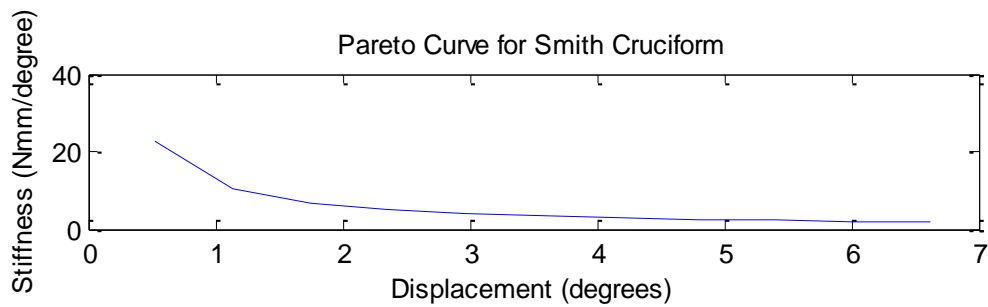


Figure A 86. Pareto curve outputs for Test Case 6, using an applied force of 10 N.

```
>> JointMenu
Rotational Joints are selected.
RoM does not have a value (na)
K has a value, 2.00 Nmm/degree.
F does not have a value (na)

Possible joints for this user input using a force of 0.100000 N are:
SmithCruciform

Possible joints for this user input using a force of 1.000000 N are:
SmithCruciform

Possible joints for this user input using a force of 10.000000 N are:
SmithCruciform
```

Figure A 87. MATLAB text output for Test Case 6.

E.7 Test Case 7

The values used for Test Case 7 can be seen in Table A 7. The GUI showing the input values, output Pareto curves, and MATLAB text output are shown in Figure A 88, Figure A 89, and Figure A 90, respectively.

Table A 7. Input values for Test Case 7.

Test Case	Type of Joint	Range of Motion [mm or degrees]	Stiffness [N/mm or Nmm/degree]	Size Constraints [mm]	Applied Force [N]	Material
7	Translational	-	-	2.4 - 19.0	2	PLA

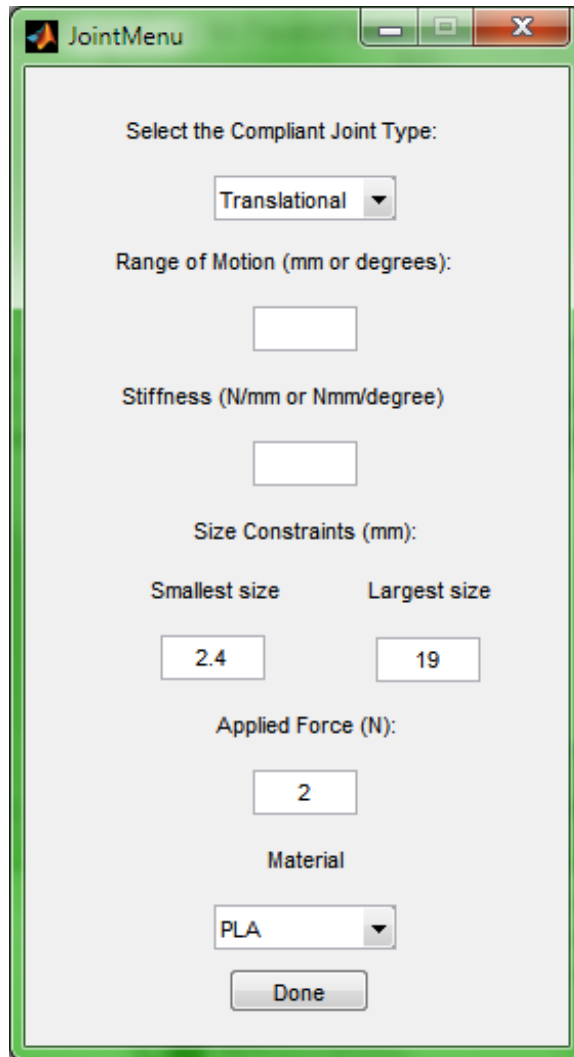


Figure A 88. GUI inputs used for Test Case 7.

