



Theses and Dissertations

---

2003-12-22

## Establishing A Quantitative Foundation for Exactly Constrained Design

Alisha M. Hammond  
*Brigham Young University - Provo*

Follow this and additional works at: <https://scholarsarchive.byu.edu/etd>



Part of the [Mechanical Engineering Commons](#)

---

### BYU ScholarsArchive Citation

Hammond, Alisha M., "Establishing A Quantitative Foundation for Exactly Constrained Design" (2003). *Theses and Dissertations*. 115.  
<https://scholarsarchive.byu.edu/etd/115>

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact [scholarsarchive@byu.edu](mailto:scholarsarchive@byu.edu), [ellen\\_amatangelo@byu.edu](mailto:ellen_amatangelo@byu.edu).

ESTABLISHING A QUANTITATIVE FOUNDATION FOR  
EXACTLY CONSTRAINED DESIGN

by

Alisha M. Hammond

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

Brigham Young University

April 2004

Copyright © 2004 Alisha M. Hammond

All Rights Reserved

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Alisha M. Hammond

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

\_\_\_\_\_

Date

\_\_\_\_\_

Alan R. Parkinson, Chair

\_\_\_\_\_

Date

\_\_\_\_\_

Kenneth W. Chase

\_\_\_\_\_

Date

\_\_\_\_\_

Carl D. Sorensen

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Alisha M. Hammond in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

---

Date

---

Alan R. Parkinson  
Chair, Graduate Committee

Accepted for the Department

---

Brent L. Adams  
Graduate Coordinator

Accepted for the College

---

Douglas M. Chabries  
Dean, College of Engineering and Technology

## ABSTRACT

### ESTABLISHING A QUANTITATIVE FOUNDATION FOR EXACTLY CONSTRAINED DESIGN

Alisha M. Hammond

Department of Mechanical Engineering

Master of Science

Exactly constrained (EC) design is a robust design method which can be used for mechanical assemblies. It entails using the minimum number of constraints to eliminate all desired motion.

While found by some engineers in industry to have many benefits (including robust assembly, no binding or play, ease of assembly, and the ability to tolerate the wear of parts), EC designs remain somewhat unrecognized by academia. One reason for this minimal exposure may be the lack of a quantitative foundation for such designs. This thesis describes the history and current background for EC designs, and it also begins to develop a quantitative foundation for EC design based on several mathematical methods.

EC designs can be analyzed quite simply by understanding that they are statically determinate. Because of this, the equations of equilibrium can be used to validate the rules and the nesting force window that have been defined by Blanding [1999]. In

addition, a generalized method using the equations of equilibrium has been developed in this thesis to analyze an EC design based on the locations of the constraints and to find the nesting force window.

The direct linearization method (DLM) is another mathematical method used to quantify information in an EC design. While EC designs provide many advantages, some EC assemblies may be “better” than others. A quantitative measure of goodness is developed in this thesis using the DLM. The goodness value assigned to each design through this process can either be used to make a decision on an individual design, or it can be used to compare similar EC designs.

Finally, the robust nature of EC design is examined using a Monte Carlo simulation. In general, the results show that EC designs have a higher rate of assembly than similar designs that are over-constrained. They are more robust. In addition, EC designs have lower assembly error than the similarly over-constrained assemblies.

## ACKNOWLEDGMENTS

This thesis would certainly not be complete without formally acknowledging everyone who made it possible. I offer my deepest gratitude to Dr. Alan Parkinson for his constant patience and attention to detail throughout this work. He spent countless hours helping to revise and improve various sections of the thesis, and for that I am grateful. I would also like to acknowledge Dr. Kenneth Chase and Dr. Carl Sorensen who provided key insights and direction at pivotal moments throughout the research. Also, much of the research was made possible through NSF grant DMI 0084880.

I would like to acknowledge my family, and specifically my parents, Michael and Martha Hammond, for their constant support throughout the entire process of this thesis. Because of their wise counsel and experience, I have accomplished many of the goals I set. In addition, countless friends and associates, of which there are too many to name here, must be acknowledged for their constant support and interest.

Finally, I am most grateful for the unique mission of BYU to combine learning “even by study and also by faith.” I am grateful that a wise Father in Heaven has offered such a great opportunity for learning; and without Him and the guidance of the Spirit, this thesis never would have been completed.



## TABLE OF CONTENTS

<b>LIST OF TABLES</b>	.....	<b>xv</b>
<b>LIST OF FIGURES</b>	.....	<b>xvi</b>
<b>CHAPTER 1</b>	<b>MOTIVATION FOR ESTABLISHING A QUANTITATIVE FOUNDATION FOR EXACTLY CONSTRAINED DESIGN.....</b>	<b>1</b>
1.1	INTRODUCTION .....	1
1.2	MOTIVATION FOR THE RESEARCH .....	2
1.3	EXACTLY CONSTRAINED DESIGN.....	4
1.4	ADVANTAGES OF EXACTLY CONSTRAINED DESIGN.....	6
1.5	ADVOCATES FOR EXACTLY CONSTRAINED DESIGN .....	7
1.6	CURRENT PRACTICES.....	8
1.7	OBJECTIVES OF THE THESIS .....	9
1.8	DELIMITATIONS.....	10
1.9	THESIS OVERVIEW .....	10
<b>CHAPTER 2</b>	<b>BACKGROUND AND LITERATURE REVIEW FOR EXACTLY CONSTRAINED DESIGN .....</b>	<b>11</b>
2.1	INTRODUCTION .....	11
2.2	EXACTLY CONSTRAINED DESIGN AS A ROBUST DESIGN METHOD .....	11
2.3	HISTORY OF EXACTLY CONSTRAINED DESIGN.....	12
2.4	BACKGROUND FOR EXACTLY CONSTRAINED DESIGN .....	16
2.5	CURRENT METHODS TO ANALYZE EXACTLY CONSTRAINED DESIGN .....	19
2.5.1	DEFINING THE CONSTRAINTS.....	19

2.5.2	DEFINING THE NESTING FORCE .....	20
2.6	ALTERNATE DESIGNS .....	27
2.6.1	OVER-CONSTRAINED DESIGN.....	28
2.6.2	UNDER-CONSTRAINED DESIGN.....	30
2.7	EXAMPLES OF EXACTLY CONSTRAINED DESIGN.....	31
2.7.1	KELVIN'S KINEMATIC CLAMP .....	31
2.7.2	BLANDING'S 2D BODY ON TWO PINS EXAMPLE.....	32
2.7.3	KRIEGEL'S COPY MACHINE EXAMPLE.....	33
2.8	USING SCREW THEORY FOR CONSTRAINT ANALYSIS.....	38
2.9	THE DIRECT LINEARIZATION METHOD USED FOR VARIATION ANALYSIS OF EC DESIGNS .....	41
2.9.1	CREATE AN ASSEMBLY GRAPH.....	43
2.9.2	LOCATE THE DATUM REFERENCE FRAME FOR EACH PART.....	43
2.9.3	LOCATE KINEMATIC JOINTS AND CREATE DATUM PATHS .....	43
2.9.4	CREATE VECTOR LOOPS.....	44
2.9.5	GENERATE ASSEMBLY EQUATIONS FROM VECTOR LOOPS .....	45
2.9.6	CALCULATE DERIVATIVES AND FORM MATRIX EQUATIONS.....	46
2.9.7	SOLVE FOR ASSEMBLY SENSITIVITIES .....	46
2.10	CONCLUSIONS.....	47
<b>CHAPTER 3</b>	<b>VALIDATION OF THE CURRENT RULES AND METHODS FOR EXACTLY CONSTRAINED DESIGN USING A QUANTITATIVE FOUNDATION.....</b>	<b>49</b>
3.1	INTRODUCTION .....	49
3.2	STATICALLY DETERMINATE ASSEMBLIES.....	50
3.2.1	TWO-FORCE MEMBERS.....	51
3.2.2	THREE-FORCE MEMBERS .....	52

3.2.3	FOUR FORCES IN A SYSTEM .....	53
3.2.4	FIVE OR MORE FORCES IN A SYSTEM.....	54
3.3	VALIDATING THE RULES FOR EXACTLY CONSTRAINED DESIGN .....	55
3.3.1	NO TWO CONSTRAINTS SHOULD BE CO-LINEAR.....	55
3.3.2	NO FOUR CONSTRAINTS ARE IN A SINGLE PLANE.....	58
3.3.3	NO THREE CONSTRAINTS ARE PARALLEL .....	59
3.3.4	NO THREE CONSTRAINTS SHOULD INTERSECT AT A POINT.....	61
3.3.5	SUMMARY OF THE RULES.....	63
3.4	NESTING FORCE WINDOW .....	64
3.4.1	QUANTITATIVE APPROACH TO FIND THE NESTING FORCE WINDOW .....	64
3.4.2	COMPARISON BETWEEN GRAPHICAL AND QUANTITATIVE APPROACHES .....	67
3.5	CONCLUSIONS.....	70
<b>CHAPTER 4</b>	<b>GENERALIZED METHOD TO USE THE EQUATIONS OF EQUILIBRIUM IN EXACTLY CONSTRAINED DESIGN .....</b>	<b>71</b>
4.1	INTRODUCTION .....	71
4.2	INITIALIZING THE ANALYSIS FOR EC DESIGN .....	72
4.3	GENERALIZED METHOD TO FIND THE NESTING FORCE WINDOW .....	75
4.3.1	DEFINITION OF TRANSITION POINTS .....	75
4.3.2	PRESENTATION OF THE GENERALIZED METHOD .....	78
4.4	GENERALIZED METHOD TO QUANTITATIVELY ANALYZE EC DESIGN .....	89
4.4.1	GENERAL METHOD TO INSPECT EC DESIGN.....	89
4.4.2	SINGULARITY OF THE [C] MATRIX.....	95
4.4.2.1	A ROW OF ZEROS.....	95

	4.4.2.2 A COLUMN OF ZEROS.....	96
	4.4.2.3 LINEAR DEPENDENCE.....	97
	4.4.3 GENERAL METHOD TO DESIGN AND MAKE PREDICTIONS FOR EC ASSEMBLIES .....	98
	4.4.4 MORE SIMPLE EXAMPLES.....	102
4.5	TRADEOFF BETWEEN THE REACTION FORCES AND THE NESTING FORCE WINDOW.....	110
<b>CHAPTER 5</b>	<b>A QUANTITATIVE MEASURE OF “GOODNESS” IN AN EXACTLY CONSTRAINED DESIGN .....</b>	<b>115</b>
5.1	INTRODUCTION .....	115
5.2	THE GOODNESS OF AN EXACTLY CONSTRAINED DESIGN.....	116
5.3	USING SCREW THEORY TO QUANTITATIVELY MEASURE GOODNESS .....	118
5.4	USING THE DIRECT LINEARIZATION METHOD (DLM) TO PROVIDE A QUANTITATIVE GOODNESS CRITERIA.....	120
5.4.1	SIGNIFICANCE OF PARTIAL DERIVATIVES IN THE DLM.....	122
5.4.2	[B] MATRIX CONTRIBUTIONS .....	128
	5.4.2.1 A ROW OF ZEROS.....	129
	5.4.2.2 A COLUMN OF ZEROS.....	131
	5.4.2.3 LINEAR DEPENDENCE.....	131
	5.4.2.4 USING THE [B] MATRIX AS A MEASURE OF GOODNESS .....	137
5.4.3	USING ASSEMBLY SENSITIVITIES TO QUANTIFY GOODNESS .....	138
5.4.4	USING THE GOODNESS VALUES FOUND IN THE DLM.....	141
5.4.5	A RETURN TO THE SLOTTED BLOCK EXAMPLE.....	144
5.5	THE [B] MATRIX FROM THE DLM AND THE [C] MATRIX FROM THE EQUATIONS OF EQUILIBRIUM .....	149
5.6	CONCLUSIONS.....	152

<b>CHAPTER 6</b>	<b>USING EXACTLY CONSTRAINED DESIGN AS A ROBUST DESIGN METHOD.....</b>	<b>153</b>
6.1	INTRODUCTION .....	153
6.2	MONTE CARLO SIMULATION TO SHOW THE ROBUST NATURE OF EC DESIGN .....	154
6.2.1	WILL THE DESIGN ASSEMBLE?.....	155
6.2.2	WHAT IS THE OVERALL ERROR?.....	155
6.3	EXAMPLES.....	157
6.3.1	EC BLOCK WITH THREE CONSTRAINTS .....	157
6.3.2	NON-EC BLOCK WITH THREE CONSTRAINTS .....	160
6.3.3	OC BLOCK WITH FOUR CONSTRAINTS .....	161
6.3.4	SUMMARY OF THE BLOCK ASSEMBLIES WITH THREE OR FOUR CONSTRAINTS .....	166
6.3.5	SLOTTED BLOCK ASSEMBLY .....	167
6.3.5.1	SLOT ANGLE: 0° .....	169
6.3.5.2	SLOT ANGLE: 90° .....	169
6.3.5.3	SLOT ANGLE: VARIED .....	170
6.4	CONCLUSIONS.....	172
<b>CHAPTER 7</b>	<b>CONTRIBUTIONS, CONCLUSIONS, AND RECOMMENDATIONS.....</b>	<b>173</b>
7.1	CONTRIBUTIONS OF THIS THESIS.....	173
7.2	CONCLUSIONS OF THIS THESIS.....	175
7.3	RECOMMENDATIONS FOR FUTURE WORK .....	176
<b>APPENDIX A</b>	<b>CONSTRAINT ANALYSIS USING SCREW THEORY .....</b>	<b>181</b>
A.1	FIND THE TRANSFORMATION MATRIX.....	181
A.2	FIND THE TWISTMATRIX FOR EACH FEATURE .....	183
A.3	DETERMINE IF THE ASSEMBLY IS UNDER-CONSTRAINED .....	185

	STEP 1: RECIPROCAL OPERATION APPLIED TO EACH TWIST .....	185
	STEP 2: UNIONIZE THE MATRICES .....	185
	STEP 3: ROW REDUCED ECHELON FORM .....	185
	STEP 4: RECIPROCAL OPERATION APPLIED TO THE WRENCH .....	186
	MATLAB <sup>®</sup> AUTOMATION OF THE PROCESS.....	186
A.4	DETERMINE IF THE ASSEMBLY IS OVER-CONSTRAINED .....	188
	STEP 1: UNIONIZE THE TWISTMATRICES .....	188
	STEP 2: ROW REDUCED ECHELON FORM .....	188
	STEP 3: RECIPROCAL OPERATION APPLIED TO THE TWIST .....	188
	MATLAB <sup>®</sup> AUTOMATION OF THE PROCESS.....	189
A.5	DETAILED EXAMPLES FROM CHAPTER 5 .....	190
	SLOT AT 45 <sup>0</sup> .....	191
	SLOT AT 70 <sup>0</sup> .....	192
	SLOT AT 89.9 <sup>0</sup> .....	193
	SLOT AT 90 <sup>0</sup> .....	194
<b>APPENDIX B</b>	<b>EXCEL<sup>®</sup> ANALYSIS FOR EXAMPLES USING THE EQUATIONS OF EQUILIBRIUM.....</b>	<b>197</b>
B.1	NO TWO CONSTRAINTS SHOULD BE CO-LINEAR ANALYSIS (FIG. 3.3) .....	197
B.2	NO THREE CONSTRAINTS SHOULD INTERSECT AT A POINT ANALYSIS (FIG. 3.10).....	199
B.3	NESTING FORCE WINDOW USING THE EQUATIONS OF EQUILIBRIUM (SECTION 3.4.2) .....	202
<b>APPENDIX C</b>	<b>COMPARING THE GRAPHICAL NESTING FORCE WINDOW TO THE QUANTITATIVE NESTING FORCE WINDOW .....</b>	<b>207</b>
C.1	BRIEF DESCRIPTION AND BASELINE RESULTS .....	207
C.2	FIRST METHOD: EQUATIONS OF EQUILIBRIUM .....	208

C.3	SECOND METHOD: EQUATION OF A LINE TO FIND INTERSECTION/TRANSITION POINTS .....	213
<b>APPENDIX D</b>	<b>DETAILED ANALYSIS USING THE EQUATIONS OF EQUILIBRIUM FOR FIGURE 4.19.....</b>	<b>217</b>
D.1	FORMULAS USED IN EXCEL <sup>®</sup> FOR FIG. 4.19 .....	217
D.2	THE RESULTS FOR FIG. 4.19 .....	218
<b>APPENDIX E</b>	<b>DETAILS FOR THE MONTE CARLO SIMULATION EXAMPLES IN CHAPTER 6.....</b>	<b>221</b>
E.1	ALL BLOCKS WITH THREE CONSTRAINTS (SECTIONS 6.3.1 AND 6.3.2) .....	221
E.2	BLOCK WITH FOUR CONSTRAINTS (SECTION 6.3.3) .....	230
E.3	SLOTTED BLOCK ASSEMBLY (SECTION 6.3.4) .....	238
<b>APPENDIX F</b>	<b>3D JOINTS IN ASSEMBLIES .....</b>	<b>255</b>
<b>APPENDIX G</b>	<b>RECIPROCAL RELATIONSHIP FOR FIG. 5.12.....</b>	<b>257</b>
<b>BIBLIOGRAPHY</b>	<b>.....</b>	<b>259</b>

## LIST OF TABLES

Table 3.1 – Summary of the results for the rules .....	63
Table 4.1 – Transition points along each side of the assembly .....	106
Table 6.1 – Block assembly with different starting points for the constraints .....	160
Table 6.2 – Several over-constrained examples .....	165
Table 6.3 – Table of Monte Carlo results for various slot angles.....	170
Table G.1 – Relationship between the equations of equilibrium and the DLM for the block with three constraints.....	257



## LIST OF FIGURES

Figure 1.1 – Block resting on a table. (a) Front view (b) Top view .....	5
Figure 1.2 – Constraints and nesting forces to make the box immobile.....	5
Figure 2.1 – Examples of kinematic clamps (a) Maxwell’s 3 V’s example (b) Lord Kelvin’s clamp (after Evans [1989]).....	14
Figure 2.2 – 2D object with three degrees of freedom. The object can translate along the x and y-axes, and it can rotate about the z-axis.....	16
Figure 2.3 – 3D object with six degrees of freedom. The object can translate along and rotate about the x, y, and z-axes.....	17
Figure 2.4 – Six kinematic joints for 2D assemblies. The arrows represent the allowable degrees of freedom (after Chase [1999]).....	18
Figure 2.5 – Triangle with three constraints .....	21
Figure 2.6 – Triangle with constraint lines drawn and instant centers defined .....	22
Figure 2.7 – Finding rotation on the left instant center.....	23
Figure 2.8 – Finding rotation on the top right instant center .....	23
Figure 2.9 – Direction of all necessary rotations on each instant center .....	24
Figure 2.10 – Test to see if the nesting force is permitted to pass through the highlighted line. ....	25
Figure 2.11 – Testing a segment in the nesting force window .....	26
Figure 2.12 – The allowable nesting force window.....	27



Figure 3.4 – Block with three constraints .....	56
Figure 3.5 – Reaction force on $C_3$ required to keep the block immobile.....	57
Figure 3.6 – Block with four constraints .....	58
Figure 3.7 – Block assembly with three parallel constraints .....	60
Figure 3.8 – Similar block assembly with three parallel constraints .....	60
Figure 3.9 – Adding an x-constraint to the block .....	61
Figure 3.10 – Triangle assembly with three constraints .....	62
Figure 3.11 – Reaction force on $C_3$ required to keep the block immobile.....	62
Figure 3.12 – Triangle assembly example used to find the nesting force window.....	65
Figure 3.13 – Nesting force window according to the equations of equilibrium .....	67
Figure 3.14 – Finding the nesting force window using the graphical method.....	68
Figure 3.15 – Nesting force window comparison.....	69
Figure 4.1 – Block with 3 constraints to be used for generalized method.....	73
Figure 4.2 – Transition points marked on the triangle assembly from Fig. 3.13.....	76
Figure 4.3 – Transition points shown in a force analysis .....	77
Figure 4.4 – Finding transition points for the bottom surface .....	80
Figure 4.5 – Transition point found along the base .....	82
Figure 4.6 – Finding the transition points for the top surface.....	83
Figure 4.7 – Transition points along the top and bottom surfaces.....	84
Figure 4.8 – Finding the transition points for the right surface .....	85
Figure 4.9 – The block assembly with all transition points .....	86
Figure 4.10 – The nesting force window on the bottom surface. The bolded portion of the line is the unacceptable region. ....	87

Figure 4.11 – The nesting force window for the block assembly. The bolded portion is the unacceptable region of the window. ....	88
Figure 4.12 – Placing the nesting force (a) Initial set-up for the block example, Fig. 4.1 (b) Nesting force window, Fig. 4.11 .....	93
Figure 4.13 – Acceptable design for the block assembly .....	94
Figure 4.14 – Triangular assembly for the inspection method .....	102
Figure 4.15 – Nesting force window for the triangle assembly. The bolded lines are the points where the nesting force is not allowed. ....	106
Figure 4.16 – Acceptable EC design for the triangle assembly .....	107
Figure 4.17 – Triangular assembly for the design method .....	108
Figure 4.18 – Configuration that makes the assembly no longer exactly constrained .....	109
Figure 4.19 – Various configurations of the same block assembly .....	111
Figure 5.1 – Results of the Screw Theory Analysis. EC means that the design is exactly constrained.....	119
Figure 5.2 – All vectors associated with $\theta_1$ in a vector loop sample .....	125
Figure 5.3 – Components of each vector for the example in Fig. 5.2.....	126
Figure 5.4 – Resultant vectors representing the values of $\frac{\partial h_x}{\partial \theta_1}$ and $\frac{\partial h_y}{\partial \theta_1}$ .....	127
Figure 5.5 – Example showing when the B matrix would have linear dependence in the columns .....	133
Figure 5.6 – Example showing when the B matrix would have linear dependence in the rows (a) EC design (b) Over-constrained in the y-direction.....	134

Figure 5.7 – Example assembly to show how to use the B matrix to predict why a design did not assemble .....	136
Figure 5.8 – DLM for the slotted block example.....	144
Figure 5.9 – Slotted block with the slot at 45° .....	146
Figure 5.10 – Slotted block with the slot at 89° .....	147
Figure 5.11 – Slotted block with slot at 90° .....	148
Figure 5.12 – Exactly constrained block with three constraints .....	150
Figure 6.1 – Example calculation for the error of an assembly .....	157
Figure 6.2 – Block assembly with three constraints .....	158
Figure 6.3 – Block assembly with constraints at varying positions from the nominal .....	158
Figure 6.4 – Over/Under-constrained block assembly with three constraints .....	161
Figure 6.5 – Block assembly with four constraints.....	162
Figure 6.6 – Three possible assembly cases for the over-constrained block .....	163
Figure 6.7 – Alternate configuration for the over-constrained block .....	164
Figure 6.8 – Slotted block example .....	167
Figure 6.9 – Slotted block assembly with the slot at 75° .....	168
Figure 6.10 – Assembly with the slot angle at 90° .....	169
Figure 6.11 – Chart of results for slotted block assembly showing % assembled and error .....	171
Figure F.1 – Six lower pair joints for 3D assemblies (after Waldron and Kinzel [1999]). The arrows represent the allowable degrees of freedom. ....	255

## **CHAPTER 1 MOTIVATION FOR ESTABLISHING A QUANTITATIVE FOUNDATION FOR EXACTLY CONSTRAINED DESIGN**

### ***1.1 INTRODUCTION***

An increasingly competitive marketplace has sparked the demand to find more effective design methods that produce higher quality, cost-competitive mechanical assemblies. Yet, quality and cost often become competing objectives in many manufacturing processes because of unanticipated variability.

Unanticipated variability can adversely affect mechanical assemblies. Examples of unanticipated variability may include worn tools, varying dimensions among similar parts, varying job skills among technicians, or changing environments (such as temperature or load changes). Ultimately, variability leads to designs that do not always properly assemble as desired.

Avoiding the effects of variability can lead to higher quality, cost-competitive assemblies. Therefore, a prominent need has surfaced to find design methods that allow parts to correctly assemble, even when subjected to variation.

## ***1.2 MOTIVATION FOR THE RESEARCH***

Recent design methods have focused on eliminating the effects of variability. Collectively called “robust design”, they are intended to reduce the effects of variability without necessarily eliminating the causes.

For example, smart assemblies have recently been recommended as a robust design method. Smart assemblies include “features, not otherwise required by the function of the design, which allow the design to absorb or cancel out the effects of variation” [Downey et. al. 2002]. Some examples of smart assemblies include adjustable screws, springs that absorb variation, and slotted holes [Downey 2001]. The smart features can adjust as needed to allow the assembly to be used under a wide range of conditions.

Traditionally, however, robust designs get overlooked in favor of the more familiar methods of problem solving in manufacturing. These traditional methods include tightening tolerances on parts, re-design, and brute-force.

Designers tighten tolerances to try to control variability in an assembly. It seems logical that if parts will not come together in an assembly, those parts must be re-manufactured with dimensions closer to the nominal position. To avoid any future re-work, the parts are assigned tighter tolerances. However, assemblies that require high accuracy can have tolerances so tight that certain parts are almost *without* tolerances, leading to very costly designs. Perfect parts are high expectations from imperfect manufacturing processes and environments.

Another popular method to deal with variability in assemblies is to redesign parts mid-process until everything comes together as desired. Shapes change; different materials are explored; designs are altered; and the new design is progressively implemented. Problematic variability is simply eliminated through design changes made over time that seem to work. Re-design is a very real, very popular, and often very costly solution in industry.

Often, the most popular or common method employed to fix the effects of variability does not involve much thought about tolerances or the various properties of the parts. Instead, sheer brute force, often leading to deformations or dysfunctional assemblies, becomes the solution of choice.

However, the recent work of engineers, especially at the Eastman Kodak Company, has suggested that the solution may be more basic or fundamental than currently practiced or understood. Faulty assemblies may not be the effect of dimensions, tolerances, shapes, or material. The problem may well be with the total number of *constraints* found in an assembly. If a design does not behave as intended, it could be due to not enough or too many constraints in the assembly. This thesis will explore another robust design method called *exactly constrained design* that absorbs variability through minimizing the total number of constraints in an assembly, while still eliminating all necessary motion.



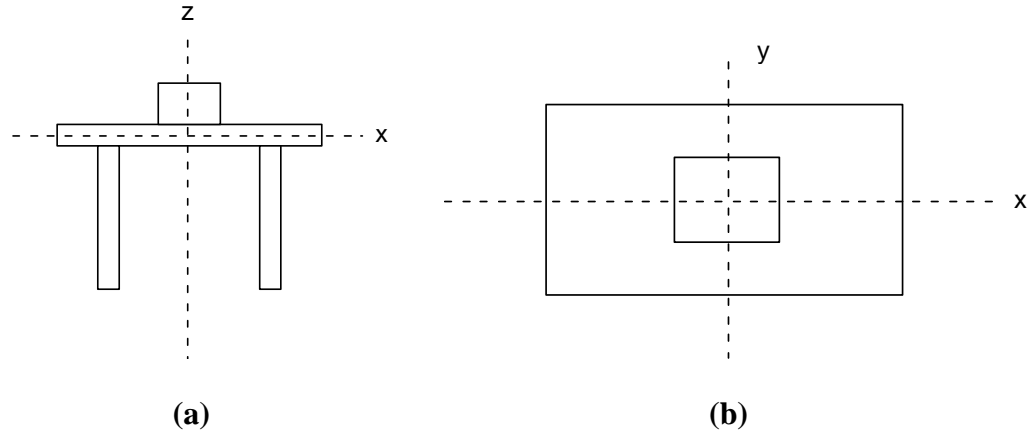
### ***1.3 EXACTLY CONSTRAINED DESIGN***

For any unconstrained part in 3D, six directions of motion are allowed: translation in the x, y, and z directions, and rotation about the x, y, and z axes. Likewise, for 2D parts, translation in the x and y-directions and rotation about the z-axis is possible. There are no limitations on motion if there are no constraints.

As parts come together to form an assembly, the joints formed by mating parts introduce constraints into the system. The constraints limit the allowable motion of the assembly.

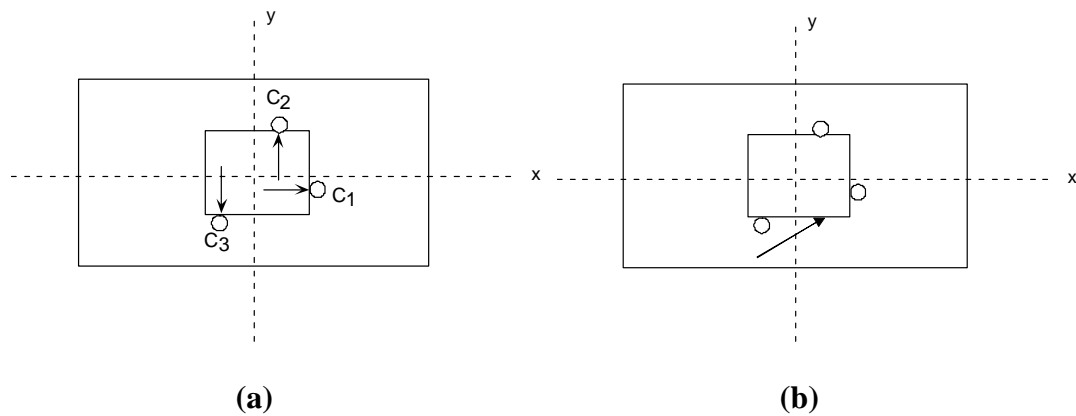
Exactly constrained (EC) design uses a minimum number of constraints to eliminate undesired motion. In addition, each constraint must have a complementary force applied (called a *nesting* force) that keeps the part and the constraint in contact. Strategic placement of the minimum number of constraints coupled with the nesting force constitutes an EC design.

As a simple example, consider Fig. 1.1, which is a block resting on a table. In this case, the block must not be allowed to move. Currently, however, this block can slide in the x and y directions, and it can rotate about z.



**Figure 1.1 – Block resting on a table. (a) Front view (b) Top view**

To inhibit the motion currently allowed, constraints must be added (Fig. 1.2). To constrain translation in the x-direction, a peg ( $C_1$ ) is placed to the right of the block. To constrain y-translation, a peg (marked  $C_2$ ) is placed at the top of the block. In order to inhibit rotation, a final peg (marked  $C_3$ ) is added to the bottom left of the block. As mentioned earlier, it is necessary to have a nesting force for each constraint (Fig. 1.2a), but these forces may be combined to find one resultant nesting force for the design (Fig. 1.2b).



**Figure 1.2 – Constraints and nesting forces to make the box immobile (a) Individual nesting forces (b) Resultant nesting force**

A simple comparison of two common assemblies may further help to understand EC design. Imagine a three-legged stool versus a normal table chair with four legs. Both assemblies need to stay flat on the ground for optimal convenience. When just one leg is attached to the seat of the stool or chair, each assembly can still rotate in all three directions. When two legs are attached, each assembly can still rotate about the line formed by the two points where the legs rest. When a third leg is added, all three legs sit flat on the ground. There is no rotation or translation. All the legs will rest squarely on the ground regardless of incline or roughness. Adding a fourth leg now makes the chair over-constrained. In order for all four legs to sit squarely on the ground (and thus allow no “wobble”), the ground must be perfect and the legs must be the same height. Otherwise, there must either be deformation between the ground and the legs so the chair sits flat, or the user must have a tolerance for “wobble.” The fourth leg makes a difference. The exactly constrained stool has more flexibility for use than the four-legged chair, although the four-legged chair has greater stability (and hence the reason four-legged chairs are used more often).

#### ***1.4 ADVANTAGES OF EXACTLY CONSTRAINED DESIGN***

Exactly constrained design yields some highly desirable advantages. As will be discussed throughout this thesis, EC designs will assemble under a wide range of conditions with no looseness or binding. In fact, EC assemblies will not just *assemble* under a wide variety of conditions, but they will *operate* under a wide variety of conditions as well. For example, changes in material due to temperature often become a non-issue. This kind of robust design methodology means that assemblies can be produced with lower priced, less accurate parts.

Additional advantages include other benefits as well. Assemblies can be assembled and reassembled with very little overall change to the function of the system. Tolerances can be looser. Costs are often reduced due to less re-work. Less time and/or resources are spent in re-design or problem solving to fix an EC assembly.

### ***1.5 ADVOCATES FOR EXACTLY CONSTRAINED DESIGN***

Exactly constrained design is unusual because the greatest advocates have come not from academia, but from industry. Lawrence Kamm [1993], who refers to EC design as “MinCD”, states the benefits of such a design.

When you do minimum constraint design (MinCD), you support and guide each body only at points, and at as few points as possible to get the desired performance. If you do so, you will achieve zero looseness and zero binding of moving parts; you will achieve assembly of fixed parts without strains or rework; and you will do so despite loose manufacturing tolerances and semiskilled assembly labor. You will minimize the manufacturing cost of your mechanism, you will make it more reliable, you will make it easier to disassemble and reassemble, and you will make it easier to maintain.

Douglass L. Blanding [1999] of the Eastman Kodak Company, and one of the leaders in defining principles related to EC design, explains some of the advantages he has found in his experience with EC design.

The use of these [EC design] principles, collectively called Exact Constraint Design principles, provides the designer with a better understanding of a machine’s behavior. This understanding allows the

designer to easily create new designs which are both low in cost and high in performance.

... Among the benefits to be attained in following these principles are extreme precision, predictable performance, and infinitesimal distortion of the component parts.

Jon Kriegel [1994], also of the Eastman Kodak company, made a plea to include EC design as part of the engineering curriculum.

Based on...an unending list of...examples, it is suggested that these problems represent a major weakness in the undergraduate engineering educational system. The objective...is to solicit the support of academicians in including Exact Constraint Design as a basic topic in Machine Design classes and textbooks. This involves vocabulary, concepts, and examples or case-studies, (the author could personally supply 30), and deserves an independent chapter heading.

Michael French, who has included EC design in his textbooks *Form, Structure and Mechanism* [1992] and *Conceptual Design for Engineers* [1998], writes, “It must not be used blindly or invariably..., but it is perhaps the most useful principle in machine design.”

## **1.6 CURRENT PRACTICES**

Current techniques to analyze and use EC design have primarily centered on intuition and graphical methods. They often rely on the designers’ experience to make decisions on where to place constraints.

For example, Blanding [1999] has developed a system where all constraints are schematically the same (essentially pin joints) thus leading to a 6 **R**s interpretation—where any translation is really just a rotation at infinity. He observes that the type of constraint is not as important as where it is placed. As will be shown in Chapter 2, rules primarily based on experience have been developed by him to find a location for the constraints in an assembly. Included in his method is a graphical approach, which shows “windows” where nesting forces may sit to keep the assembly properly seated.

### ***1.7 OBJECTIVES OF THE THESIS***

The purpose of this thesis is to build a solid quantitative foundation for exactly constrained design. This foundation will be built upon two primary concepts. First, it has been observed that EC designs are statically determinate. This observation allows equilibrium equations to be used to compute forces. Second, EC designs can be analyzed using the direct linearization method (DLM). The vector loops in the DLM are utilized to show the effects of variation on EC designs.

The concepts stated above will be used to build a quantitative foundation for exactly constrained design by:

- Developing a solid definition of exactly constrained design based on quantitative principles
- Developing quantitative methods for analyzing locations of constraints
- Developing quantitative methods for analyzing the nesting force window

- Developing a quantitative method to determine the “goodness” of an EC design
- Examining the effects of variation in exactly constrained designs vs. over-constrained designs

## ***1.8 DELIMITATIONS***

The thesis presented will primarily treat 2D assemblies. The results found for 2D assemblies can be generalized for 3D assemblies.

Also, mechanisms are not included in this thesis. Only immobile assemblies are presented as examples.

## ***1.9 THESIS OVERVIEW***

The remainder of the thesis will proceed in logical fashion. Chapter 2 will give much more detail and further background for exactly constrained design. Chapter 3 will define and use mathematical or quantitative principles to validate many of the heuristic rules developed over time for EC design. Chapter 4 will expand the work of Chapter 3 by presenting a general method to analyze the placement of constraints in an EC design. Chapter 5 builds upon the work in Chapter 4 by showing a procedure to find a measure of “goodness” between several configurations of an EC design. Chapter 6 will then show how EC designs are more robust than similarly over-constrained designs. Finally, Chapter 7 will state all contributions, conclusions, and recommendations related to this thesis.

## **CHAPTER 2 BACKGROUND AND LITERATURE REVIEW FOR EXACTLY CONSTRAINED DESIGN**

### ***2.1 INTRODUCTION***

As mentioned in Chapter 1, exactly constrained (EC) design can be a useful tool in the engineering world. Chapter 2 will discuss the background, history, and current analysis methods for EC design.

EC design is a robust design method for mechanical assemblies with a history dating back to at least the early 1840's. The history, background, and current methods of analysis will be followed by an explanation of the antitheses to EC design: over-constrained and under-constrained design. Examples of EC design will be followed by the explanation of several tools that will help to ultimately develop the quantitative foundation presented in this thesis.

### ***2.2 EXACTLY CONSTRAINED DESIGN AS A ROBUST DESIGN METHOD***

Parkinson [1995] has defined robust design as “a design that works properly even when subjected to variation, which may be introduced by manufacturing processes, by the environment, by the end user, or by parts provided by outside suppliers.” Although applicable to assemblies, robust design also pertains to all types of design models.



As will be illustrated in Chapter 6, EC designs fall into the category of robust design because they can appropriately assemble and reassemble even when subjected to variation. As will be explained in Section 2.4, EC designs do so by using the minimum number of constraints to hold an assembly in place.

Before the technical details are presented for such a robust design, however, the history of the method will be reviewed. The purpose in reviewing the history is best explained by Chris Evans [1989] in *Precision Engineering* as a way to “show the present to itself by revealing its origin.” The history shows that the basic principles were understood and preserved by a handful of followers in the generations that followed the original pioneers.

### **2.3 HISTORY OF EXACTLY CONSTRAINED DESIGN**

One of the earliest leaders in EC design was Robert Willis, who extensively lectured and published his *Principles of Mechanism* beginning around 1841. Presumably as a result of Willis’ work, William Thomson (Lord Kelvin) and James Clerk Maxwell carried on the principles known in that day as “geometric” or “kinematic” design.

In 1876, James Clerk Maxwell [Niven 1890] clearly described the basic ideas relating to what was often referred to as “kinematic design”.

Each solid piece of the instrument is intended to be either fixed or movable, and to have a certain definite shape. It is acted on by its own weight, and other forces, but it ought not to be subjected to unnecessary stresses, for these not only diminish its strength, but (what for scientific

purposes may be more injurious), they alter its figure, and may, by their unexpected changes during the course of an experiment, produce disturbance or confusion in the observations we have to make.

We have, therefore, to consider the methods of relieving the pieces of an instrument from unnecessary strain, of securing for the fixed parts a determinate position, and of ensuring that the movable parts shall move freely, yet without shake.

This we may do by attending to the well-known fact in kinematics—‘A RIGID BODY HAS SIX DEGREES OF FREEDOM’.

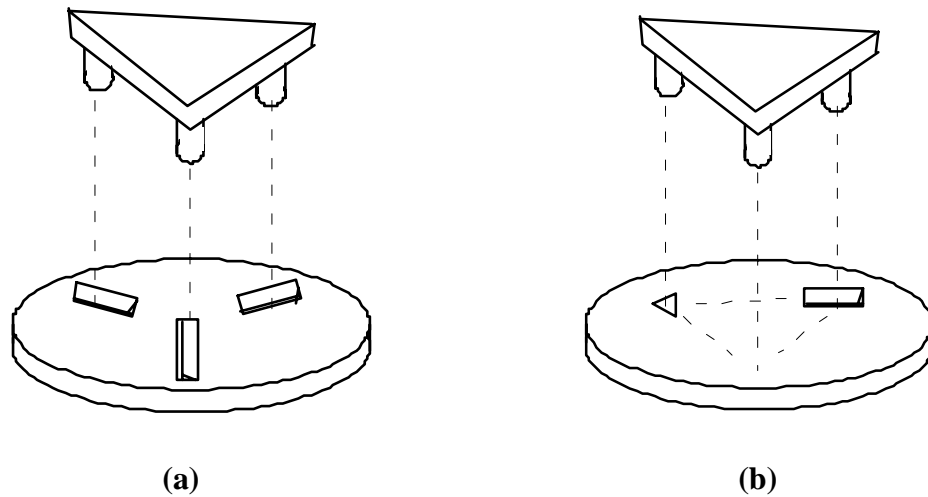
A rigid body is one whose form does not vary. The pieces of our instrument are solid, but not rigid. They are liable to change of form under stress, but such change of form is not desirable, except in certain special parts, such as springs.

*Hence if a solid piece is constrained in more than six ways it will be subject to internal stress, and will become strained and distorted, and this in a manner which, without the most exact micrometrical measurements, it would be impossible to specify.*

In instruments which are exposed to rough usage it may sometimes be advisable to secure a piece from becoming loose, even at the risk of jamming it; *but in apparatus for accurate work it is essential that the bearings for every piece should be properly defined, both in number and in position* (emphasis added).

Thus, Maxwell described that any solid piece must not be constrained in more than six ways; otherwise, the part will become strained and disfigured. He continued by

illustrating “[m]ethods of placing an instrument in a definite position”. His example explains the three V’s method for a geometrical or kinematic clamp (Fig. 2.1a), which he compares to Kelvin’s kinematic clamp (which uses a trihedral hole, a V-groove, and the horizontal plane of the base to constrain motion). While more specific details will be discussed in Section 2.4, the six degrees of freedom are constrained by three joints to allow no motion between parts in both clamps.



**Figure 2.1 – Examples of kinematic clamps (a) Maxwell’s 3 V’s example (b) Lord Kelvin’s clamp (after Evans [1989])**

In 1937 and again in 1954, T. N. Whitehead published his *Design and Use of Instruments and Accurate Mechanism* which is based on the principles described by Maxwell. It resurrected and developed the ideas promoted by the previous experts, and it formed the basis for what is today called *exactly constrained design*.

Kinematic design, as that term is frequently used, implies a design whereby the various links of each element, forming an instrument, are

constrained by the theoretically minimum number of point contacts, given the degrees of constraint required.

Lawrence Kamm [1993] developed an entire career based on the principles as outlined in Whitehead's book. In his book, *Designing Cost-Efficient Mechanisms*, Kamm follows Whitehead's approach in presenting "a book of ideas and not of calculations." He offers a basic description and many ideas on designing mechanisms based on the number and placement of constraints.

Several other industrial leaders also built their careers on principles as defined in the work of Whitehead. Most notably, engineers from the Eastman Kodak Company used EC design for over 20 years to solve many problems. John McLeod (who coined the term "exact constraint design") and John E. Morse ("Exact Jack" Morse) used the EC principles for designs ranging from structures to conveyor belts to flexure mechanisms.

The design principles of EC design were more fully described by another member of the Eastman Kodak Company, Douglass L. Blanding, author of *Exact Constraint: Machine Design Using Kinematic Principles*. He had the benefit of working with and learning from Jack Morse for about two years. This mentoring helped him establish and further define many basic principles in EC design. Blanding's work is the basis for many of the EC designs in use today.

Before a detailed overview can be presented for the current methods developed by Blanding and others on how to use the principles of EC design, the underlying concepts

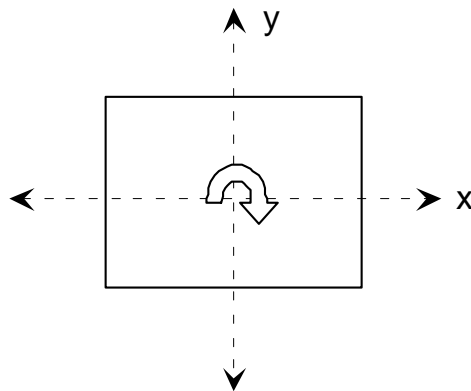
must be discussed. A brief background for EC design will be followed by the methods currently in use.

#### **2.4 BACKGROUND FOR EXACTLY CONSTRAINED DESIGN**

An assembly is comprised of smaller, interconnecting parts. It can be immobile, such as a stool is an assembly made of a seat and legs, or it can have motion, such as a robot.

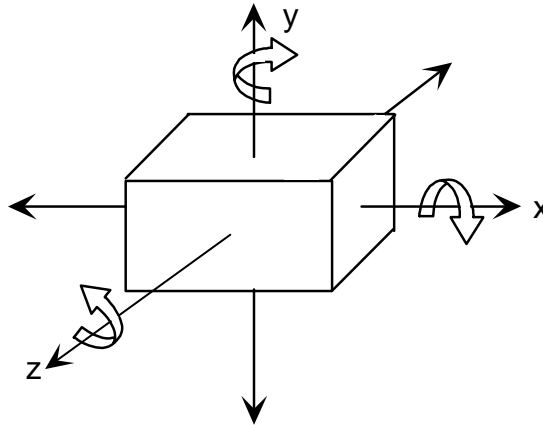
Each part in an assembly connects to another part by way of a joint. Each time parts are joined together, the degrees of freedom are affected.

The degrees of freedom define the motion that is allowed. It is well known that for 2D solids with no constraints (Fig. 2.2), there are three degrees of freedom: x-translation, y-translation, and rotation about the z-axis.



**Figure 2.2 – 2D object with three degrees of freedom. The object can translate along the x and y-axes, and it can rotate about the z-axis.**

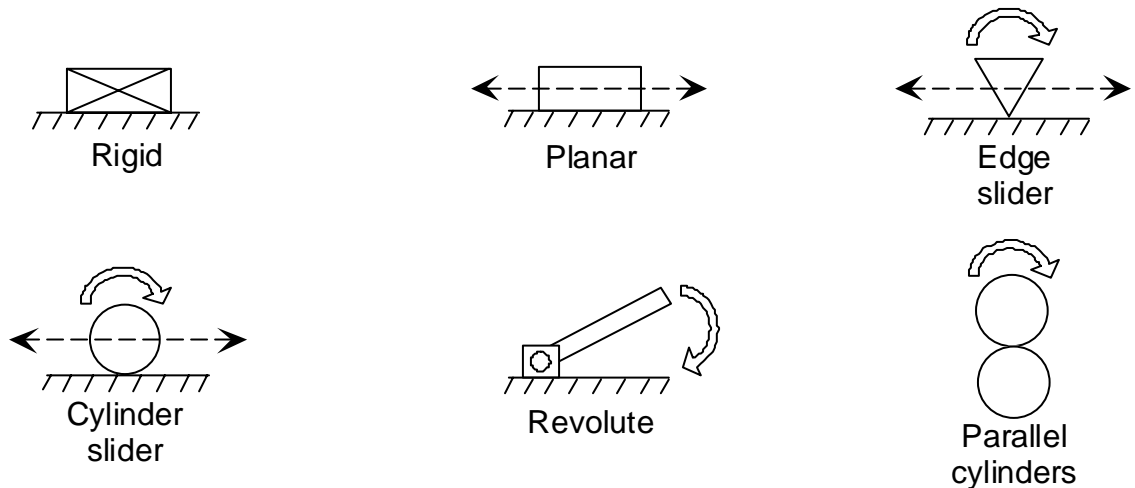
For 3D solids with no constraints (Fig. 2.3), there are six degrees of freedom. There will be x, y, and z translation, as well as rotation about the x, y, and z-axes.



**Figure 2.3 – 3D object with six degrees of freedom. The object can translate along and rotate about the x, y, and z-axes.**

When parts assemble together and form joints, the degrees of freedom change. Each type of joint in the assembly constrains motion in one or more directions, but the joint may also continue to allow motion in other directions. For each direction inhibited by a joint, one degree of freedom is lost, thus adding one degree of constraint to the assembly. The types and number of joints used ultimately determine the degrees of freedom in an assembly.

Chase [1999] has suggested that all mating parts in 2D assemblies can be described by the six kinematic joint types in Fig. 2.4 (a similar graphic for 3D assemblies is found in Appendix F). The rigid joint provides three constraints. A planar joint provides two constraints. An edge slider joint provides one constraint, and so on.



**Figure 2.4 – Six kinematic joints for 2D assemblies. The arrows represent the allowable degrees of freedom (after Chase [1999]).**

It should be noted that while Blanding [1999] has chosen to limit his designs to the cylinder slider type joint, any of the joints presented in Fig. 2.4 can be used in an EC design. Figure 2.24, for example, shows a block assembly which uses a revolute joint to constrain two degrees of freedom.

With a basic understanding of assemblies and their constraints, EC design can be easily described. An EC design entails assigning a minimal number of constraints to eliminate each necessary degree of freedom in an assembly. Constraints are achieved through joints, and the appropriate type and number of joints to use depends on which degree(s) of freedom should be constrained. The 2D joints shown earlier (Fig. 2.4) can provide one, two, or three constraints per joint, depending on the type of joint chosen.