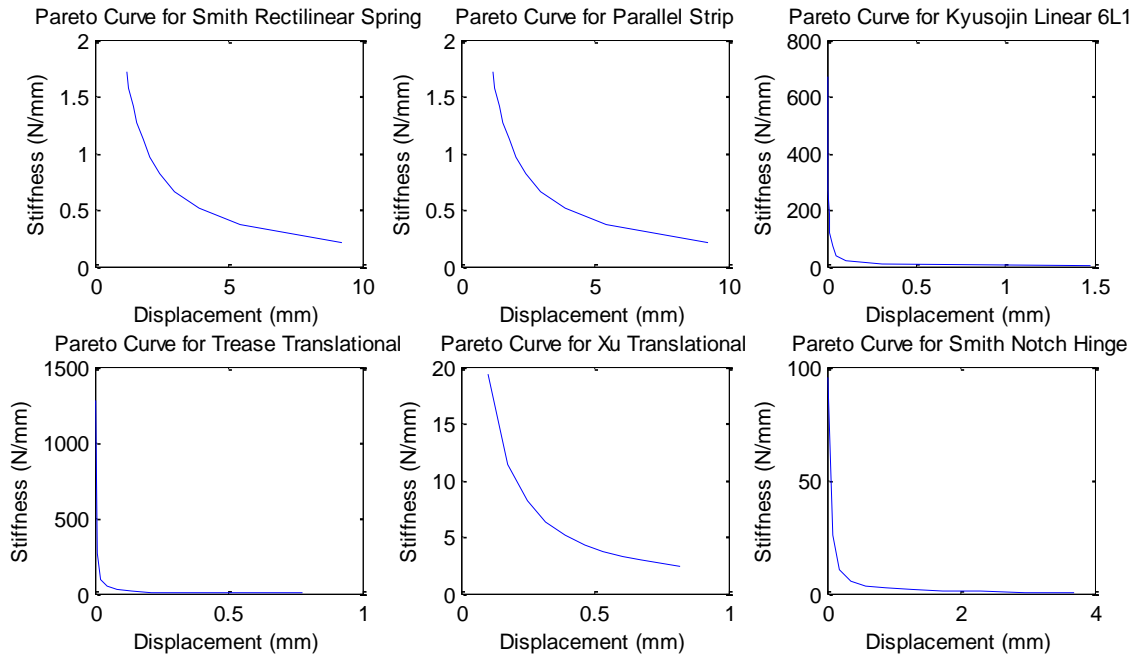


Figure A 89. Pareto curve outputs for Test Case 7.

```
>> JointMenu
Translational Joints are selected
RoM does not have a value (na)
K does not have a value (na)
F has a value, 2.00 N.

The following joints are organized by RoM,
Largest to smallest, for the given force, 2.000000 N :
ParallelStrip with 1.1658 mm displacement.
SmithRectilinear with 1.1658 mm displacement.
XuTranslational with 0.1033 mm displacement.
SmithNotchHinge with 0.0210 mm displacement.
KyusojinLinear6L1 with 0.0030 mm displacement.
TreaseTranslational with 0.0016 mm displacement.
```

Figure A 90. MATLAB text output for Test Case 7.

E.8 Test Case 8

The values used for Test Case 3 can be seen in Table A 8. The GUI showing the input values and MATLAB text output are shown in Figure A 91 and Figure A 92, respectively. It should be noted that there is no Pareto curve output for this selection of inputs. This is because with no initial values, a large amount of output is shown to the user. To prevent the data from being displayed misleadingly, it is recommended the user add some additional requirements and use the selection algorithm again.

Table A 8. Input values for Test Case 8.

Test Case	Type of Joint	Range of Motion [mm or degrees]	Stiffness [N/mm or Nmm/degree]	Size Constraints [mm]	Applied Force [N]	Material
8	Rotational	-	-	1.8 - 28.4	-	Aluminum