Placing a constraint includes more than just finding a location for a joint. If the constraint does not stay in constant contact with its mating part in an assembly, it is as if

no constraint exists and unexpected motion results. Therefore, every constraint must also have a nesting force to guarantee constant contact with the part.

If there are three constraints, the assembly must then also have three nesting forces. However, these three nesting forces can be combined into one resultant nesting force that would properly seat the assembly. It is important to emphasize that to properly seat the assembly, the nesting force must be applied so as to provide force in all necessary directions; thus, there must be a nesting force that provides seating in both the x and y-directions.

Examples of nesting forces may be the weight of an assembly, an applied force, friction, and smart features that absorb variability. Pearce [2003] gives a method to design nesting forces using smart features.

## ***2.5 CURRENT METHODS TO ANALYZE EXACTLY CONSTRAINED DESIGN***

The basic background to EC design naturally leads to a discussion on the current methods used by the experts. EC design requires several major considerations. First, how many constraints will be required to exactly constrain the assembly as necessary, and where should those constraints be placed in an assembly? Then, where should the nesting force(s) be placed?

### **2.5.1 DEFINING THE CONSTRAINTS**

As mentioned earlier, the type and number of constraints depend on the required motion of the assembly. However, certain rules have been described by Blanding to help



correctly place constraints for EC design. Those rules are presented as summarized by Skakoon [2000].

### **2D rules for exactly constrained design**

1. No two constraints are co-linear.
2. No four constraints are in a single plane.
3. No three constraints are parallel.
4. No three constraints intersect at a point.

### **3D rules for exactly constrained design**

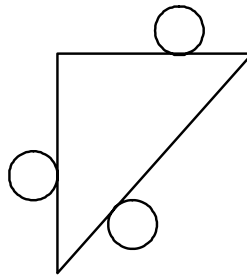
1. No four constraints are parallel.
2. No four constraints intersect at a point.
3. No four constraints are in the same plane.

#### **2.5.2 DEFINING THE NESTING FORCE**

Blanding describes a graphical method to find a proper position for the nesting force. His method finds a “window”, and the nesting force can be placed in this region without causing motion or instability in the assembly. His method can be summarized, as follows.

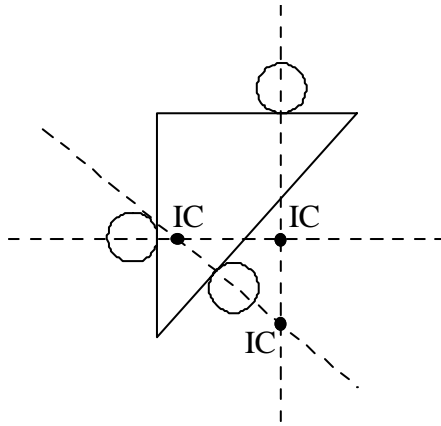
1. Draw constraint lines to find the instant centers between constraints.
2. Remove the constraints one at a time to find the effects on the assembly.
3. Determine the proper rotation of the block with respect to the pertinent instant center to enable the constraint to restore contact with the assembly.
4. Determine which line segments of the constraint lines drawn allow a force to cross it and still maintain the proper rotation for each instant center.
5. If the conditions in step 4 are met, the segment is allowed into the nesting force window.

This method can be better understood with an example. Therefore, consider the constrained triangle given in Fig. 2.5.



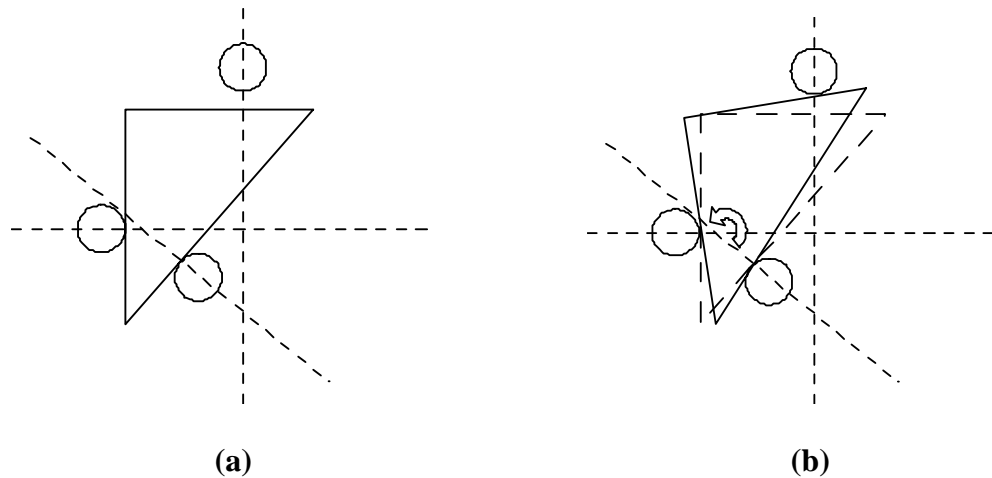
**Figure 2.5 – Triangle with three constraints**

The resultant nesting force must sit within a certain window in order to be effective. To find that window, “constraint lines” must first be drawn. They are infinitely long lines normal to the object’s surface at the point of contact. The intersection of each of these lines is called an instant center (Fig. 2.6).



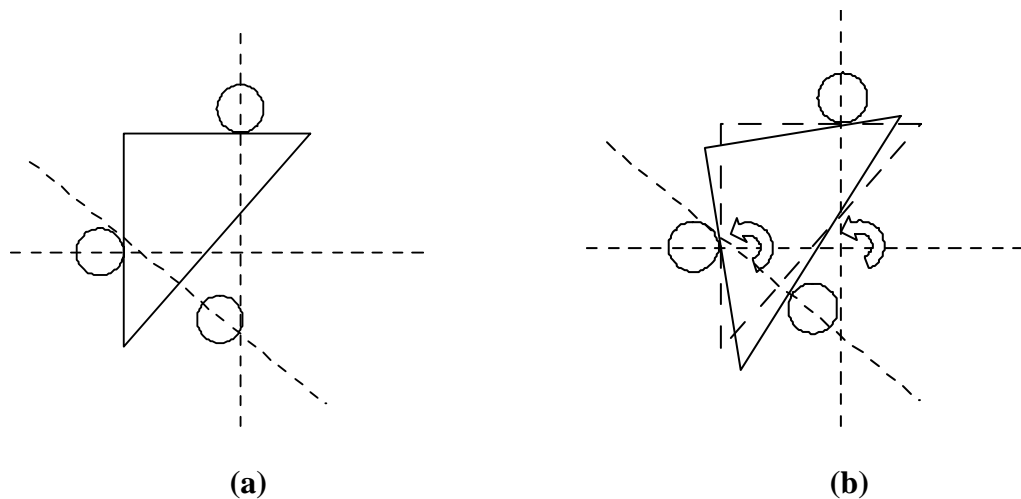
**Figure 2.6 – Triangle with constraint lines drawn and instant centers defined**

Next, each constraint must be moved away from the triangle, one at a time, to determine the effects on the assembly. What would need to happen for the object to restore contact with the constraints? As an example, if the top constraint were moved as shown in Fig. 2.7a, the triangle would need to pivot counterclockwise about the instant center of the two remaining constraints in order to restore contact with the top constraint. Therefore, a counterclockwise arrow is drawn around that instant center to show what would need to happen to restore contact between the part and the constraint (Fig. 2.7b).



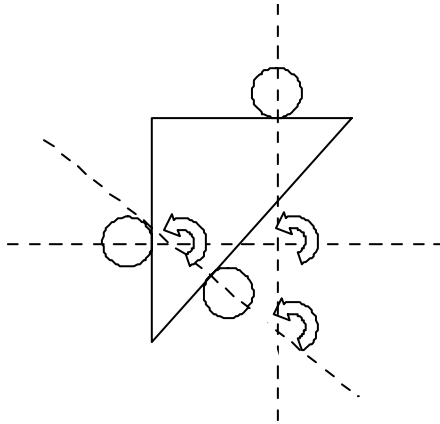
**Figure 2.7 – Finding rotation on the left instant center (a) Moving the top constraint (b) Direction of necessary rotation at the instant center to restore contact between the constraint and the triangle**

Next, the constraint on the hypotenuse is moved away from the triangle (Fig. 2.8a). In order to re-establish the contact between the constraint and the object, it will again be necessary to rotate the triangle counterclockwise about the instant center for the two remaining constraints (Fig. 2.8b).



**Figure 2.8 – Finding rotation on the top right instant center (a) Moving the constraint on the hypotenuse (b) Direction of necessary rotation to restore contact between the constraint and the triangle**

After moving the left side constraint away from the triangle, the final rotation can be found. The object would need counterclockwise rotation about the remaining instant center to maintain its current position (Fig. 2.9).



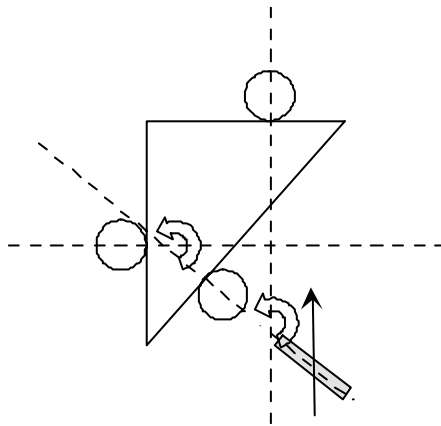
**Figure 2.9 – Direction of all necessary rotations on each instant center**

After finding the proper rotations on each instant center, each segment of each constraint line is individually evaluated to determine if it is allowed in the nesting force window. Certain conditions must be met for a line segment to qualify to be in the window.

1. The line of action for the nesting force must intersect the segment, AND
2. The direction of the force must cause correct rotation about each of the instant centers related to the segment of the constraint line that is being evaluated.

If the conditions are met, the segment is part of the allowable region of the nesting force window. If the conditions are not met, a thick, bolded line is imposed on the drawing to show which segments the nesting force line of action cannot cross.

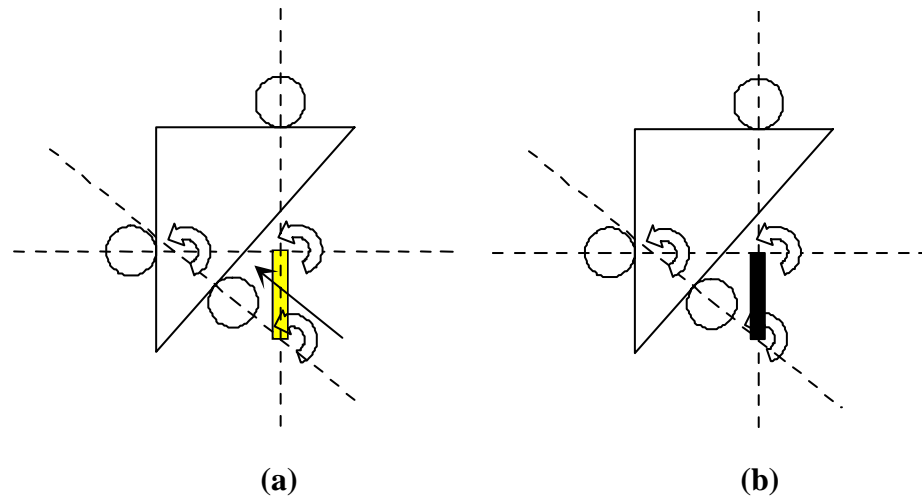
To continue the example presented earlier, consider the line highlighted in Fig. 2.10 below, with a force crossing the segment (only the rotations for the relevant instant centers are shown). The force causes counterclockwise rotation about the instant center for the top left instant center, as well as the bottom instant center. Thus, the line of action for a nesting force would be allowed to cross this segment, and it is an acceptable region of the nesting force window.



**Figure 2.10 – Test to see if the nesting force is permitted to pass through the highlighted line.**

Now, consider the highlighted line segment in Fig. 2.11a, which has a force passing through it. The force allows correct rotation for the lower instant center, but the rotation is incorrect for the top instant center. Therefore, this line segment is not allowed,

and the highlighted line is blackened to show that the line of action for the nesting force cannot pass through this line segment (Fig. 2.11b).

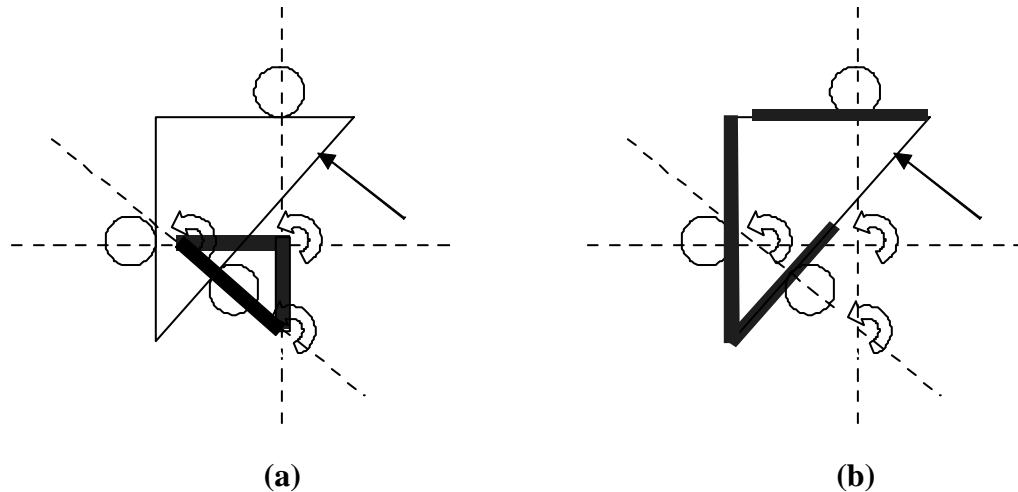


**Figure 2.11 – Testing a segment in the nesting force window (a) Testing a line segment to determine if it is in the nesting force window (b) Bolded line to show the line segment is not allowed in the nesting force window**

Each segment of each constraint line in turn can be tested. Fig. 2.12a shows the nesting force window after all line segments have been checked. Again, the bolded lines show where the line of action for the nesting force cannot cross. Also, a resultant nesting force is shown on the assembly.

The nesting force window can be more easily understood if it is projected onto the surface of the assembly. While specific details showing how to project the graphical nesting force window onto the boundary of the assembly have never been seen in publication, this projection is easily accomplished by placing a nesting force along each segment of the boundary. If the nesting force passes through the unacceptable region of

Fig. 2.12a, that portion of the boundary is bolded. If it does not pass through the unacceptable region, all constraint line segments through which the line of action for the nesting force passes must be re-tested to ensure proper rotation at the instant centers. Figure 2.12b shows the nesting force window projected onto each surface.



**Figure 2.12 – The allowable nesting force window. The bolded lines show the segments through which the nesting force line of action cannot go (a) The line of action of the nesting force cannot pass through the bolded region (b) A perpendicular nesting force cannot be placed in any bolded portion of the assembly.**

Thus, the graphical method used by Blanding finds the nesting force window based on the position of the constraints and the intersection of the constraint lines. The nesting force itself can be placed anywhere in the acceptable region of the window.

## 2.6 ALTERNATE DESIGNS

With such a powerful tool in engineering, one may wonder why EC design is not used more frequently. As the principles of EC design are not generally taught in school,



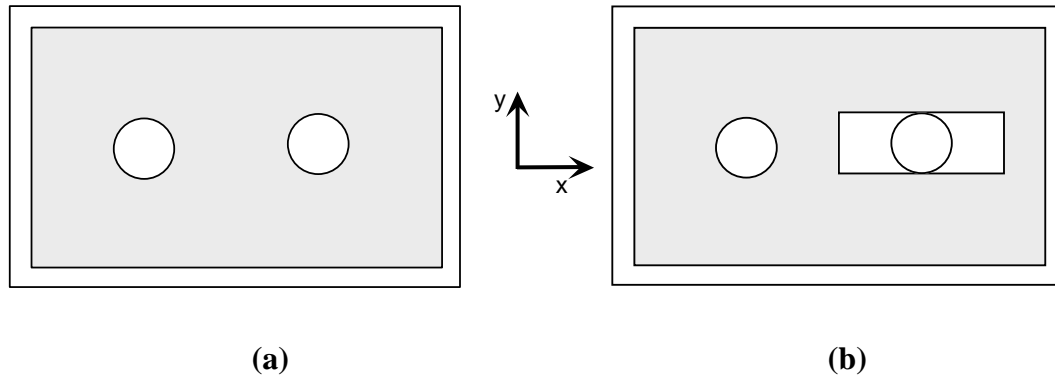
engineers are not usually familiar with, or they simply do not have an appreciation for the method. They tend to have more experience with two other modes of design: over-constrained design and under-constrained design.

### **2.6.1 OVER-CONSTRAINED DESIGN**

It is not fair to say that all designs must be exactly constrained, although there are indeed many advantages to making a design exactly constrained. For example, lower cost, inaccurate parts can be assembled and reassembled with a very high level of precision (meaning that the results can be consistently reproduced in the same assembly). However, sometimes higher stability, or greater load sharing is desired. One alternative is to over-constrain the design, at a higher cost (and often lower quality) to the manufacturer.

An over-constrained (OC) design simply uses more constraints than the minimum number necessary, and it is very common in design today. Two or more of the constraints compete to hinder the same degree of freedom.

As a very simple example, consider the block assembly in Fig. 2.13a below. The base plate of the assembly has two rigidly-connected pegs. They fit through the two drilled holes of the top plate.



**Figure 2.13 – Slotted block OC and EC design (a) Over-constrained assembly  
(b) Exactly constrained assembly**

The left peg constrains the assembly in the x and y-directions. Only one more constraint is needed—something to eliminate rotation; however, the right peg not only eliminates rotation, but it also competes to constrain the block in the translational x-direction. If the location of either peg is manufactured with any variability in the x-direction, this block will not assemble. A similar design that is exactly constrained is shown in Fig. 2.13b. This new assembly could absorb variation in the x-direction.

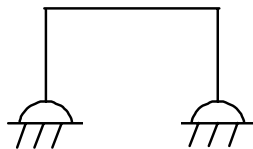
Over-constrained assemblies often happen without the designer realizing what is causing the assembly problems in a design. Designers have learned to tighten tolerances, develop manufacturing methods with higher precision, expect higher accuracy from intermediate parts, and just force something to work when they do not understand that the design is over-constrained. Valuable resources are used in redesign, and time is lost when things do not work correctly. In all of the redesign, very rarely does an engineer turn to the root of the problem—the design is not properly constrained to allow for flexibility in an assembly.

In the case of over-constrained designs, the designer must determine if the advantages of over-constraint outweigh the advantages of an EC design. Symptoms of over-constraint include binding or loose fitting parts, built-up stresses in assemblies, and the need for tighter tolerances.

### 2.6.2 UNDER-CONSTRAINED DESIGN

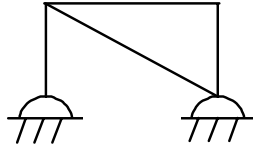
An under-constrained design is a design that should not have motion in a particular direction, but motion still occurs. There is insufficient constraint or insufficient nesting force to inhibit the motion.

An example of an under-constrained design can be found in a simple four bar mechanism.



**Figure 2.14 – Four bar mechanism.**

Under most circumstances, a designer would want one degree of freedom in the four bar mechanism. However, sometimes designers use the four bar for immobile designs, such as fences or platforms. If not properly constrained, the design will fail. An exactly constrained immobile design is shown in Fig. 2.15.



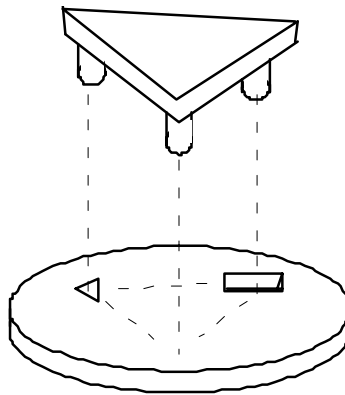
**Figure 2.15 – Exactly constrained design.**

## ***2.7 EXAMPLES OF EXACTLY CONSTRAINED DESIGN***

Now that the background has been presented, several examples will be offered to show just how effective EC design can be. Kelvin’s kinematic clamp will first be revisited, followed by an example from Blanding. Finally, an industrial example will be given to show how EC design improved a copy machine component.

### **2.7.1 KELVIN’S KINEMATIC CLAMP**

Lord Kelvin’s [Evans 1989] kinematic clamp example from Fig. 2.1 (shown below as Fig. 2.16) provides a simple case in which to see the strength of EC design. Three simple joints will provide six constraints which eliminate all six degrees of freedom to provide a repeatable assembly.



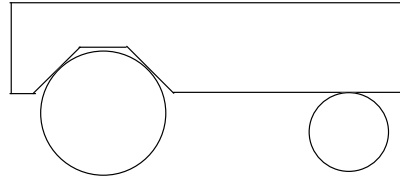
**Figure 2.16 – Kelvin’s kinematic clamp (after Evans [1989])**

To begin, imagine the triangular top piece with three pegs, and the circular fixed base without any notches. If the two pieces are assembled, the design would obviously not be immobile. Rather, three points from the triangular part would slide along the face of the base. The assembly would be partially constrained, allowing two directions of translation and one direction of rotation.

After adding the trihedral notch (on the left), three degrees of freedom are eliminated ( $x$ ,  $y$ , and  $z$  positions have been set) because the trihedral has three points of contact with its mating peg. The trough shape eliminates two more degrees of freedom (two rotations) after two points of the right-back peg make contact with it. The top piece can now only rotate about the line formed between the trihedral notch and the trough. That leaves one degree of freedom, and thus the third peg contacts the face of the base, which eliminates the remaining rotation. In this example, the weight of the clamp is the nesting force. Notice that this clamp shows high precision because it can be assembled and reassembled with no change to the overall function of the clamp. Also, the tolerances on any of the parts would not have much effect on the function of the assembly. Inaccurate parts can be used with similar results.

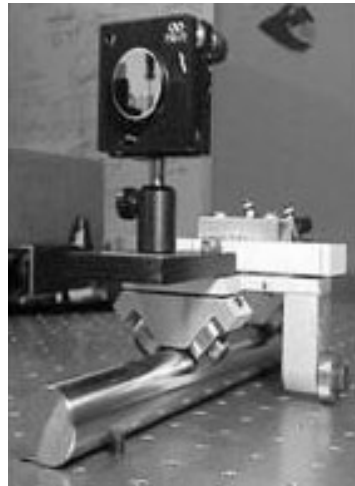
### **2.7.2 BLANDING'S 2D BODY ON TWO PINS EXAMPLE**

Douglass Blanding [1999] shows the example depicted in Fig. 2.17. This 2D assembly has the body situated on two pins. The left pin provides two constraints for the assembly, which leaves the body to rotate. The right pin provides the constraint against rotation.



**Figure 2.17 – 2D body located by two pins (after Blanding [1999])**

Figure 2.18 shows an industrial example in 3D based on the same principles. It is called the MicronWorm, and it is used in optical research. Notice at the base of the machine that two rollers move along a shaft, while to the bottom right of the design a roller moves along the base.

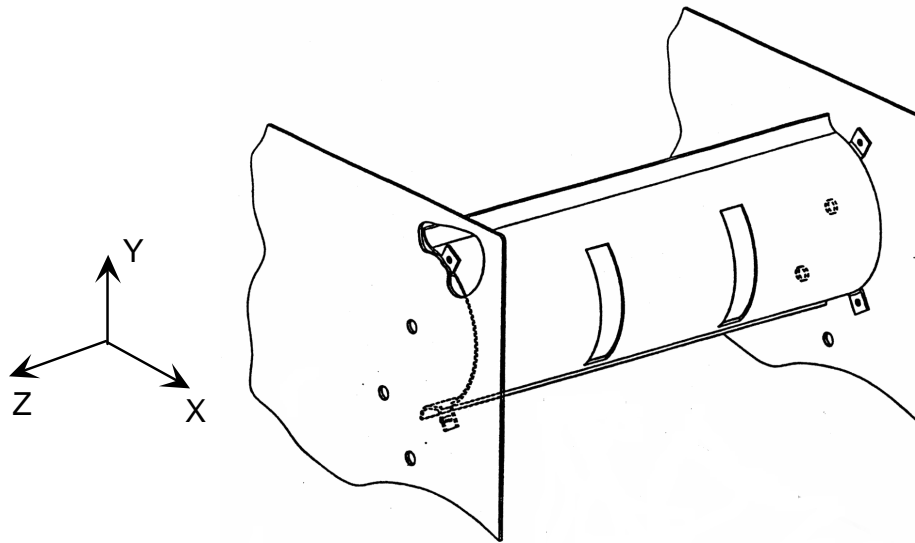


**Figure 2.18 – Industrial example using the assembly presented in Fig. 2.17 (after Savoie, MIT)**

### **2.7.3 KRIEGEL'S COPY MACHINE EXAMPLE**

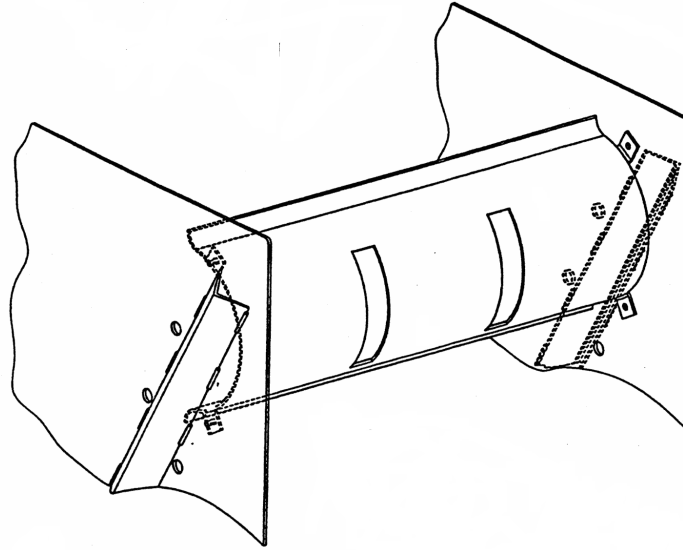
Kriegel [1994] used the theory of EC design to resolve over-constraint in an office copier machine. The copier machine had two large foam rolls mounted inside the part in question, called a baffle. The rollers inside the baffle feed paper around a turn,

and they send it on to the staple hopper. The initial baffle design (Fig. 2.19) had four mounting screws holding the part in place between the side plates of a major frame. Unless the baffle was nearly perfect when manufactured, there was some clearance between the baffle and the side plates. As the screws were tightened, the side plates deflected inward, and that in turn inhibited other parts of the machine from performing properly.



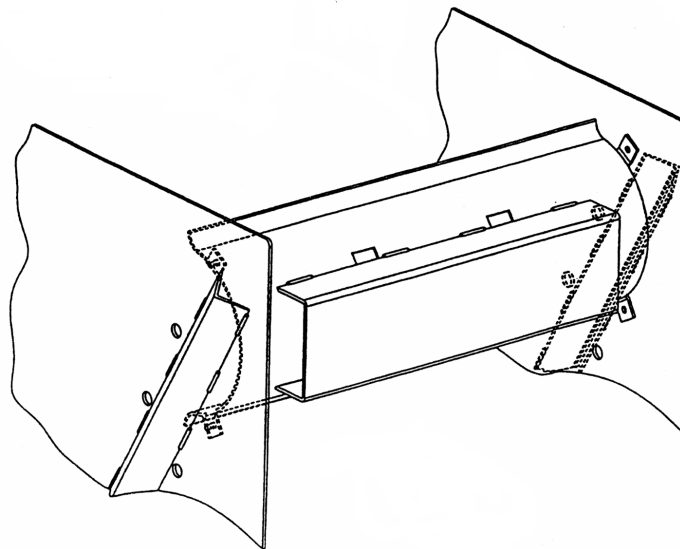
**Figure 2.19 – Initial baffle design (after Kriegel [1994])**

In an attempt to stop the side plates from deflecting, an engineer added angle-iron channels to be welded onto the side frames (Fig. 2.20). This step indeed stopped the deflection of the side plates; but when the screws were tightened, the baffle arched in such a manner that there was a gap in the foam rolls, and the paper-drive was lost.



**Figure 2.20 – Baffle with angle-iron channels (after Kriegel [1994])**

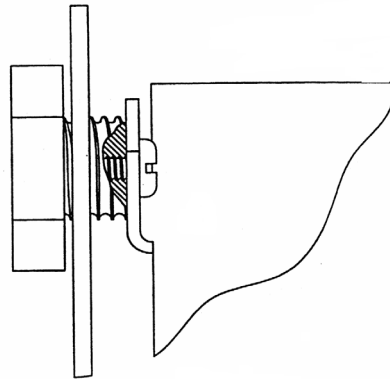
In response, the engineer added an additional stiffening brace to the baffle (Fig. 2.21) to eliminate the possibility of the rolls losing their function. Hence, when the screws were tightened, the ears fractured, and they tore off the baffle.



**Figure 2.21 – Baffle with an additional stiffening brace (after Kriegel [1994])**



A designer began to focus on the source of the problem instead of just the symptoms of the problem. To counter the latest issue, two relatively expensive fasteners were designed into the system to be used at whichever end had the screws tightened last (Fig. 2.22). The fastener could now be adjusted at the time of assembly to set a specific gap for each baffle.

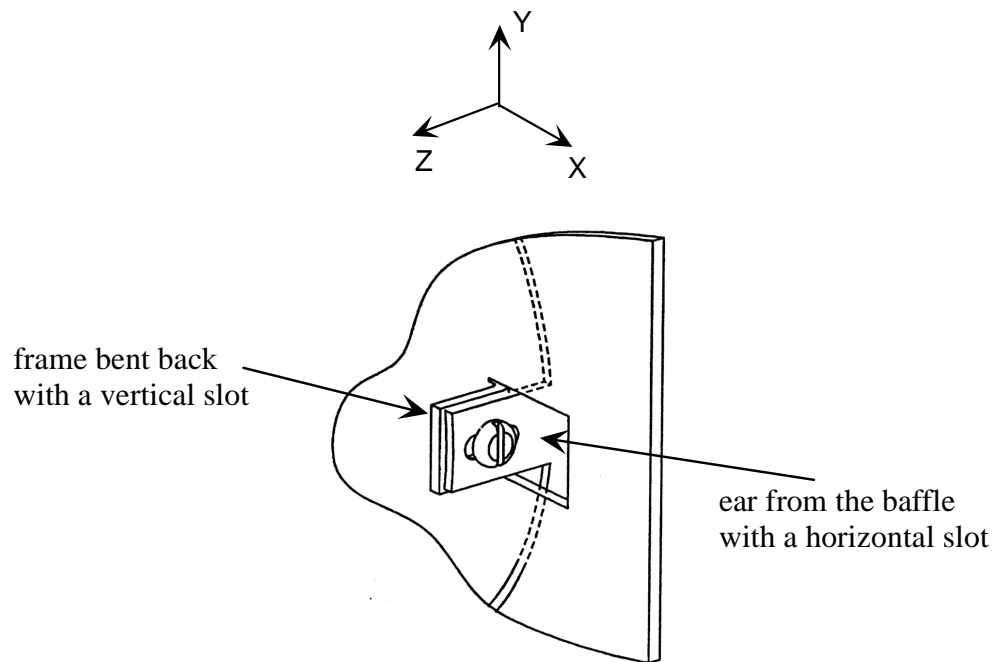


**Figure 2.22 – Fastener for the baffle (after Kriegel [1994])**

Kriegel explained that the solution to this problem was to be found in the number of constraints controlling the baffle. Once the first two screws were tightened on one end of the baffle, the final position of the baffle along the z-axis (see Fig. 2.19) was already decided. Trying to tighten the screws on the other side of the baffle competed with the already established position of the screws on the first side. In fact, the first two screws fixed all degrees of freedom for the assembly except rotation about the line connecting the two screws.

Thus, on the free end, a pin may be inserted through one, but not both of the remaining ears. To ensure that the baffle would not rattle (which may result with just one

pin inserted), a single straight tab through the frame allowed for a natural clamp without applying a load along the z-axis (Fig. 2.23). As Kriegel states, “Slots in both parts, arranged at 90° to each other, allow loose tolerances and low cost with no compromise to positional accuracy.”



**Figure 2.23 – Final design for the baffle (after Kriegel [1994]). The double slot absorbs variation in the design without affecting the assembly along the z-axis.**

Thus, EC design provided Kriegel, Blanding, and Kelvin simple, inexpensive solutions to some very complicated problems. They all showed that controlling the constraints in a system can be beneficial and productive.

As illustrated in the examples, a designer does not always realize a design is over-constrained (or perhaps under-constrained) until it is too late. Methods have been

developed to help determine the constraint status of an assembly. For example, Adams [1998] has developed a tool to find the constraint status of a design. This method will be discussed in the next section.

## **2.8 USING SCREW THEORY FOR CONSTRAINT ANALYSIS**

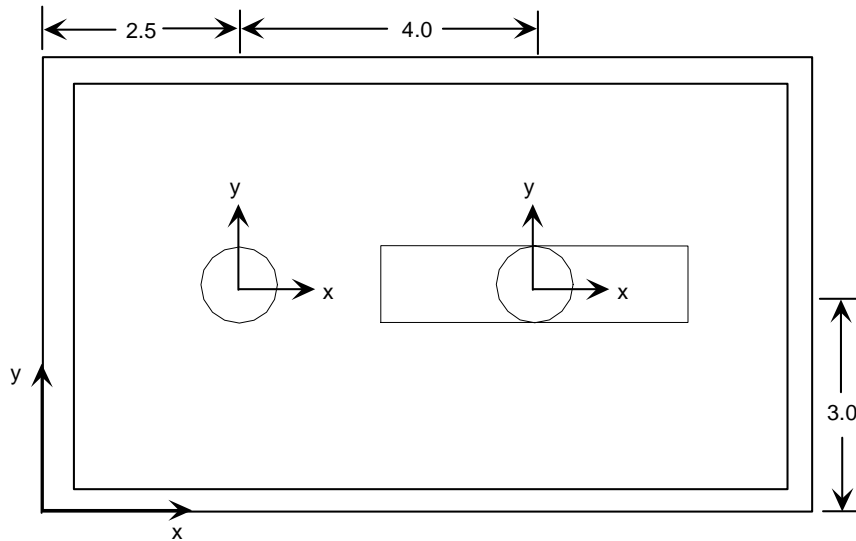
It is not the aim of this thesis to redevelop the concepts behind screw theory as a method for constraint analysis. A detailed description of the method can be understood by reading Adams [1998]. In short, screw theory [Ball 1900] can be used to find whether an assembly is under-constrained through an analysis in “twist” space or over-constrained through an analysis in “wrench” space. If the assembly is not shown through screw theory to fall into either of these cases, the design is exactly constrained.

To find whether a design is under-constrained, the designer must develop a twist matrix for each joint in an assembly. In the twist matrix, each row represents one degree of freedom *allowed* by the joint. A reciprocal operation (which is a series of matrix operations described in Appendix A) transforms each twist matrix into a wrench matrix. All the wrenches are unionized (combined into the same matrix), and the reciprocal operation is again applied to transform the matrix into a resultant twist matrix. From this twist matrix, any under-constraint can be detected.

To find whether a design is over-constrained, the designer must develop a twist matrix for each joint in the assembly (these are the same twist matrices as found above for the under-constrained analysis—they are not the resultant twist matrix). They are initially unionized, and then the reciprocal operation is applied. The resulting wrench

matrix shows the over-constraint that is present. Additional explanation can be found in Adams [1998], Adams and Whitney [2001], Whitney et. al [1999], Phillips (1984), Roth (1984), Waldron (1966), Konkar (1993), and Konkar and Cutkosky (1995).

As an example of Adams' constraint analysis method, consider the 2D assembly in Fig. 2.24. Similar examples are outlined in several documents [Adams 1998, 2001]. The assembly has a base block with two rigidly connected pegs. A top plate has two features machined out such that the left peg fits exactly into the left hole of the plate. The right peg fits in the slot. The dimensions of the hole and slot are not under consideration; however, the angle of the slot is allowed to vary from assembly to assembly. Note that the specific details and calculations for this example can be found in Appendix A.



**Figure 2.24 – Slotted block example for Screw Theory**

To determine if this assembly is exactly constrained, twist matrices must be found. To do so, a transformation matrix relates each feature to the parts in the assembly.

From these transformation matrices, a twist matrix can be found. Each twist matrix will have a row for each possible degree of freedom allowed by the joint. For brevity, only the twist matrices are shown here. More detail on how to find the twist matrix can be found in Appendix A.

$$\text{Twist}_{\text{left}} = [0 \quad 0 \quad 1 \quad 3 \quad -2.5 \quad 0]$$

$$\text{Twist}_{\text{right}} = \begin{bmatrix} 0 & 0 & 1 & 3 & -6.5 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

A motion analysis is performed to learn if the assembly is under-constrained. This entails applying the reciprocal operation (Appendix A) to each twist to form wrench matrices. These wrenches are combined into the same matrix by a union, and the reciprocal operation is applied to the unionized wrench matrix, leading to a resultant twist matrix. For Fig. 2.24, the resultant twist matrix is empty, as shown below.

$$\begin{array}{cccccc} & x & y & z & \theta_x & \theta_y & \theta_z \\ \text{Twist} = & [0 & 0 & 0 & 0 & 0 & 0] \end{array}$$

Thus, for the given setup, this assembly is not under-constrained. All degrees of freedom are constrained, and there will be no unanticipated motion.

A force analysis is performed to learn if the assembly is over-constrained. This time, the twist matrices are unionized at the beginning. Then, the reciprocal operation is applied. The first three columns in the resulting wrench matrix show *translation* in the x, y, and z directions. The last three columns show *rotation* in x, y, and z, respectively. For this example, the wrench matrix shown below illustrates that this assembly is not over-constrained. Based on the description above, there is no translation in x or y, and there is no rotation about z. As this example is only in 2D, the over-constrained directions shown in the wrench matrix (z-translation and x and y-rotation) do not apply.

$$\text{Wrench} = \begin{matrix} & \begin{matrix} x & y & z & \theta_x & \theta_y & \theta_z \end{matrix} \\ \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{matrix}$$

## **2.9 THE DIRECT LINEARIZATION METHOD USED FOR VARIATION ANALYSIS OF EC DESIGNS**

The direct linearization method [DLM] is a tolerance analysis method for assemblies [Chase, 1999]. The analysis shows the effects that tolerances from various parts have on the overall assembly. Understanding how the tolerances propagate through an assembly can help the designer choose good tolerances on dimensions for the parts. The method applied to tolerance analysis by the DLM can be adjusted to perform a “variation analysis” for EC design.

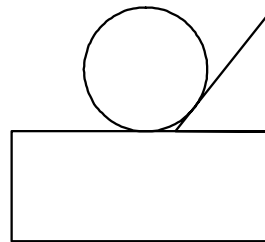
One of the major issues in manufacturing and assemblies is variability in dimensions from part to part. As has been suggested earlier and in literature, variability

is much less of an issue for EC design. The effects of dimensional variation on EC assemblies versus over-constrained (OC) assemblies will be explored (Chapter 6) using the vector loops from the DLM.

Several steps from the DLM can be used to find the effects of variation in assemblies. They are listed here, and each is briefly described.

1. Create an assembly graph.
2. Locate the datum reference frame for each part.
3. Locate the kinematic joints and create datum paths.
4. Create vector loops.
5. Generate vector loop equations.
6. Calculate derivatives and form matrix equations.
7. Solve for assembly sensitivities.

The general method will be illustrated with the simple assembly shown in Fig. 2.25.



**Figure 2.25 – Sample assembly to show variation analysis method**

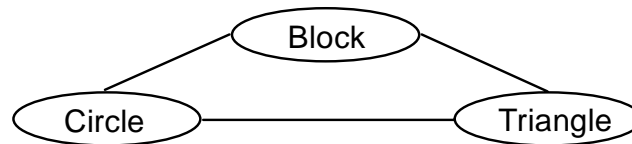
### 2.9.1 CREATE AN ASSEMBLY GRAPH

The assembly graph is a diagram representing connectivity relationships in an assembly. For the given example, the diagram looks like Fig. 2.26. The assembly graph shows that there will be one vector loop for this example.

The number of vector loops can also be determined by using a simple equation.

$$\# \text{ Loops (L)} = \# \text{ joints (J)} - \# \text{ parts (P)} + 1$$

Thus, for the example in Fig. 2.26,  $J = 3$ ,  $P = 3$ , and  $L = 1$ .



**Figure 2.26 – Assembly graph**

### 2.9.2 LOCATE THE DATUM REFERENCE FRAME FOR EACH PART

The datum reference frame (DRF) is a local coordinate system for each part. The datum reference frames for each part in Fig. 2.25 are illustrated as black solid shapes on Fig. 2.27.

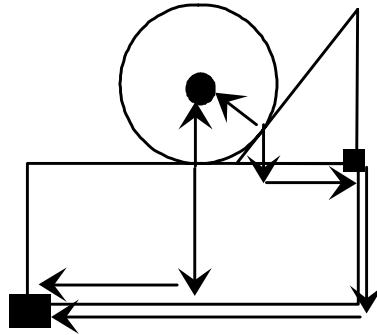
### 2.9.3 LOCATE KINEMATIC JOINTS AND CREATE DATUM PATHS

The next step is to relate all the joints to the datum reference frames established in Section 2.9.2 through vectors that form what is called a datum path. The kinematic joints are the contact points between parts. A datum path is a chain of vectors that link a joint



to all relevant datum reference frames. The chain starts at a joint and goes to all the related datum reference frames for that joint.

Figure 2.27 shows an example of the datum paths. The arrows represent the datum path vectors.



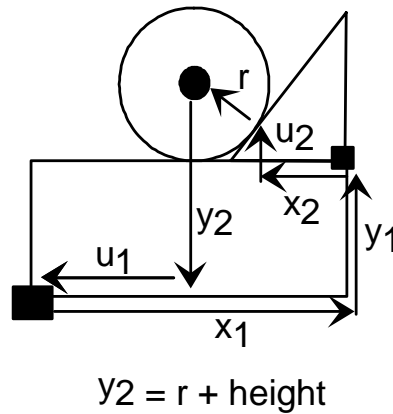
**Figure 2.27 – Datum paths**

#### **2.9.4 CREATE VECTOR LOOPS**

The vector loops are formed from the datum paths. The loops are formed by rules established in Chase [1999].

1. Enter through a joint.
2. Follow the datum paths.
3. Follow a second datum path leading to another joint.
4. Exit to the next adjacent part in the assembly.

In short, vector loops are created by linking together datum paths tip-to-tail passing through all the joints, but not passing through any part or joint twice. The vector loop for Fig. 2.25 is shown in Fig. 2.28.



**Figure 2.28 – Vector loop for the assembly**

### 2.9.5 GENERATE ASSEMBLY EQUATIONS FROM VECTOR LOOPS

Three assembly equations per vector loop can be derived by summing all the vectors in the x and y-directions and by finding the overall sum of the rotations of all vectors in the loop. The vector loops for Fig. 2.28 are shown below. A non-linear equation solver can be used to find the unknowns.

$$h_x = x_1 \cos(0) + y_1 \cos(90) + x_2 \cos(180) + u_2 \cos(90) + r \cos(90 + \theta) + y_2 \cos(270) + u_1 \cos(180)$$

$$h_y = x_1 \sin(0) + y_1 \sin(90) + x_2 \sin(180) + u_2 \sin(90) + r \sin(90 + \theta) + y_2 \sin(270) + u_1 \sin(180)$$

$$h_\theta = 0 + 90 + 90 - 90 + \theta + 180 - \theta - 90 + 180$$

## 2.9.6 CALCULATE DERIVATIVES AND FORM MATRIX EQUATIONS

In the case of tolerance analysis, only small changes in the components are of interest. Finding the unknowns as stated in Section 2.9.5 is not the final goal.

Finding small changes is easily done by linearizing the vector loop equations by a first-order Taylor series expansion. The linearized equation for  $h_x$  is shown below. All the equations can be linearized in similar fashion.

$$\delta h_x = \frac{\partial h_x}{\partial x_1} \delta x_1 + \frac{\partial h_x}{\partial y_1} \delta y_1 + \frac{\partial h_x}{\partial x_2} \delta x_2 + \frac{\partial h_x}{\partial y_2} \delta y_2 + \frac{\partial h_x}{\partial r} \delta r + \frac{\partial h_x}{\partial \theta} \delta \theta + \frac{\partial h_x}{\partial u_1} \delta u_1 + \frac{\partial h_x}{\partial u_2} \delta u_2$$

The linearized loop equations are written in matrix form. If the partial derivatives of all known variables are placed in a matrix called [A], and the partial derivatives of the unknown variables are placed in a matrix called [B], the vector loops are given by the matrix equation below.

$$[A]\{\delta X\} + [B]\{\delta U\} = \{0\}$$

## 2.9.7 SOLVE FOR ASSEMBLY SENSITIVITIES

The matrix equation can be solved to find the assembly sensitivities, as shown below.

$$\{\delta U\} = -[B^{-1}A]\{\delta X\}$$

The matrix  $[B^{-1}A]$  is the matrix of assembly sensitivities. It is a key matrix in tolerance analysis of assemblies because it represents how the dependent variables change with

small variations in the independent variables. It will be used and further explained in Chapter 5.

## ***2.10 CONCLUSIONS***

This chapter showed the history and background of EC design to better familiarize the reader with the long history of this method. It also described over-constrained and under-constrained designs. Examples were presented that helped to illustrate some of the benefits of EC design when properly used.

Rules for the placement of constraints in an EC design and a method to find the nesting force window were presented in this chapter. Chapter 3 will take this information and validate it through a quantitative means. Chapter 4 will continue to build upon that foundation by presenting a generalized method to analyze EC design.

Screw theory was applied in this chapter to determine the constraint status of a design. It will be revisited in Chapter 5 during the formulation of a method to find a measure of “goodness” for EC designs.

Finally, the background and steps for the DLM were presented in preparation for Chapters 5 and 6. In Chapter 5, it will be used to help establish a method to find the “goodness” of EC designs. In Chapter 6, it will be used to help validate the claim that EC designs are more robust than over-constrained designs.



## **CHAPTER 3 VALIDATION OF THE CURRENT RULES AND METHODS FOR EXACTLY CONSTRAINED DESIGN USING A QUANTITATIVE FOUNDATION**

### ***3.1 INTRODUCTION***

Chapter 2 presented the fundamental concepts and current methods that exist for generating exactly constrained (EC) designs. In addition, examples were given which illustrated the advantages allowed by EC design.

Kriegel's baffle example [1994] especially showed the efficiency and strength of EC design. The final solution was more robust than previous versions, and the tolerances did not control the overall design and function of the baffle. In addition, the threat of binding or tearing of key parts disappeared.

However, considering the strength of the methodology, relatively few designs use the principles of EC design. And, despite pleas to include some of the basic principles of EC design in engineering curriculum [Kriegel, 1994], the material is largely unknown to the engineering community. One reason for the apparent oversight may be the lack of a solid quantitative foundation for the rules, principles, and design methods already in practice for EC design.

If this is the case, EC design need not be so unfamiliar to the engineering world. The reason is that the mathematical premise for EC design rests in basic engineering principles. It has been independently observed by the author and others (Hammond and Parkinson [2003], Kamm [1993]) that EC designs can be analyzed quantitatively based on the following principle:

***Exactly constrained designs are statically determinate.***

With this definition in hand, the heuristic rules established by Blanding can be validated based on the equations of equilibrium. A review of the basic rules governing the application of the equations of equilibrium will be followed by several sections validating the rules and guidelines established through years of experience for EC design.

### **3.2 *STATICALLY DETERMINATE ASSEMBLIES***

Forces acting on an assembly come in two forms: applied forces and reaction forces. An applied force is an external force which pushes or pulls on the assembly, such as a nesting force. A reaction force is the resulting force at any joint or constraint of the assembly.

If an assembly is statically determinate, it means that the equations of equilibrium can be used to find all the unknown (reaction) forces in a system. It is commonly known that the sum of the forces must equal zero for a system in static equilibrium. If all of the

forces are not pointing in the same direction, they must be broken down into their directional components before they can be summed together.

$$\Sigma \mathbf{F} = 0$$

*OR*

$$\Sigma F_x = 0$$

$$\Sigma F_y = 0$$

$$\Sigma M = 0$$

Because EC designs are statically determinate, the equations of equilibrium can be used to analyze them. Later in this chapter, the equilibrium equations will be used to validate the rules that have been established for EC assemblies.