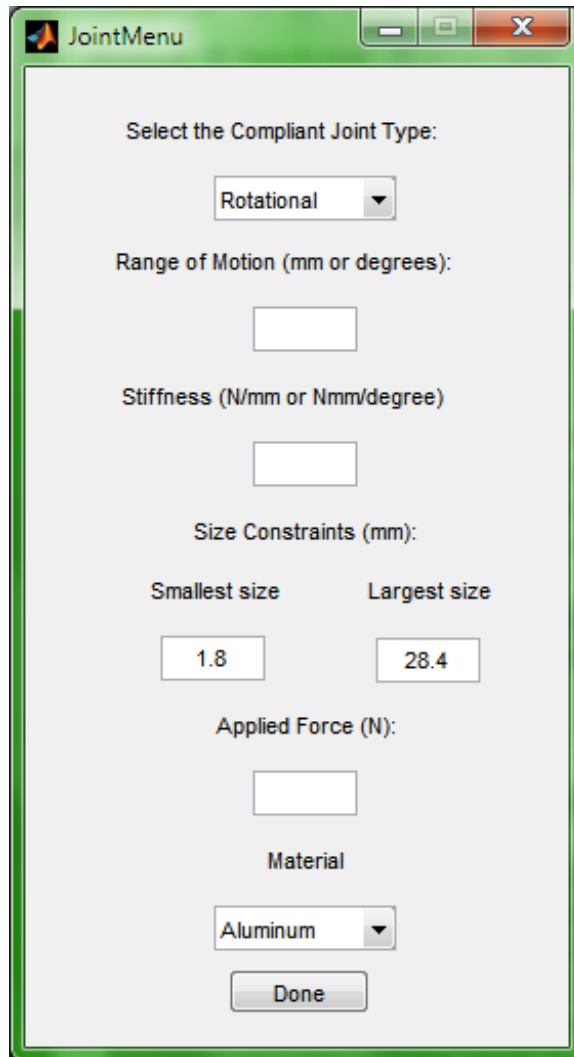


Figure A 91. GUI inputs used for Test Case 8.

```
>> JointMenu
Rotational Joints are selected.
RoM does not have a value (na)
K does not have a value (na)
F does not have a value (na)

Possible joints for this user input using a force of 0.100 N,
organized by RoM are:
SmithTwoAxis with 3.0006 degrees displacement.
SmithCruciform with 0.2297 degrees displacement.
LobontiuSymmetricNotch with 0.1911 degrees displacement.
RotationallySymmetric with 0.1905 degrees displacement.
JensenCrossAxis with 0.1862 degrees displacement.
ConventionalSplitTube with 0.0595 degrees displacement.
SmithCartwheel with 0.0358 degrees displacement.
```

LobontiuSymmetricCircular with 0.0270 degrees displacement.
TangSymmetricCircular with 0.0269 degrees displacement.
TreaseConcept with 0.0252 degrees displacement.
VShape with 0.0189 degrees displacement.
SmithAnnulus with 0.0025 degrees displacement.
Kyusojin6R2 with 0.0001 degrees displacement.
LobontiuCornerFilletted with 0.0000 degrees displacement.

Possible joints for this user input using a force of 1.000 N,
organized by RoM are:

ConventionalSplitTube with 30.0057 degrees displacement.
SmithCartwheel with 2.2967 degrees displacement.
VShape with 1.9114 degrees displacement.
TreaseConcept with 1.9054 degrees displacement.
LobontiuCornerFilletted with 1.8610 degrees displacement.
SmithTwoAxis with 0.5947 degrees displacement.
LobontiuSymmetricCircular with 0.3583 degrees displacement.
SmithAnnulus with 0.2702 degrees displacement.
JensenCrossAxis with 0.2689 degrees displacement.
RotationallySymmetric with 0.2524 degrees displacement.
LobontiuSymmetricNotch with 0.1889 degrees displacement.
TangSymmetricCircular with 0.0247 degrees displacement.
Kyusojin6R2 with 0.0001 degrees displacement.
SmithCruciform with 0.0001 degrees displacement.

Possible joints for this user input using a force of 10.000 N,
organized by RoM are:

SmithTwoAxis with 300.0568 degrees displacement.
LobontiuSymmetricCircular with 22.9673 degrees displacement.
LobontiuSymmetricNotch with 19.4838 degrees displacement.
RotationallySymmetric with 19.4201 degrees displacement.
SmithCruciform with 17.3258 degrees displacement.
ConventionalSplitTube with 5.9473 degrees displacement.
SmithAnnulus with 3.5832 degrees displacement.
TangSymmetricCircular with 2.7028 degrees displacement.
LobontiuCornerFilletted with 2.6886 degrees displacement.
TreaseConcept with 2.5238 degrees displacement.
VShape with 1.8890 degrees displacement.
JensenCrossAxis with 0.2469 degrees displacement.
Kyusojin6R2 with 0.0015 degrees displacement.
SmithCartwheel with 0.0001 degrees displacement.

Figure A 92. MATLAB text outputs for Test Case 8.

APPENDIX F. ADDITIONAL APPROACHES CONSIDERED

During the development of this research, multiple approaches were considered that did not provide results towards the development of the compliant joint repository and selection method. Two techniques of representing the models of compliant joints were partially implemented.

One technique was the development of models within Modelica, an acausal modeling language, where models could be solved iteratively based on user requirements. This method of modeling allowed all inputs to be non-required, but the solution space provided was difficult to manage due to size. Models developed within Modelica were also difficult to relate to one another, since no normalization had been implemented at the time.

Another technique was the development of models within Phoenix Integration's ModelCenter. ModelCenter had a few advantages over the current methodology, foremost was integrated optimization of compliant joints. Each individual model could be optimized to achieve precisely the requirements the user input. However, since each model would have to be optimized to determine if it was a potential solution, this technique was incredibly slow. ModelCenter also had difficulties processing complex logic to determine between different potential outcomes. Each model's optimization had to be pre-set for every combination of potential inputs, which led to a large library of potential optimizations that would be performed each time the selection algorithm was used.