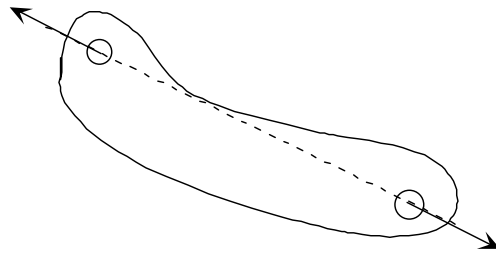
First, however, recall from Chapter 2 that no two constraints should be co-linear. Nevertheless, there are some situations where the equations of equilibrium require co-linear constraints. Therefore, a brief discussion will first explore the various force systems (two force, three force, four force, etc.) that can be analyzed using the equations of equilibrium, and this discussion will show which kinds of force systems qualify as an EC design.

### **3.2.1 TWO-FORCE MEMBERS**

Recall that Newton's laws require that the sum of the forces for a system in equilibrium must equal zero. If only two forces act on a system, those forces must then



be equal in magnitude, opposite in direction, and co-linear in placement. Otherwise, the forces would not properly balance, and some motion would occur. Figure 3.1 shows an example of a two-force member.

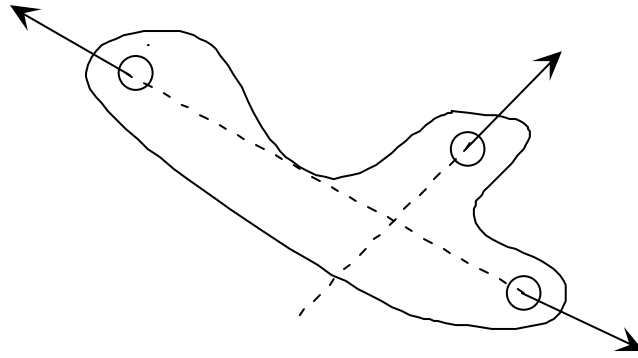


**Figure 3.1 – Two-force member**

While the two-force member will not rotate as long as the forces are co-linear, the member itself is not constrained from motion. Any applied force will cause this system to no longer be in equilibrium. Therefore, a two-force member cannot be exactly constrained.

### **3.2.2 THREE-FORCE MEMBERS**

Again, for any system in equilibrium, the forces must balance and sum to zero. Therefore, if only three forces are acting on a system, the lines of action for the forces must intersect at a point. Otherwise, the force whose line of action does not intersect the other two lines at the same point would cause a resultant moment about the instant center of the other two forces. It should be noted that none of the constraints have to be co-linear, only that all three forces must be concurrent at the same point. Figure 3.2 shows an example of a three-force member.



**Figure 3.2 – Three-force member**

As in the two-force member, there is nothing actually constraining this three-force member. It is in equilibrium as long as no outside forces displace or cause the member to rotate. It is not an EC design.

### **3.2.3 FOUR FORCES IN A SYSTEM**

Especially in 2D space, a four-force assembly provides more design flexibility than the two or three force systems because the attributes (direction, magnitude, and point of contact) of each force no longer must follow such rigid restrictions to maintain equilibrium. Equilibrium can be maintained with many different configurations of the forces. All attributes for each force can vary according to the needs or limitations of the design.

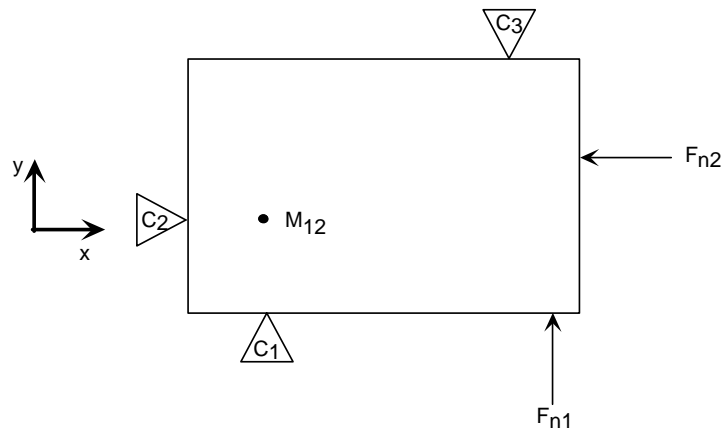
If the four forces in the system are all reaction forces, the system is statically indeterminate, and the equations of equilibrium cannot be used. If there are at most three reaction forces, which would make the final force an applied force, the system is statically determinate and the equations of equilibrium can be applied.

An EC design can be a four-force system if there are at most three reaction forces (constraints) and one applied force (a resultant nesting force). The three reaction forces must be placed in such a manner as to constrain the motion in each direction only once. The nesting force is required to keep the assembly seated.

### 3.2.4 FIVE OR MORE FORCES IN A SYSTEM

An assembly with five or more forces can only be statically determinate if there are no more than three reaction forces that constrain three different degrees of freedom. All remaining forces must be applied forces.

This condition could exist, for example, as in Fig. 3.3. In this EC design, there are three reaction forces (one at each constraint), and one nesting force perpendicular to the block in each necessary direction. This EC assembly is statically determinate.



**Figure 3.3 – Statically determinate block**

Again, four or more reaction forces leave the assembly statically indeterminate. It also makes the design over-constrained.

### **3.3 VALIDATING THE RULES FOR EXACTLY CONSTRAINED DESIGN**

With this explanation of forces in a system, two main considerations for EC design can be validated using the equations of equilibrium. First, the rules established by Blanding [1999] will be quantitatively validated by using the equations of equilibrium. Then, the nesting force window will be constructed using the equations of equilibrium, and the results will be compared to the nesting force window found by the graphical approach.

This section will validate the rules for 2D EC assemblies using simple examples in conjunction with the equations of equilibrium. As mentioned in Chapter 2, Blanding's rules for 2D assemblies can be summarized in four points [Skakoon, 2000].

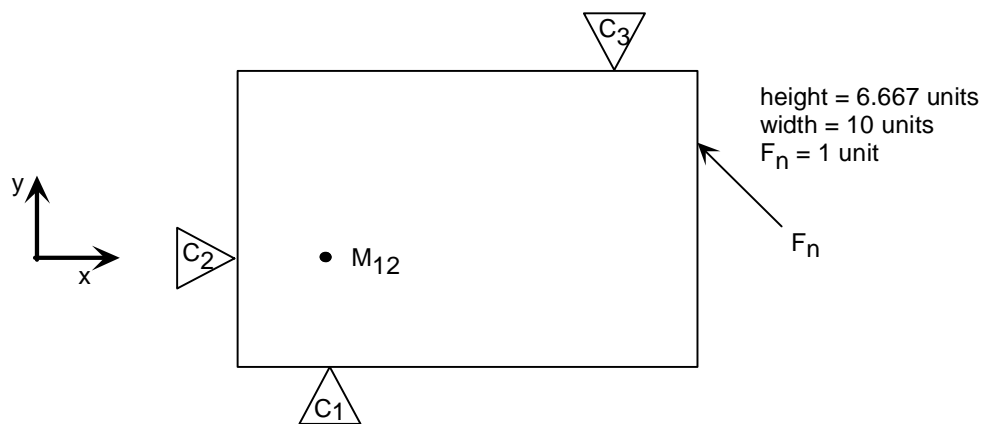
1. No two constraints should be co-linear.
2. No four constraints are in a single plane.
3. No three constraints are parallel.
4. No three constraints should intersect at a point.

#### **3.3.1 NO TWO CONSTRAINTS SHOULD BE CO-LINEAR**

The equations of equilibrium validate the rule that no two constraints should be co-linear in an EC design. Remember that there are at least four forces in an EC design:

three reaction forces and one or more nesting forces. In general, if any two of the three reaction forces become co-linear, they will not be able to resist a moment, and motion (such as “wobble”) may result. The equations of equilibrium make it easy to understand this general result: the two reaction forces of the co-linear constraints can no longer act as a force couple and resist a moment applied by the nesting force.

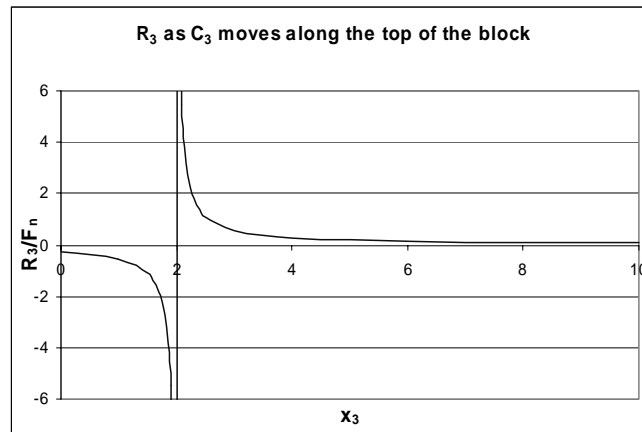
As an example, Fig. 3.4 shows a block with three edge slider constraints. The dimensions can be found in the figure. Constraints 1 and 2 do not move during the analysis; however, constraint 3 is allowed to slide along the top of the block in the x-direction. Note that while the nesting forces to seat an assembly in the translational directions are usually applied perpendicular to the surface of an assembly, for simplicity, one resultant nesting force is here placed at a 45° angle to the side of the block to provide the necessary seating in the x and y-directions.



**Figure 3.4 – Block with three constraints**

To mathematically see what happens when constraints 1 and 3 line up, a moment is taken about  $M_{12}$  (which is the instant center between constraints 1 and 2, thus allowing  $R_1$  and  $R_2$  to fall out of this stage of the analysis) to find the reaction force on constraint 3 as it moves along the top of the block. The values for  $R_3$  are plotted on a graph to see the results.

Figure 3.5 shows the results of the moment analysis in Excel<sup>®</sup>. (Please note that while the overall results are presented here, the detailed analysis may be found in Appendix B.) When constraints 1 and 3 line up, the reaction force  $R_3$  (constraint 3) necessary to keep the block immobile goes to infinity!



**Figure 3.5 – Reaction force on  $C_3$  required to keep the block immobile**

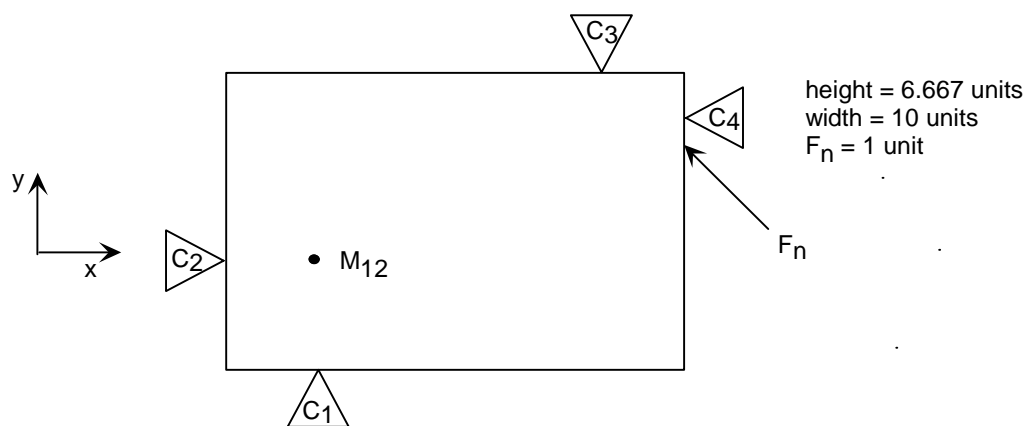
As noted earlier, the co-linear reaction forces on constraints 1 and 3 cannot resist the moment caused by the nesting force. As will be described in more detail later in this thesis, the block becomes over-constrained in the y-direction, and under-constrained in rotation.

This simple example quantitatively demonstrates the information presented in the first paragraph of this section. Although only one simple example has been presented, the result is general. No two constraints should be co-linear.

It should also be noted that this example violates another rule when two constraint lines become co-linear: no three constraint lines should intersect at a point. These two rules are simultaneously violated because of the use of the edge slider joints to constrain the assembly. The rule that no three constraint lines should intersect at a point will be investigated later in this chapter.

### 3.3.2 NO FOUR CONSTRAINTS ARE IN A SINGLE PLANE

Figure 3.6 shows a similar block to Fig. 3.4, only now there are four constraints instead of just three. A quick glance at the number of reaction forces shows that this block has four unknowns. The equations of equilibrium only provide three equations for three unknowns, and thus the block is statically indeterminate.



**Figure 3.6 – Block with four constraints**

A brief look at the constrained degrees of freedom can also show the over-constraint that is present. Constraint 1 eliminates translation in the y-direction. Constraint 2 eliminates translation in the x-direction. Adding constraint 3 now eliminates any rotation. The nesting force ensures that contact is maintained with each constraint. Thus, constraint 4 is simply competing with constraint 2 to eliminate translation in the x-direction, and this block is over-constrained. Skakoon states, “Since there are only three constraints required or possible in one plane, four would be over-constrained” [2002].

In general, if there are more unknowns than equations of equilibrium, the assembly is statically indeterminate and over-constrained. Additional information would be required to find all the reaction forces.

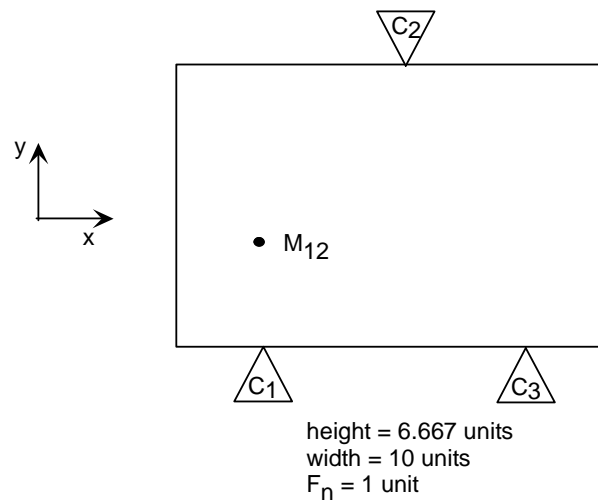
In order for this part to correctly assemble, either some type of deformation to the block would be required for all four constraints to be touching it, or the tolerances would have to be very tight to ensure a perfect fit. Regardless, no four constraints are allowed in a single plane for an EC design.

### **3.3.3 NO THREE CONSTRAINTS ARE PARALLEL**

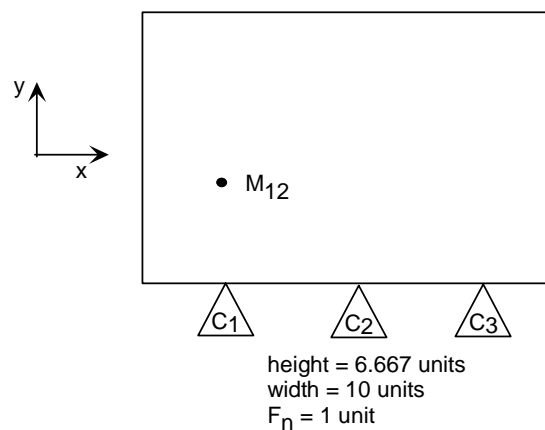
Figures 3.7 and 3.8 show equivalent assemblies. Each assembly has three parallel constraint lines. A simple look at the degrees of freedom shows the block is over-constrained in the y-direction, and under-constrained in the x-direction.



When trying to apply the equations of equilibrium to these assemblies, the force equation in the x-direction disappears. It will not sum to zero if the nesting force is applied in any way to the x-direction. Regardless, there remain three reaction forces to be found by two equations (one equation to sum the forces in the y-direction and one moment equation), and the assembly is again statically indeterminate.

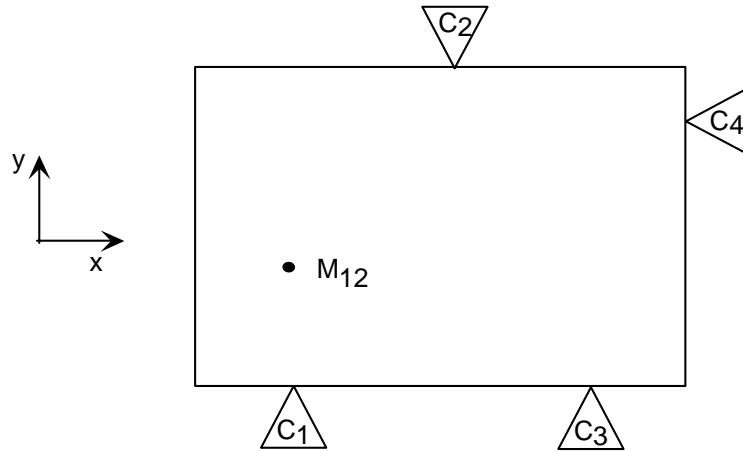


**Figure 3.7 – Block assembly with three parallel constraints**



**Figure 3.8 – Similar block assembly with three parallel constraints**

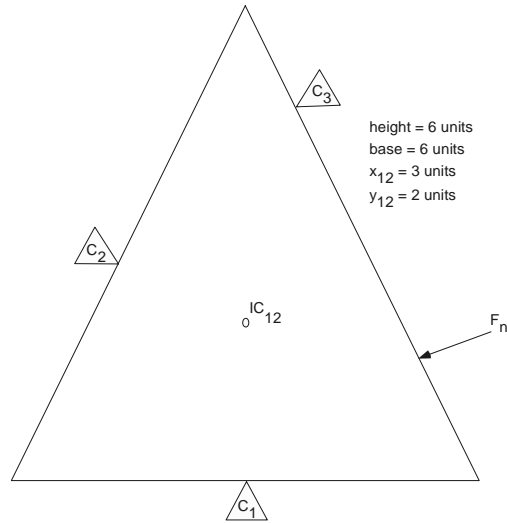
Suppose another constraint is added, as in Fig. 3.9, to inhibit motion in the x-direction. The over-constraint in the y-direction does not go away.



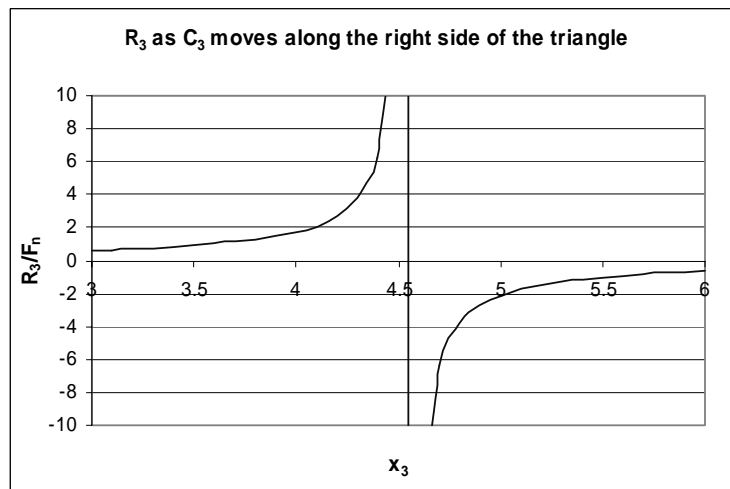
**Figure 3.9 – Adding an x-constraint to the block**

### 3.3.4 NO THREE CONSTRAINTS SHOULD INTERSECT AT A POINT

Figure 3.10 shows a simple triangle with three constraints. Constraint 3 is allowed to move along the right side of the part, while constraint 1, constraint 2, and the nesting force remain fixed in the positions shown. The reaction forces are found using the equations of equilibrium, and the baseline results for  $R_3$  are shown in Fig. 3.11. Again, the detailed results can be found in Appendix B.



**Figure 3.10 – Triangle assembly with three constraints**



**Figure 3.11 – Reaction force on  $C_3$  required to keep the block immobile**

As in the case with two co-linear constraints, when all three constraint lines intersect at a point, the assembly cannot resist a moment caused by the nesting force. This fact is shown in Fig. 3.11 where  $R_3$  goes to infinity when the three constraint lines

intersect. The equations of equilibrium thus show that no three constraints should ever intersect at a point for an EC design.

Although shown here through a simple example, this result is general for all 2D EC assemblies when three constraint lines intersect at a point. In this case, the equations of equilibrium show that the moment equation goes to zero on the left hand side (all reaction forces go to zero because the sum of the moments about the point of intersection— $IC_{12}$ —leaves no reaction forces); however, when the nesting force is applied, the right hand side of the moment equation is no longer zero. The assembly cannot resist the moment caused by the nesting force, and the equilibrium conditions are not satisfied.

### 3.3.5 SUMMARY OF THE RULES

Table 3.1 gives a summary of the results outlined above. While each rule was only illustrated with one simple example, these rules are general and hold for all EC assemblies.

**Table 3.1 – Summary of the results for the rules**

<b>Rule</b>	<b>Why it will not work for EC design</b>
No two constraints should be co-linear	Moment equation will not sum to zero
No four constraints are in a single plane	Statically indeterminate
No three constraints are parallel	Statically indeterminate
No three constraints should intersect at a point	Moment equation will not sum to zero

### **3.4 NESTING FORCE WINDOW**

One of the most important considerations in EC design relates to the placement of the nesting force. As mentioned in Chapter 2, there is a “window” that shows the appropriate locations where a nesting force may be applied.

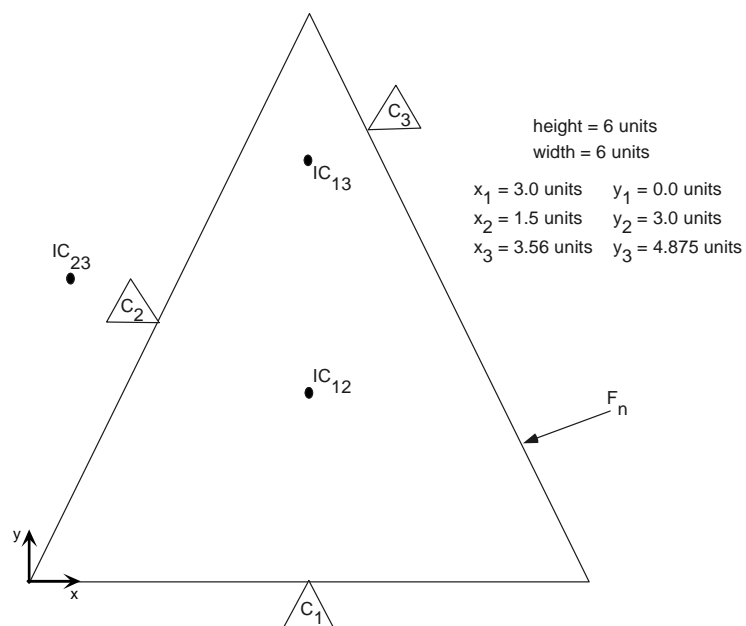
This section will use the equations of equilibrium to illustrate how to find the window in a quantitative fashion. This quantitative approach will then be compared to the graphical approach presented in Chapter 2.

#### **3.4.1 QUANTITATIVE APPROACH TO FIND THE NESTING FORCE WINDOW**

The equations of equilibrium provide a straightforward, quantitative approach to find the nesting force window. For this thesis, each constraint is represented as a reaction force in *compression* on the block. The nesting force is an applied force, also in compression. All of the forces are summed in the x and y-directions, and a moment is taken about some point on the assembly. Solving these equations will find the reaction forces, given a determinate system of constraints.

If any of the reaction forces are in *tension* (recognized as a negative value from the equations) for a given placement of the nesting force, that point on the assembly is not allowed in the window. Conversely, for any point along the assembly where all reaction forces meet the compression criteria (all forces are positive), that position is a valid point in the nesting force window.

Figure 3.12 will now be used to show how the nesting force window can be found using the equations of equilibrium. All the constraints remain fixed in the given positions, and the nesting force is allowed to slide along each edge of the assembly. The reaction forces are calculated for several points along the path of the nesting force in order to determine the allowable window. Only one nesting force is applied, and it is always perpendicular to the surface/edge of the assembly. Note that only one nesting force is necessary because, coupled with the directions of the reaction forces, it provides the necessary directions of force to seat the assembly.



**Figure 3.12 – Triangle assembly example used to find the nesting force window**

To begin, a moment is taken about point  $IC_{12}$ , and the appropriate equations for  $R_3$  are developed. Note that in this example, “left” refers to when the nesting force moves along the left edge of the assembly, “right” refers to when the nesting force moves

along the right edge of the assembly, and “base” refers to when the nesting force moves along the base edge of the assembly.

$$R_{3\text{-LEFT}} = \frac{F_n \cos(\theta) * (y_{12} - y_n) + F_n \sin(\theta) * (x_{12} - x_n)}{\sin(\theta) * (x_3 - x_{12}) - \cos(\theta) * (y_3 - y_{12})}$$

$$R_{3\text{-RIGHT}} = \frac{F_n \cos(\theta) * (y_n - y_{12}) - F_n \sin(\theta) * (x_n - x_{12})}{\sin(\theta) * (x_3 - x_{12}) - \cos(\theta) * (y_3 - y_{12})}$$

$$R_{3\text{-BASE}} = \frac{F_n (x_n - x_{12})}{\sin(\theta) * (x_3 - x_{12}) - \cos(\theta) * (y_3 - y_{12})}$$

To find the window for the whole system, the other two reaction forces must also be found. Thus, forces are summed in the x-direction to find  $R_2$ . And finally, the forces are summed in the y-direction to find  $R_1$ .

$$R_{2\text{-LEFT}} = \frac{F_n \cos(\theta) - R_3 \cos(\theta)}{-\cos(\theta)}$$

$$R_{2\text{-RIGHT}} = \frac{F_n \cos(\theta) + R_3 \cos(\theta)}{\cos(\theta)}$$

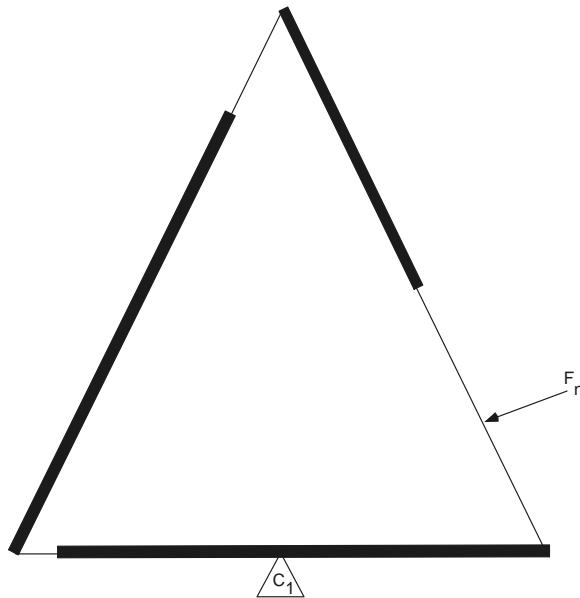
$$R_{2\text{-BASE}} = R_3$$

$$R_{1\text{-LEFT}} = F_n \sin(\theta) + R_2 \sin(\theta) + R_3 \sin(\theta)$$

$$R_{1\text{-RIGHT}} = F_n \sin(\theta) + R_2 \sin(\theta) + R_3 \sin(\theta)$$

$$R_{1\text{-BASE}} = -F_n + R_2 \sin(\theta) + R_3 \sin(\theta)$$

For simplicity, Fig. 3.13 shows the results of the equations in a graphical representation based on the detailed calculations shown in Appendix B. The bolded lines in Fig. 3.13 show the points along the figure where the nesting force is *NOT* allowed because the reaction forces as found in the equations of equilibrium do not all stay positive. All other points on the assembly are valid positions for the nesting force because the reaction forces show that the constraints stay in contact with the part.

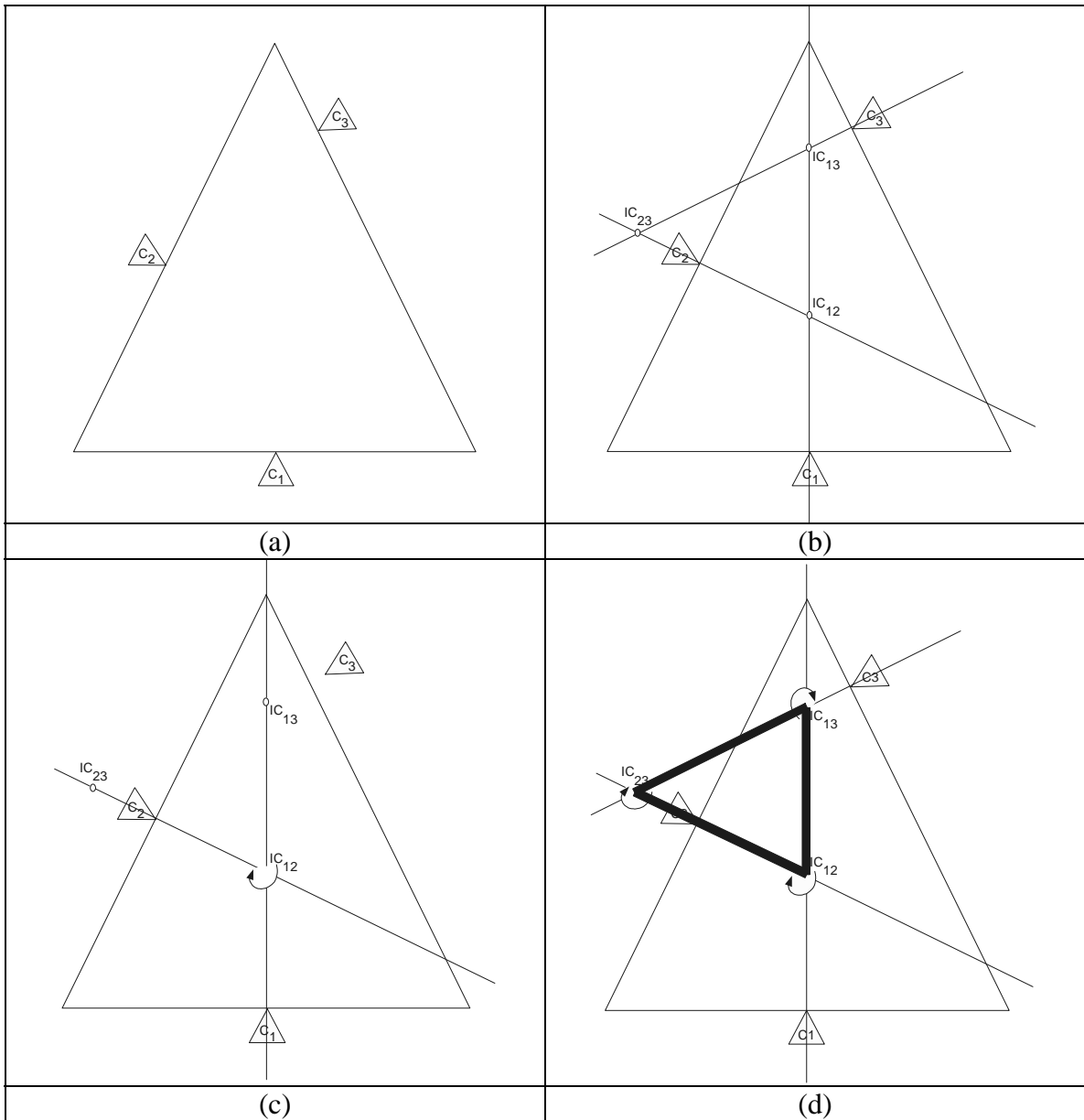


**Figure 3.13 – Nesting force window according to the equations of equilibrium**

### **3.4.2 COMPARISON BETWEEN GRAPHICAL AND QUANTITATIVE APPROACHES**

Recall from Chapter 2 that Blanding uses a graphical method to find the nesting force window based on instant centers, constraint lines, proper rotations, etc. Figure 3.14 shows the window as found by the graphical method for the triangle assembly example.

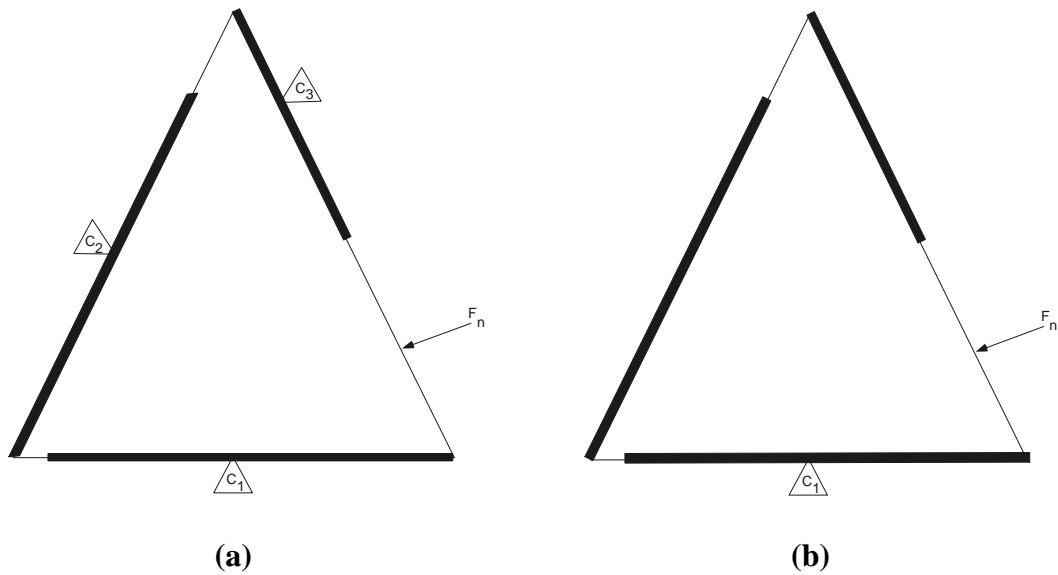




**Figure 3.14 – Finding the nesting force window using the graphical method (a) An assembly with three constraints (b) Finding instant centers for the constraints (c) Removing constraint  $C_3$  to find the rotation the nesting force must exert on the part to restore contact with the constraint. (d) The nesting force window—neither the nesting force nor the line of action of the nesting force can pass through the darkened triangle.**

How does the window found by the equations of equilibrium compare to that found by the graphical method? Before this question can be answered, Fig. 3.14d must

be transformed to match the representation of Fig. 3.13. This transformation is simply accomplished by applying a force to each segment of the assembly. If the force maintains the proper rotation through all necessary instant centers, that segment of the assembly boundary is allowed. Figure 3.15 shows a comparison of the two methods, and it appears that the two methods match.



**Figure 3.15 – Nesting force window comparison (a) The graphical method transformed (b) The window found with the equations of equilibrium**

In order to determine just how similar the windows are to each other, the point at which the window transitions from the acceptable to unacceptable region was found for the graphical method and compared to the point of transition found from the equations of equilibrium. The detailed explanation and results of this analysis can be found in Appendix C. From the analysis, the results show that the nesting force windows are the same, within round-off error.

Thus, the nesting force window can effectively be found using the equations of equilibrium. They in fact provide a more fundamentally basic approach to find the window because they show the designer when and why the design will or will not work according to the position of the nesting force.

### **3.5 CONCLUSIONS**

In conclusion, the equations of equilibrium become a very simple yet powerful tool by which to analyze EC design. Both the rules established through years of experience and the nesting force window were validated using these basic equations.

However, looking at every point along the surface of the assembly is rather cumbersome and tedious. Chapter 4 will present a generalized method using the equations of equilibrium to more efficiently find both the nesting force window and the configuration(s) when assemblies violate EC design rules.

## **CHAPTER 4 GENERALIZED METHOD TO USE THE EQUATIONS OF EQUILIBRIUM IN EXACTLY CONSTRAINED DESIGN**

### ***4.1 INTRODUCTION***

Chapter 3 provided a quantitative validation for exactly constrained (EC) design based on the equations of equilibrium for the rules governing the placement of constraints and the nesting force window. Various simple examples illustrated how the rules established through heuristics and years of experience agree with the results from the equations of equilibrium.

Chapter 3 also identified two main design considerations for EC design. First, the location of the constraints must not violate any EC rules. Second, the nesting force can only be placed in a certain region, denoted as the “nesting force window.” With the equations of equilibrium, both these considerations may be analyzed and predicted.

The work in Chapter 3 applied the equations of equilibrium from point-to-point along every surface in an assembly to find the resultant reaction forces given a certain nesting force. Validating the rules with simple examples adds strength and integrity to the existing methods, and it begins to lay the quantitative foundation for EC design.

However, it is quite inefficient to analyze each point along the surface of an assembly to determine whether the design meets the criteria for EC design. A stronger foundation could be built by generalizing the method to analyze EC design. Rather than look at each point along the surface of the assembly, a more efficient method utilizes the equations of equilibrium to inspect or predict the behavior of a design.

Chapter 4 presents a generalized method to quantitatively analyze an EC design. First, the equations of equilibrium are set up in a general matrix form,  $\mathbf{Cr} = \mathbf{b}$ , based on the initial locations of the constraints in the design. Then, a general method will be introduced to find the nesting force window, followed by a general approach to either inspect or predict the behavior of an EC design.

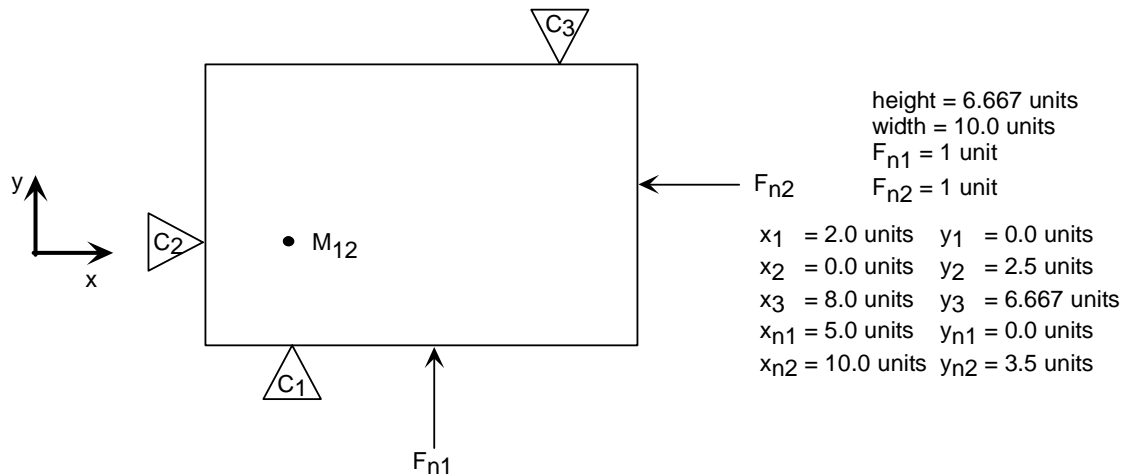
#### ***4.2 INITIALIZING THE ANALYSIS FOR EC DESIGN***

The first step to generalize the quantitative method to analyze EC design is to set up the equations of equilibrium in matrix form,  $\mathbf{Cr}=\mathbf{b}$ . To initialize the set-up of the matrix, all necessary information for each reaction and nesting force must be known or assumed.

Each force is defined by three attributes: a magnitude, a direction, and a point of contact. The magnitudes of the reaction forces are unknown, but they can be found by solving the equations of equilibrium, given the nesting force(s). For the examples presented in this chapter, the positive direction for the reaction and nesting forces are assumed to be in compression. The point of contact for each reaction force may be known or assumed. Initially, the point of contact for the nesting force must be assumed,

but a method to find an acceptable point of contact will be presented in Section 4.3, and it will be more fully utilized in Section 4.4.

With the initial information defined, the generalized process can begin by formulating the equations of equilibrium and setting them up in matrix form to find the reaction forces. Figure 4.1 introduces an example that will be used throughout the chapter to illustrate each step in the general process.



**Figure 4.1 – Block with 3 constraints to be used for generalized method**

This assembly uses two nesting forces, thus allowing them to be perpendicular to the block, while still seating the assembly in both the x and y-directions. Remember that the positions of the nesting forces are assumed. It would be just as reasonable to assume that the vertical nesting force, denoted  $F_{n1}$ , could be along the top of the block.

From the information provided in Fig. 4.1, the equations of equilibrium are set up in matrix form,  $\mathbf{C}\mathbf{r} = \mathbf{b}$ . The moment equation is summed about the point  $M_{12}$ , which corresponds to the coordinate values for  $x_1$  and  $y_2$ . After forming the matrix equation, it can be solved to find the reaction forces.

$$+\curvearrowright \sum M_{12}=0 \quad -R_3(x_3 - x_{12}) + F_{n1}(x_{n1} - x_{12}) + F_{n2}(y_{n2} - y_{12}) = 0$$

$$+\rightarrow \sum F_x=0 \quad R_2 - F_{n2} = 0$$

$$+\uparrow \sum F_y=0 \quad R_1 - R_3 + F_{n1} = 0$$

$$\begin{bmatrix} 0 & 0 & -(x_3 - x_{12}) \\ 0 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -F_{n1}(x_{n1} - x_{12}) - F_{n2}(y_{n2} - y_{12}) \\ F_{n2} \\ -F_{n1} \end{bmatrix}$$

$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & (x_{12} - x_3) \\ 0 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix}^{-1} \begin{bmatrix} -F_{n1}(x_{n1} - x_{12}) + F_{n2}(y_{12} - y_{n2}) \\ F_{n2} \\ -F_{n1} \end{bmatrix}$$

The information from Fig. 4.1 can be plugged into the matrix equation above to find the numerical values for the reaction forces. They are shown below.

$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -0.333 \\ 1 \\ 0.667 \end{bmatrix}$$

While setting up the equations of equilibrium as a system of linear equations is not new, it is a very important step in generalizing the design method. Using this basic process to find the reaction forces, an efficient method to find the nesting force window can be presented.

### **4.3 GENERALIZED METHOD TO FIND THE NESTING FORCE WINDOW**

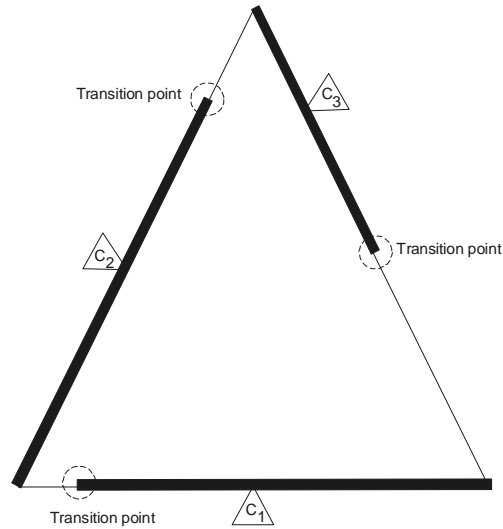
An applied nesting force can only sit within a specific range of the assembly. It will be shown later on that the nesting force window depends upon the location of the constraints. However, for a given placement of the constraints, the nesting force window can be found.

Recall in Chapter 3 that the nesting force window was found by analyzing many points along each side of the assembly. However, here, a more general approach is presented where the nesting force window is generated based on transition points.

#### **4.3.1 DEFINITION OF TRANSITION POINTS**

To begin generalizing the method to find the nesting force window requires the understanding of one significant term: a transition point. The transition point is any point on an assembly at which the unacceptable portion of the nesting force window transitions to an acceptable region of the window. Figure 4.2 shows the transition points found in Fig. 3.13.





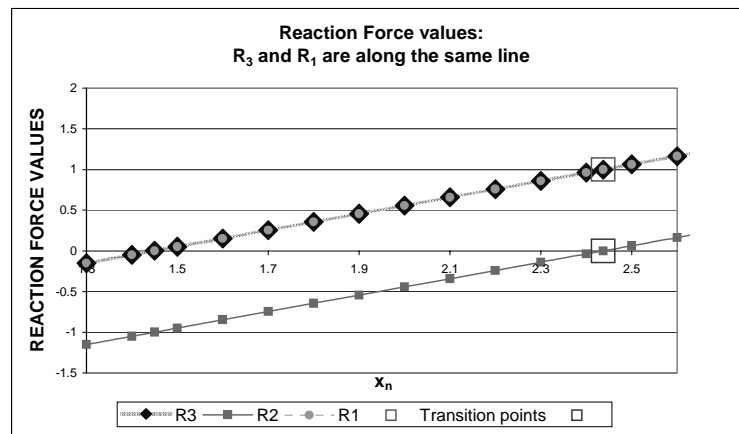
**Figure 4.2 – Transition points marked on the triangle assembly from Fig. 3.13**

The *acceptable* region of the nesting force window shows the possible locations for the nesting force that make all reaction forces greater than zero. This definition physically means that all constraints would be in compression on the part.

The *unacceptable* region of the nesting force window shows the possible locations for the nesting force that make any one or more of the reaction forces negative. This definition physically means that one or more of the constraints would need to be in tension, instead of compression, for equilibrium.

The point at which the acceptable region and the unacceptable region come together is the transition point. To further clarify, a portion of the point-by-point force analysis from Chapter 3 is shown in Fig. 4.3. This section of the force analysis corresponds to the transition point located along the left surface of the assembly in Fig.

4.2. Notice in Fig. 4.3 that  $R_1$  and  $R_3$  turn positive when the x-coordinate of the nesting force,  $x_n$ , is about 1.45 units. However,  $R_2$  is still negative.  $R_2$  does not transition out of the negative region until  $x_n$  is at 2.4375 units. Based on the requirement defined earlier for this thesis that all reaction forces must stay in compression to the main part, the transition point does not happen until all reaction forces have turned positive; therefore, the transition point in this case is at (2.4375, 4.875). Notice that the value of  $R_2$  at the transition point is zero; therefore, the transition point itself is technically in the infeasible region.



$x_n$	$y_n$	$R_3$	$R_2$	$R_1$
1.3	2.6	-0.14959	-1.149592	-0.14959
1.4	2.8	-0.04853	-1.048529	-0.04853
<b>1.45</b>	<b>2.9</b>	<b>0.002003</b>	<b>-0.997997</b>	<b>0.002003</b>
1.5	3	0.052534	-0.947466	0.052534
1.6	3.2	0.153597	-0.846403	0.153597
1.7	3.4	0.25466	-0.74534	0.25466
1.8	3.6	0.355723	-0.644277	0.355723
1.9	3.8	0.456786	-0.543214	0.456786
2	4	0.557849	-0.442151	0.557849
2.1	4.2	0.658912	-0.341088	0.658912
2.2	4.4	0.759975	-0.240025	0.759975
2.3	4.6	0.861038	-0.138962	0.861038
2.4	4.8	0.962101	-0.037899	0.962101
<b>2.4375</b>	<b>4.875</b>	<b>1</b>	<b>0</b>	<b>1</b>
2.5	5	1.063164	0.0631644	1.063164
2.6	5.2	1.164227	0.1642274	1.164227
2.7	5.4	1.26529	0.2652905	1.26529
2.8	5.6	1.366353	0.3663535	1.366353
2.9	5.8	1.467417	0.4674165	1.467417
3	6	1.56848	0.5684795	1.56848

Figure 4.3 – Transition points shown in a force analysis

To quickly find the nesting force window, the x and y-coordinates of the transition points on each surface must be found. It is now possible to present a simple, yet powerful method to efficiently find the nesting force window.

#### **4.3.2 PRESENTATION OF THE GENERALIZED METHOD**

By understanding the transition points, the nesting force window is generated through a series of simple steps.

1. Find all possible transition points from the equations of equilibrium.
2. Determine which side of the transition point is allowed and which is not.
3. If desirable, draw the window on a sketch of the assembly.

#### **STEP 1: FIND THE TRANSITION POINTS**

The first step to efficiently define the nesting force window is to find the transition points. Recall from Fig. 4.3 that the transition point is simply where one reaction force has gone to zero, and the other reaction forces are positive. Therefore, rather than evaluate every point on the surface of the assembly, it is only necessary to find if and where each reaction force goes to zero along *each* surface.

To find the transition point, therefore, first requires that the equations of equilibrium be set-up in matrix form as outlined in Section 4.2. After formulating the matrix equations for each possible surface where the nesting force may be placed, any simple iteration routine may be used to find the transition point.

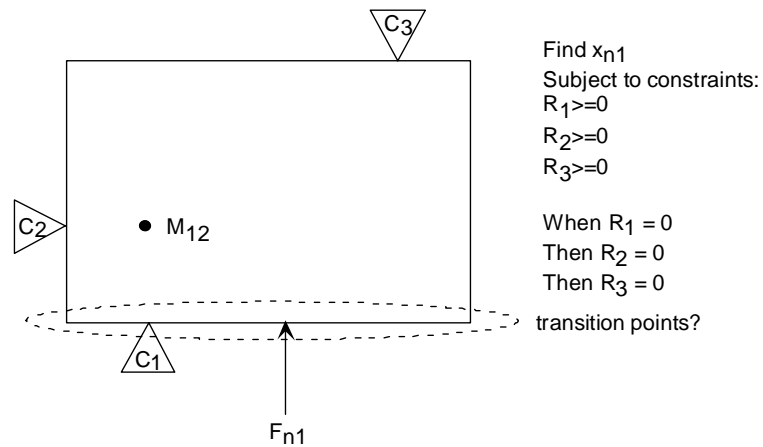
For this thesis, a simple and well defined optimization routine called the generalized reduced gradient (GRG) method (as found in Microsoft Excel's Solver<sup>®</sup>) will be used to find the transition points. As the equations of equilibrium represent a system of linear equations, the GRG algorithm will easily converge to the transition points.

As with any optimization routine, the design variables and design functions (optimization constraints and design objective) must be defined. The design variables are the x and/or y-coordinates of the nesting force. Remember that an initial point of contact for the nesting force is assumed in the set-up. That initial guess gives the optimization routine somewhere to begin. In many cases, once the x-coordinate has been chosen, the y-coordinate is set based on the geometry of the surface; therefore, an optimization constraint may define any relationship between the x and y-coordinates of the assembly surface. Three additional optimization constraints are defined to require the reaction forces to stay greater than or equal to zero. The objective function is to make each reaction force go to zero on each surface of the assembly. Thus, the optimization routine will be run three times per surface to find all possible transition points.

It is worthy to note that the objective function is not defined in the traditional sense. Usually, the objective function would be "maximized" or "minimized". However, using the capabilities of Excel<sup>®</sup>, it is possible to drive the reaction force to a specific target value. As the transition points will happen when one reaction force is at a value of zero, the objective function for this routine will be to drive each reaction force to a target value of zero.

The transition points will now be found for the assembly in Fig. 4.1. Because there are two nesting forces, the nesting force window analysis will be performed in two steps. The transition points related to the vertical nesting force will be found first. Then, the horizontal nesting force will be added to the assembly to find its transition points. This method is a form of superposition, and it can be shown to work the same as having both nesting forces applied from the start and iterating through locations until the transition points can be found. This approach will be used to maintain clarity and simplicity.

Figure 4.4 shows the assembly as it will be analyzed to find the transition points along the base. Applying the values shown in Fig. 4.1 generates the matrix equation to be used for this optimization routine.



**Figure 4.4 – Finding transition points for the bottom surface**

$$\begin{bmatrix} 0 & 0 & -0.6 \\ 0 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -0.3 \\ 0 \\ -1 \end{bmatrix}$$

$$\mathbf{Cr} = \mathbf{b}$$

$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -1.67 & 0 & 1 \\ 0 & 1 & 0 \\ -1.67 & 0 & 0 \end{bmatrix} \begin{bmatrix} -0.3 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 0 \\ 0.5 \end{bmatrix}$$

$$\mathbf{r} = \mathbf{C}^{-1}\mathbf{b}$$

Notice that  $R_1$  is negative. The negative sign gives the indication that the initial placement of the nesting force ( $x_{n1} = 5.0$  units) is not in the nesting force window. The initial guess happens to be in the infeasible region.

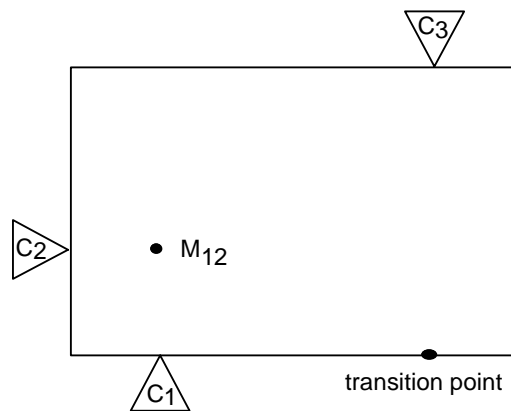
Using Excel's Solver<sup>®</sup> to find the transition points, the first objective function (called the target) will be  $R_1$ . It must equal a value of 0.0. The design variable (changing cell) will be  $x_{n1}$  (which currently sits at 5.0 units). The only optimization constraints defined will be that each reaction force must be greater than or equal to zero ( $R_1 \geq 0$ ,  $R_2 \geq 0$ ,  $R_3 \geq 0$ ). Solving the routine finds where the nesting force resides when  $R_1$  goes to zero.

$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \text{ when } x_{n1} = 8.0 \text{ units, } y_{n1} = 0.0 \text{ units}$$

As all reaction forces are greater than or equal to zero, there is a transition point along the base of the block at (8.0, 0.0).

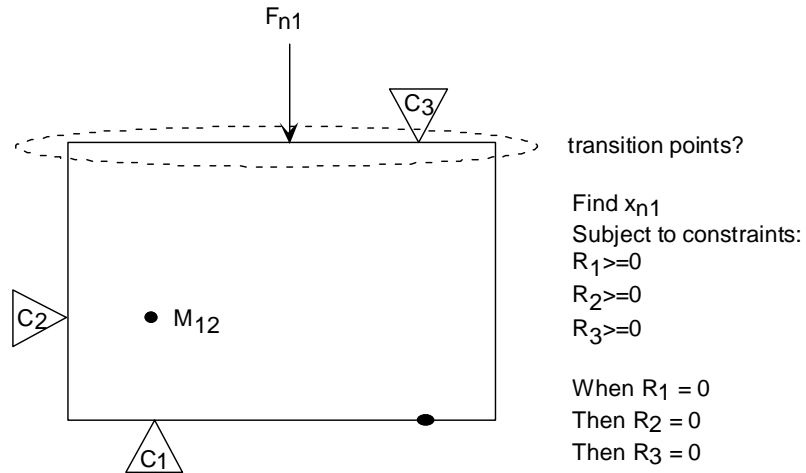
Note that because the horizontal nesting force has not yet been added onto the problem,  $R_2$  will always be zero. It is not necessary to investigate it as an objective function at this point.

Thus, the next optimization routine for the base surface of this assembly will be to find when  $R_3$  equals a value of 0.0. However, because of the optimization constraints (all reaction forces must be greater than or equal to zero), the routine finds that there are no transition points for when  $R_3$  is driven to zero along the base. Thus, only one transition point is found along the base of the block, as shown in Fig. 4.5.



**Figure 4.5 – Transition point found along the base**

Next, the transition points along the top of the block will be found. Figure 4.6 shows the assembly with the problem definition for the nesting force window along the top of the block.



**Figure 4.6 – Finding the transition points for the top surface**

Equations are reformulated based on the nesting force's current position. The initial coordinates of the nesting force on the top surface are  $x_{n1} = 5.0$  units and  $y_{n1} = 6.67$  units.

$$\begin{bmatrix} 0 & 0 & -0.6 \\ 0 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 0.3 \\ 0 \\ 1 \end{bmatrix}$$

$\mathbf{Cr} = \mathbf{b}$

$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -1.67 & 0 & 1 \\ 0 & 1 & 0 \\ -1.67 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.3 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0 \\ -0.5 \end{bmatrix}$$

$\mathbf{r} = \mathbf{C}^{-1}\mathbf{b}$

The transition points can now be found.  $R_1$  will be driven to a value of zero (objective function) as  $x_{n1}$  is allowed to translate along the top surface. However, as  $R_1$

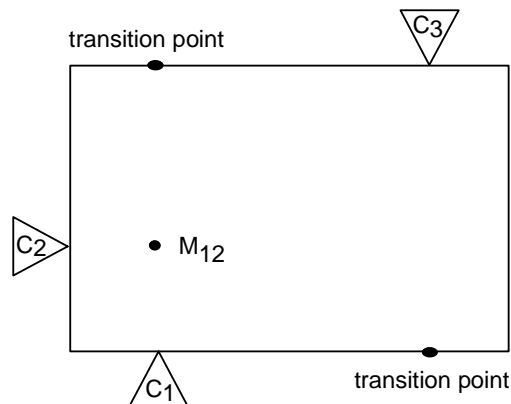


drives to zero, the optimization constraints cannot be satisfied. There are no transition points found from this optimization run.

Now, the other two reaction forces must be driven to zero. Again,  $R_2$  will always equal zero for the given configuration because there is no horizontal nesting force assumed on the block yet. Therefore, only  $R_3$  is left to be driven to zero in order to find any transition points.

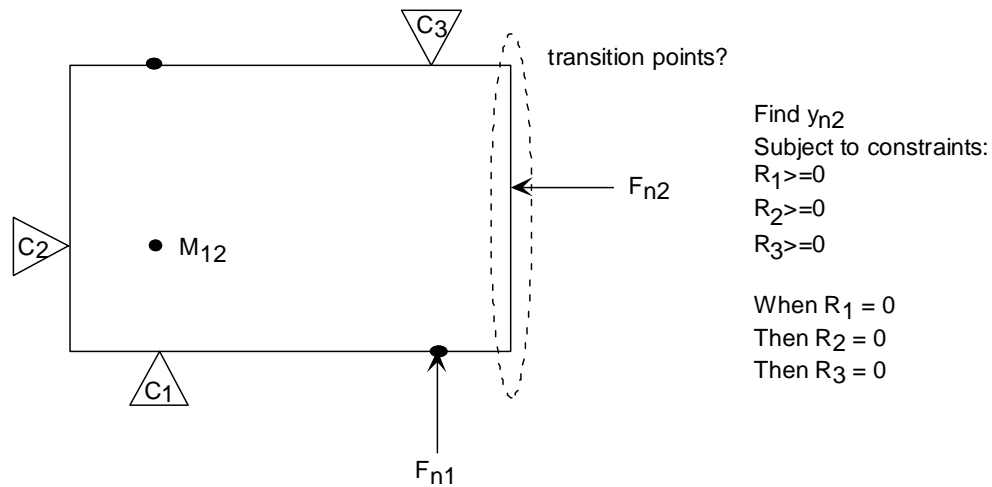
$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{ when } x_{n1} = 2.0 \text{ units, } y_{n1} = 6.67 \text{ units}$$

All reaction forces are greater than or equal to zero, and this point is a transition point for the nesting force window. Figure 4.7 shows all the transition points currently found.



**Figure 4.7 – Transition points along the top and bottom surfaces**

Now that the transition points for  $F_{n1}$  have been found, the horizontal nesting force,  $F_{n2}$ , will be added to the assembly. Notice that  $F_{n2}$  cannot be placed on the left side of the block because it would cause the block to displace in the x-direction. Figure 4.8 shows the problem definition to find any transition points associated with the right surface of the block.



**Figure 4.8 – Finding the transition points for the right surface**

After substituting in the coordinate values as shown in Fig. 4.1 (except now  $F_{n1}$  is placed at  $x_{n1} = 8.0$  and  $y_{n1} = 0.0$ ), the reaction forces are found (see below). Notice that they are positive, and the initial guess for  $F_{n2}$  lies in the feasible region.

$$\begin{bmatrix} 0 & 0 & -6 \\ 0 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -7 \\ 1 \\ -1 \end{bmatrix}$$

**Cr = b**

$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -0.167 & 0 & 1 \\ 0 & 1 & 0 \\ -0.167 & 0 & 0 \end{bmatrix} \begin{bmatrix} -7 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.167 \\ 1 \\ 1.167 \end{bmatrix}$$

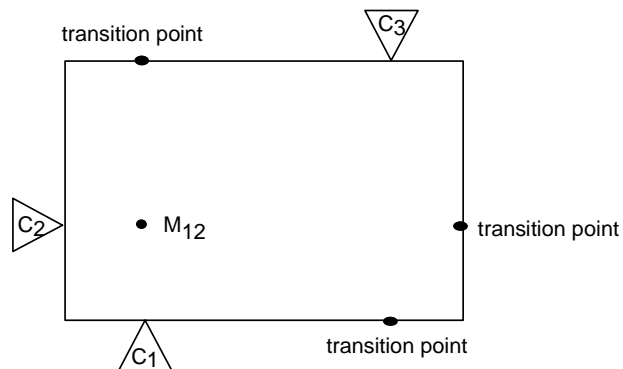
$$\mathbf{r} = \mathbf{C}^{-1}\mathbf{b}$$

The optimization routine can now find the transition points on the right surface.

First,  $y_{n2}$  will be allowed to change as  $R_1$  goes to zero.

$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \text{ when } x_{n2} = 10 \text{ units, } y_{n2} = 2.5 \text{ units}$$

All values are greater than or equal to zero, and thus (10, 2.5) is a transition point. Note that  $R_2$  cannot have any other value than 1.0 because the forces must balance. Finally, running the routine to find the value of  $y_{n2}$  when  $R_3$  goes to zero shows that  $R_3$  never goes to zero. Thus, only one transition point is found. Figure 4.9 shows the assembly with all the transition points.



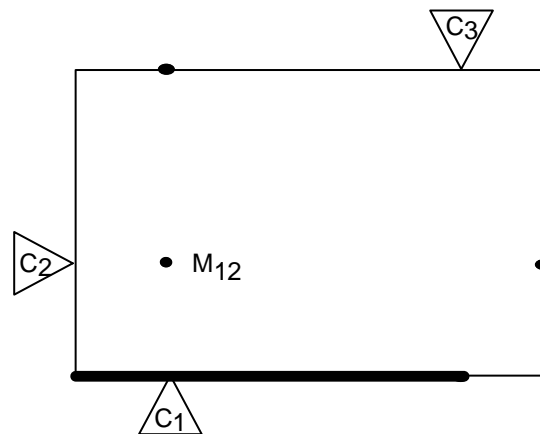
**Figure 4.9 – The block assembly with all transition points**

With all possible transition points found, the next stage in the process is to find which side of each transition point is acceptable and which is not. That process will now be explained.

**STEP 2: DETERMINE WHICH SIDE OF THE TRANSITION POINT IS ALLOWED AND WHICH SIDE IS NOT**

To find the acceptable side of each transition point requires an examination of the reaction forces for some other point along each surface. A good point to use would be the initial guess from step 1 for the location of the nesting force.

For example, recall that while finding the transition point for the base of the block, the initial nesting force position ( $x_{n1}=5.0$  units) led to a negative reaction force. Therefore, the nesting force is not allowed to the left of the transition point on the base surface. Figure 4.10 illustrates this unacceptable region as a thick, bold line. Hence, the region to the right of the transition point is allowed in the window.

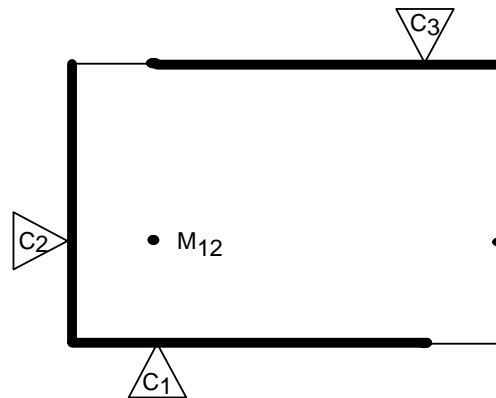


**Figure 4.10 – The nesting force window on the bottom surface. The bolded portion of the line is the unacceptable region.**

The same procedure can be done for the other two transition points. For the top surface, the initial guess of  $x_{n1} = 5.0$  units led to a negative reaction force. Therefore, all points to the right of the transition point along the top surface are not allowed. When  $y_{n2}$  was 3.5 units, the reaction forces were all positive. Thus, everything above the transition point on the right surface is allowed in the nesting force window.

**STEP 3: IF DESIRED, DRAW THE NESTING FORCE WINDOW ON A SKETCH OF THE ASSEMBLY**

It is often easier to visualize the nesting force window if it is drawn on a sketch of the assembly. Figure 4.11 shows the block assembly with the nesting force window drawn. The thick, bolded lines represent the unacceptable locations for the nesting force. It is easily seen that the nesting force window found using the general quantitative method matches the window found by the graphical method outlined by Blanding [1999].



**Figure 4.11 – The nesting force window for the block assembly. The bolded portion is the unacceptable region of the window.**

Thus, the nesting force window can easily be found using the equations of equilibrium to locate transition points. This simple procedure will now be used as part of the generalized method to quantitatively analyze EC design.

#### ***4.4 GENERALIZED METHOD TO QUANTITATIVELY ANALYZE EC DESIGN***

Chapter 3 showed how the equations of equilibrium can be used to analyze an assembly from point-to-point. Now, all the information presented in this chapter will be brought together to show a general, more efficient method to quantitatively analyze EC designs.

There are two primary reasons to analyze any EC design: to inspect the design to make sure all criteria are met for an EC assembly, or to make predictions about the behavior of the design. First, a general method will be presented in Section 4.4.1 to inspect an EC design. A very similar method will be presented in Section 4.4.3 that predicts unwanted and avoidable behavior of an EC assembly for design purposes.

##### **4.4.1 GENERAL METHOD TO INSPECT EC DESIGN**

It is the purpose of this section to present a general method using the equations of equilibrium to verify that an assembly complies with the rules for EC design. It is a useful method when the designer simply wants to inspect or check an assembly. (Section 4.4.3 will show a general method that will predict when an assembly violates EC rules so unwanted configurations can be avoided.) Four steps define the procedure.

1. Find the reaction forces using the method outlined in Section 4.2.
2. Verify that the reaction forces meet EC design rules.
3. If necessary, determine the appropriate changes to bring the assembly into compliance.
4. Find the nesting force window to appropriately place the nesting force.

Each step will be explained. Figure 4.1 will continue to be used to illustrate how the method works.

### **STEP 1: FIND THE REACTION FORCES**

Section 4.2 shows how to find the reaction forces for an EC design when the equations of equilibrium are set up in matrix form. The matrix equation associated with the given assembly in Fig. 4.1 yields the following reaction forces.

$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -0.333 \\ 1 \\ 0.667 \end{bmatrix}$$

### **STEP 2: VERIFY THAT THE REACTION FORCES MEET EC DESIGN CRITERIA**

The next step is to verify that the reaction forces meet EC design criteria. There are two criteria in particular which must be monitored: (1) The magnitude(s) of the reaction forces must not approach infinity, and (2) The signs on the reaction forces must be positive (according to the definition presented earlier in this thesis).

Recall from Chapter 3 that reaction forces go to infinity when the rules of EC design are violated. If any of the reaction force magnitudes approach infinity (i.e. very large in comparison to the applied force), it is an indication that the design is close to violating the rules in Chapter 3. Therefore, in quantitative terms, this step means that the magnitudes of the reaction forces are not approaching infinity.

In Fig. 4.1, the reaction forces are on the same order of magnitude as the nesting forces. They are in no way approaching infinity. Therefore, the locations chosen for the constraints do not violate the rules for EC design.

Notice, however, that  $R_1$  is negative. Clearly, by the end of the analysis, the sign on  $R_1$  must be positive. However, the signs on the reaction forces are determined by the location of the nesting force(s). That discrepancy will be remedied in the fourth step when the proper nesting force window has been found.

### **STEP 3: DETERMINE THE APPROPRIATE CHANGES TO BRING THE ASSEMBLY INTO COMPLIANCE**

If any of the reaction forces are approaching infinity, it will be necessary to change the location (point of contact) of one or more of the constraints. To determine which one to move, choose the constraint associated with a reaction force that has a large magnitude. The equations of equilibrium are then re-solved. Continue to move the constraints until all magnitudes are acceptable.

If any of the reaction forces are negative, the nesting force must be moved into the acceptable region. To do so, the nesting force window is found. Then, the nesting force



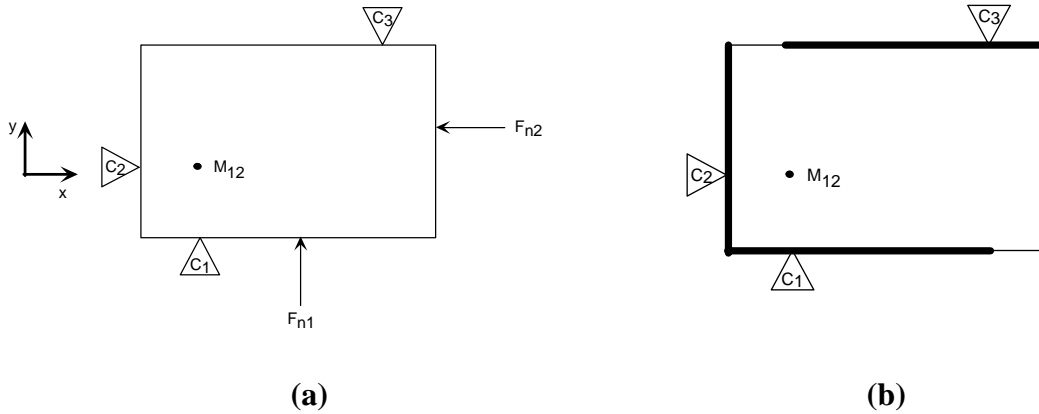
is moved into the acceptable region. The matrix equation is re-solved to ensure that the signs are correct on each reaction force.

#### **STEP 4: FIND THE NESTING FORCE WINDOW**

The next step is to find the nesting force window. The details to do this are found in Section 4.3.2, and they will not be repeated here.

Once the window has been found, the nesting force can be placed within the allowable region. The equations of equilibrium are again solved based on the new location for the nesting force. With the nesting force in an appropriate location, all criteria should be met for an EC design. The reaction forces should not be approaching infinity, and they must be positive.

Returning to Fig. 4.1 (shown below as Fig. 4.12a), this final step can be illustrated. Recall from step 2 that the magnitudes of the reaction forces were acceptable, but the directions were not. The wrong sign on  $R_1$  shows that the nesting force is not in the acceptable window. Also recall that the nesting force window was found earlier for this example (Section 4.3.2), and it is shown as Fig. 4.12b.

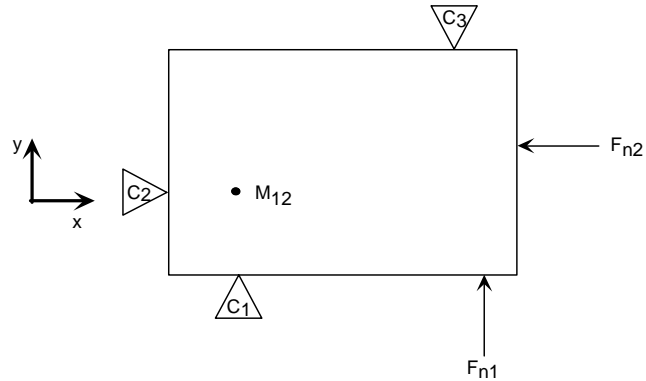


**Figure 4.12 – Placing the nesting force (a) Initial set-up for the block example, Fig. 4.1 (b) Nesting force window, Fig. 4.11**

Comparing the information in Fig. 4.12, the initial assignment for the location of  $F_{n1}$  is not in the allowable nesting force window. Moving the nesting force to where  $x_{n1}$  rests at 9.0 units changes the values for the reaction forces.

$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 0.33 \\ 1 \\ 1.33 \end{bmatrix}$$

All magnitudes and directions are now appropriate. Figure 4.13 shows the acceptable design.



**Figure 4.13 – Acceptable design for the block assembly**

This generalized method inspects an assembly to determine if it meets all quantitative criteria to be exactly constrained. It shows when changes must be made to find an acceptable design. The major benefit of using the generalized method shown here is that it is no longer necessary to check every point on the assembly to ensure that the design is exactly constrained.

In some instances, however, it may be desirable to use the equations of equilibrium to design for the locations of the constraints (and not just inspect them). The process shown above can be slightly modified to accommodate this desire.

However, before the method is presented to show how to predict and design an assembly based on the equations of equilibrium, it will be important to more fully understand when and why EC rules are violated from a mathematical perspective. Therefore, a brief discussion about when the  $C$  matrix becomes singular will be followed by the presentation of a generalized method to predict and design for EC assemblies.

## 4.4.2 SINGULARITY OF THE [C] MATRIX

When reaction forces have gone to infinity, it is because the **C** matrix is singular.

In turn, the singular **C** matrix yields a determinant of zero.

Any matrix becomes singular due to one of three reasons. When the reaction forces go to infinity, any of these three reasons could be the contributor. In addition, at the point where the **C** matrix becomes singular, the EC assembly has become over and/or under constrained, as can be seen in the matrix. In order to learn what caused the singular results in the **C** matrix, each one of these cases must be checked.

1. A row is zero
2. A column is zero
3. Linear dependence

### 4.4.2.1 A ROW OF ZEROS

Each row in the **C** matrix represents either the sum of a moment or the sum of the forces in a particular direction. When a row in the **C** matrix has gone to zero, it mathematically shows that there is no resistance to motion in that direction, and the assembly is *under-constrained*.

Consider the example when two constraints become co-linear, as was discussed in Section 3.3.1. The **C** matrix for such a design shows that the sum of the moments row has gone to zero.

$$C = \begin{matrix} \sum M = 0 \\ \sum F_x = 0 \\ \sum F_y = 0 \end{matrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix}$$

There is no resistive couple in the assembly, and rotation will result because of the applied nesting force (i.e. the right hand side has not gone to zero). In physical terms, this will be recognized as rotation in the assembly. The **C** matrix shows the under-constraint in rotation for this assembly.

#### 4.4.2.2 A COLUMN OF ZEROS

Each column in the **C** matrix contains coefficients from the equations of equilibrium relating to the individual constraints. Physically, the terms in each column show the components of direction for each constraint with respect to the main part.

In the moment equation, the coefficients show the moment arm. In the force equations, the coefficients give the angle of each force in each respective direction. For example, assuming that the resultant magnitude of some force is one unit, a coefficient of one shows that the force is parallel to the direction being summed. A coefficient of zero shows the force has no influence in the direction being summed.

If one column has gone to zero it means that one of the constraints is no longer in contact with the part. The assembly is again *under-constrained*. Whichever column has gone to zero will tell the designer which constraint is no longer in contact with the part, and the constraint can be modified accordingly.

### 4.4.2.3 LINEAR DEPENDENCE

In order to ensure that the assembly is exactly constrained, the equations of equilibrium must be a system of linearly independent equations. If there is linear dependence in the columns or rows, the assembly is no longer exactly constrained.

Linear dependence really means that one equation or constraint is a scalar multiple of another. Thus, when columns are linearly dependent, two or more constraints are competing to constrain the same degree of freedom. Linear dependence shows that the assembly is *over-constrained* in one direction.

Using the example when two constraints become co-linear from Section 3.3.1, the first column and the third column are linearly dependent by a factor of -1.

$$C = \begin{matrix} & R_1 & R_2 & R_3 \\ \sum M = 0 & \left[ \begin{array}{ccc} 0 & 0 & 0 \end{array} \right] \\ \sum F_x = 0 & \left[ \begin{array}{ccc} 0 & 1 & 0 \end{array} \right] \\ \sum F_y = 0 & \left[ \begin{array}{ccc} 1 & 0 & -1 \end{array} \right] \end{matrix}$$

The **C** matrix also shows *which* two constraints are competing. In this example, constraints 1 and 3 are competing to constrain the y-direction, and thus the linear dependence shows that the assembly is over-constrained in the y-direction.

Notice that in this example, the matrix demonstrates two reasons for the singularity. As mentioned, this matrix is linearly dependent (showing over-constraint in

the y-direction). However, it also reveals the moment equation as a row of zeros (showing under-constraint in rotation).

#### **4.4.3 GENERAL METHOD TO DESIGN AND MAKE PREDICTIONS FOR EC ASSEMBLIES**

Now that a more thorough explanation has been given concerning why the **C** matrix may become singular, the generalized method can help design for the locations of constraints in an EC design. This method determines the locations of the constraints where the assembly will no longer be exactly constrained, and those locations are avoided.

The general method to predict and design the location of the constraints is very similar to the method presented to inspect an EC design. Many steps are the same, and they are both based on quantitative principles instead of heuristics. However, in the method for design or prediction, the locations (contact points) where constraints violate EC rules are found. The steps are given below.

1. Find the reaction forces using the method outlined in Section 4.2.
2. Find the constraint coordinates, such that the **C** matrix becomes singular.
3. Find an acceptable location for the constraints, taking note to avoid those locations found where the **C** matrix goes singular.
4. Find the nesting force window to appropriately place the nesting force.

### **STEP 1: FIND THE REACTION FORCES**

Section 4.2 defined the process to find the reaction forces using the equations of equilibrium. Recall that the matrix equation associated with Fig. 4.1 was also presented in step 1 of the general method to inspect EC design.

$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -0.333 \\ 1 \\ 0.667 \end{bmatrix}$$

### **STEP 2: FIND THE CONSTRAINT LOCATIONS TO AVOID**

Just as the transition points can easily be found using a GRG algorithm, the locations for the constraints that violate the rules for EC design can be found using a similar process. This time, however, instead of setting each reaction force equal to zero, each force will be maximized.

To find the desired point of contact using the GRG algorithm, the design variables and functions must be defined. The design variable(s) will be the x and/or y-coordinates for the constraint under surveillance. One optimization constraint will define the relationship between the x and y-coordinates of the constraint, if one exists. Another optimization constraint requires that the reaction forces must again be greater than or equal to zero. The objective function will be to maximize the desired reaction force. In connection with this objective, another optimization constraint could set an upper limit on the reaction force.



This process will need to be repeated for each set of matrix equations developed. There will be one set of matrix equations per surface under consideration.

Now the unacceptable contact points for  $C_3$  will be found for Fig. 4.1. First,  $R_1$  will be set as the objective function (target cell in Microsoft Excel's Solver<sup>®</sup>) to be maximized. The design variable (changing cell) will be  $x_3$ . After solving the routine, the reaction forces for constraints 1 and 3 go to infinity.

$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -\infty \\ 1 \\ -\infty \end{bmatrix} \text{ when } x_3 = 2.0 \text{ units}$$

This result shows that when constraints 1 and 3 line up, the system is no longer exactly constrained. It agrees with all the quantitative and heuristic examples already presented.

The  $C$  matrix shows what has happened so that this assembly is no longer exactly constrained. It is under-constrained in rotation, as shown by the moment equation going to zero, and it is over-constrained in the y-direction as shown by the linearly dependent columns for  $R_1$  and  $R_3$ .

$$C = \begin{matrix} & R_1 & R_2 & R_3 \\ \sum M = 0 & \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \\ \sum F_x = 0 & \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \\ \sum F_y = 0 & \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \end{matrix}$$

Because the forces must balance,  $R_2$  will never reach infinity. Thus, for this example, the design will remain exactly constrained as long as constraints 1 and 3 do not become co-linear.

### **STEP 3: FIND AN ACCEPTABLE LOCATION FOR THE CONSTRAINTS**

This step takes the information found in Step 2 to make an informed decision on where to place the constraints. The points found in that step should be avoided, while still maintaining a good balance between the nesting forces and the reaction forces. In other words, the magnitudes for the reaction forces are checked at this stage to ensure that they are all on the same order of magnitude. High reaction forces signal problems, as discussed earlier. This step is similar to Steps 2 and 3 of the inspection method.

For the design in Fig. 4.1, the current placement of the constraints is satisfactory. The magnitudes are all about the same order of magnitude. No one force is controlling the assembly. Now, the directions must be corrected, and this is done through the placement of the nesting force.

### **STEP 4: FIND THE NESTING FORCE WINDOW**

The details to find the nesting force window are outlined in Section 4.3.2. This step is also found in Step 4 of the inspection method, where the current example has also been presented and solved. Figure 4.13 shows the final design based on the nesting force window.

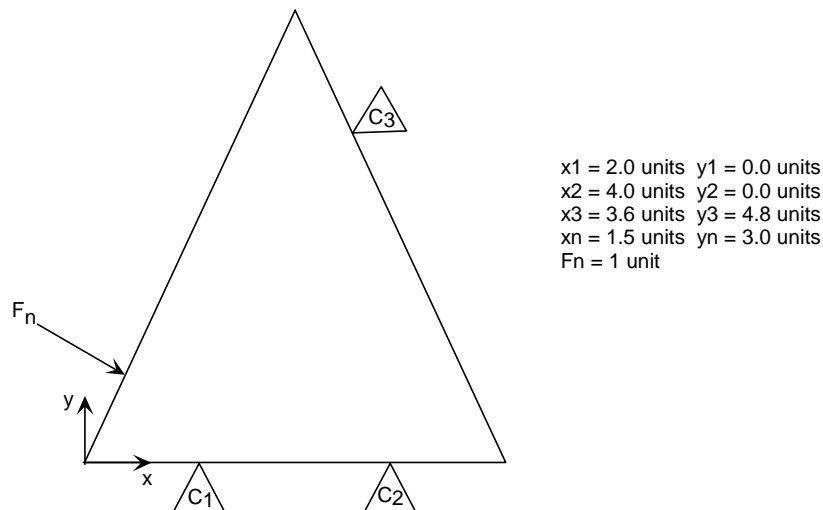
Figure 4.1 is an extremely oversimplified example, which has been used to illustrate the methods in this chapter. However, it easily shows the benefits and strengths of using a quantitative approach to analyze EC design. Additional examples will now be presented to show how easy these methods can be implemented.

#### 4.4.4 MORE SIMPLE EXAMPLES

Several very simple examples can be used to help further illustrate the methods outlined in this chapter. The first example will illustrate the method for inspection, while the second example will use the method for prediction.

##### *EXAMPLE 1: INSPECTION METHOD*

Consider the triangle shown in Fig. 4.14. The coordinates for all the constraints and the initial placement of the nesting force along the left surface are listed.



**Figure 4.14 – Triangular assembly for the inspection method**

This triangle is very similar to that shown in Fig. 3.12, except now there are two constraints along the bottom, none on the left side, and one constraint on the right side. While a visual inspection of this assembly shows that the nesting force cannot be placed along the right side or the base of the assembly, all the matrix equations will be analyzed to show what happens.

The first step is to set up the equations of equilibrium in matrix form. The equations obviously differ depending on where the nesting force rests. Therefore, three cases will need to be examined: the nesting force will move along the left surface (denoted “left”), it will continue its path along the right surface (denoted “right”), and it finishes its path along the base of the assembly (denoted “base”).

$$\begin{bmatrix} 0 & (x_2 - x_{13}) & 0 \\ 1 & 1 & -\sin(\theta) \\ 0 & 0 & -\cos(\theta) \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -F_n \cos(\theta) * (y_{13} - y_n) - F_n \sin(\theta) * (x_{13} - x_n) \\ F_n \sin(\theta) \\ -F_n \cos(\theta) \end{bmatrix}$$

*left*

$$\begin{bmatrix} 0 & (x_2 - x_{13}) & 0 \\ 1 & 1 & -\sin(\theta) \\ 0 & 0 & -\cos(\theta) \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -F_n \cos(\theta) * (y_n - y_{13}) + F_n \sin(\theta) * (x_n - x_{13}) \\ F_n \sin(\theta) \\ F_n \end{bmatrix}$$

*right*

$$\begin{bmatrix} 0 & (x_2 - x_{13}) & 0 \\ 1 & 1 & -\sin(\theta) \\ 0 & 0 & -\cos(\theta) \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -F_n (x_n - x_{13}) \\ -F_n \\ 0 \end{bmatrix}$$

*base*

Applying the initial values shown in Fig. 4.14 to the matrices given above produces the following systems of equations. The initial point of contact along each surface for the nesting force is also listed.

$$\begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & -0.5 \\ 0 & 0 & -0.866 \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -1.116 \\ 0.5 \\ -0.866 \end{bmatrix}$$

*left*

$x_n = 1.5 \text{ units} \quad y_n = 3.0 \text{ units}$

$$\begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & -0.5 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 2.93 \\ 0.5 \\ 1 \end{bmatrix}$$

*right*

$x_n = 3.828 \text{ units} \quad y_n = 4.344 \text{ units}$

$$\begin{bmatrix} 0 & 2 & 0 \\ 1 & 1 & -0.5 \\ 0 & 0 & -0.866 \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -2 \\ -1 \\ 0 \end{bmatrix}$$

*base*

$x_n = 1.5 \text{ units} \quad y_n = 0.0 \text{ units}$

The next step is to solve the equation  $\mathbf{C}\mathbf{r} = \mathbf{b}$  to find the reaction forces. The vectors below show the values for the reaction forces.

$$\begin{array}{ccc} \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 1.56 \\ -0.558 \\ 1 \end{bmatrix} & \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -0.308 \\ 0.308 \\ -1 \end{bmatrix} & \begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} -1.25 \\ 0.25 \\ 0 \end{bmatrix} \\ \textit{left} & \textit{right} & \textit{base} \end{array}$$

Now, it is necessary to make certain that the reaction forces are not approaching infinity. A quick glance shows that indeed this assembly continues to be exactly constrained. In addition, all the forces are about the same order of magnitude, so no one force is carrying a disproportionate amount of the load. Note that when the nesting force has been placed along the base of the block,  $R_3$  falls out. Only two constraints are carrying the load. Because of this, the nesting force cannot be placed along the base.

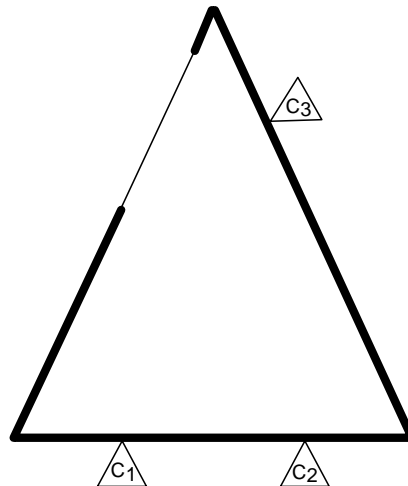
To fix the signs on all the reaction forces, the nesting force must be moved into the nesting force window. All of the negative signs show that the nesting force is not in the window.

The nesting force window is created by first finding the transition points. In Microsoft Excel's Solver<sup>®</sup>,  $R_1$  is chosen to go to zero, and the transition point is found. Then,  $R_2$  is forced to zero followed by  $R_3$  forced to zero. The process is repeated for each surface. The resulting transition points are found in Table 4.1.

**Table 4.1 – Transition points along each side of the assembly**

Side of the assembly	Transition point	Value of Reaction Forces
Left	$x_n = 2.896$ units $y_n = 5.792$ units	$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$
Left	$x_n = 2.0$ units $y_n = 4.0$ units	$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$
Right	No point exists	N/A
Bottom	No point exists	N/A

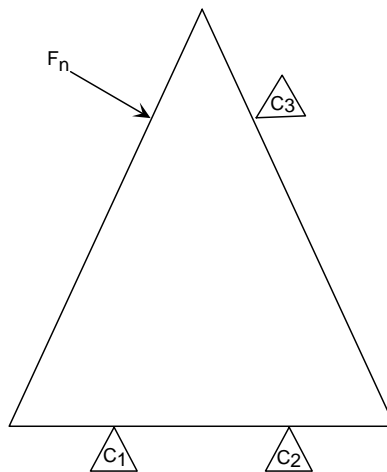
Toggling the x-coordinate a little above and below each transition point shows which portion of the segment is allowed in the window and which portion is not allowed. From this information, a sketch of the nesting force window is drawn (Fig. 4.15) to get a better view of the physical dimensions. The nesting force window is comprised of a very small portion of the overall assembly.



**Figure 4.15 – Nesting force window for the triangle assembly. The bolded lines are the points where the nesting force is not allowed.**

The nesting force is moved into the allowable region at  $x_{n1} = 2.5$  units, and the reaction forces are re-calculated. The method outlined in this chapter quickly found that Fig. 4.16 is an acceptable EC design.

$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 0.441 \\ 0.558 \\ 1 \end{bmatrix}$$



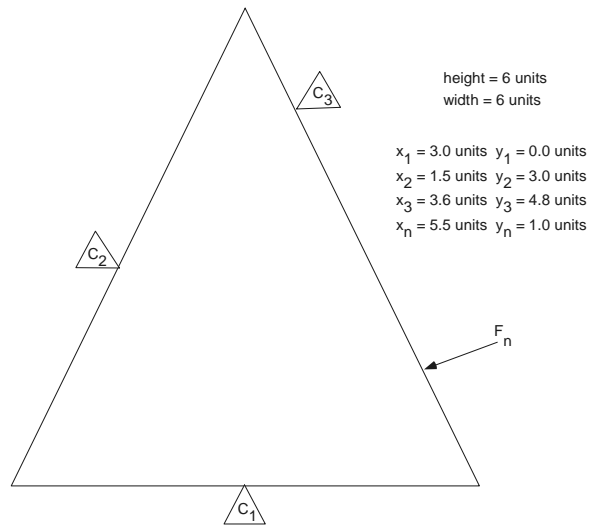
**Figure 4.16 – Acceptable EC design for the triangle assembly**

This short example showed how the equations of equilibrium can inspect an assembly to ensure it is exactly constrained. In the current set-up, unless constraints 1 and 2 were placed at the same point, it would be impossible for any two of the three constraints to become co-linear or for all three constraint lines to intersect at a point because of the two parallel constraints along the bottom.



### EXAMPLE 2: METHOD FOR PREDICTION

Now, constraint 2 will be moved over to the left side of the triangular part, as in Fig. 3.12 (and shown in Fig. 4.17). This time, the method to predict bad configurations will be used to analyze the part.

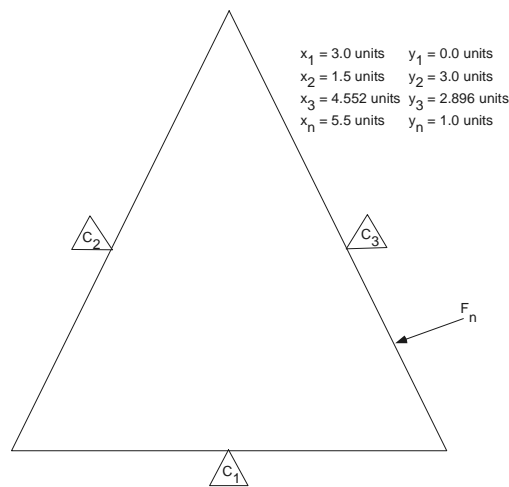


**Figure 4.17 – Triangular assembly for the design method**

First, the matrix equations must be formulated and solved to find the reaction forces. The actual equations can be found in Appendix C. For brevity, only the analysis will be shown for when the nesting force is along the right surface of the assembly. The reaction forces are given below.

$$\begin{bmatrix} R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} 1.996 \\ 1.996 \\ 0.996 \end{bmatrix}$$

The next step is to find the configuration(s) of the constraints that violate the EC design rules of Chapter 3. This step is simply accomplished through Excel's Solver<sup>®</sup>:  $x_3$  is the design variable;  $y_3 = -2 * x_3 + 12$  (the equation of the right surface);  $R_1 \geq 0$ ,  $R_2 \geq 0$ , and  $R_3 \geq 0$  are the optimization constraints; and the objective function is to maximize  $R_1$ ,  $R_2$ , and  $R_3$ , respectively. In this example, they all go to infinity for the same configuration of the constraints. Figure 4.18 shows the only configuration for this assembly that fails to be exactly constrained. This configuration violates the rule validated earlier that no three constraint lines should intersect at a point.



**Figure 4.18 – Configuration that makes the assembly no longer exactly constrained**

This configuration is under-constrained in rotation, as shown in the **C** matrix.

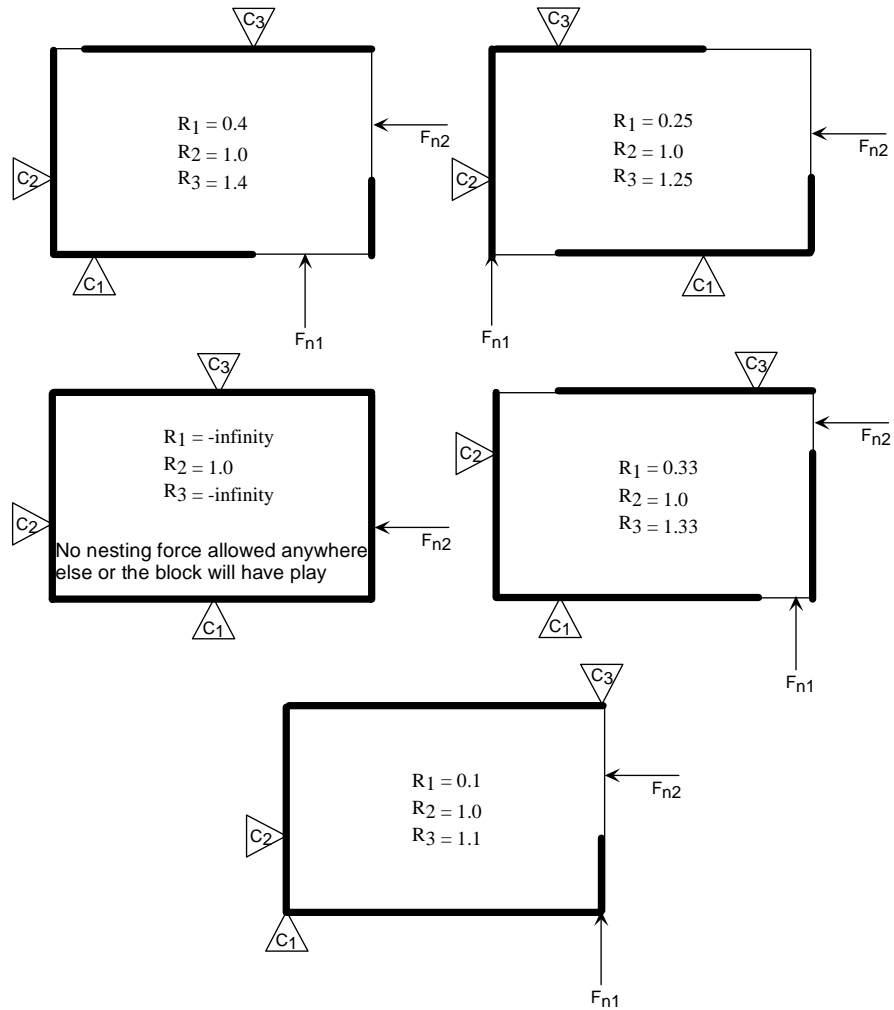
$$C = \begin{matrix} \sum M = 0 \\ \sum F_x = 0 \\ \sum F_y = 0 \end{matrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.866 & -0.866 \\ 1 & -0.5 & -0.5 \end{bmatrix}$$

The under-constraint means that no resistive moment exists to overcome the moment that will be caused by the application of the nesting force.

Keeping this information in mind, a position now needs to be found for the constraints. The original set-up provided a good representation for the reaction forces. None of the forces carried a disproportional amount of the load, and they were all positive. The positive reaction forces show that the nesting force is also in a good place (which was known from Chapter 3). Therefore, Fig. 4.17 represents a good configuration for this assembly.

#### ***4.5 TRADEOFF BETWEEN THE REACTION FORCES AND THE NESTING FORCE WINDOW***

Each analysis using the generalized method is based on a given placement of the constraints. Consider Fig. 4.19, which shows Fig. 4.1 in several different configurations. The constraints, reaction forces, and nesting force windows are all shown. The detailed force analysis for each design may be found in Appendix D.



**Figure 4.19 – Various configurations of the same block assembly**

Notice that the nesting force window changes according to the locations of the constraints. There is a tradeoff between the location of the constraints and the size of the nesting force window. While the tradeoff is not always negative, in this example, the further apart the constraints are to each other, the smaller the nesting force window. The closer the constraints are to each other, the larger the nesting force window, but the greater the reaction forces climb until the design becomes unstable.

Recall from Chapter 2 that one of the benefits of EC design is the allowance for greater variation in the location of the constraints. While Fig. 4.19 again validates this benefit, it can also be seen that there are some EC designs that may be better than others. For example, one design may have lower reaction forces, while another design may provide a more flexible nesting force window.

There may be additional design considerations that help decide where the constraints should be placed.

1. Space limitations—it often happens that due to the number of parts or due to the locations of the parts within an assembly, space is limited.
2. Geometry—the geometry of the part often limits the possible locations for the constraints or joints in an assembly.
3. Required function of the assembly—the constraints cannot interfere with the function of the assembly, and this stipulation may limit the possible locations for the constraints.
4. Reaction forces exerted on the assembly—high reaction forces should be avoided. In addition, balancing the reaction forces will not place undue stress on the assembly.

However, sometimes the designer may simply locate constraints based on personal preference. In that case, it would be beneficial to have a quantitative means available to find a factor of “goodness” for each EC design to help make the decision.

The equations of equilibrium show the results of the reaction forces, but this is not the only mathematical criterion that may be useful for EC design.

Chapter 5 will explore an additional quantitative method to find the “goodness” of an EC design. This method could be used to determine which design might provide the most beneficial tradeoff for given conditions.



## CHAPTER 5 A QUANTITATIVE MEASURE OF “GOODNESS” IN AN EXACTLY CONSTRAINED DESIGN

### 5.1 INTRODUCTION

Chapter 4 introduced a generalized method using the equations of equilibrium to quantitatively analyze exactly constrained (EC) designs. Using this generalized method, a designer can either ensure EC design compliance in an assembly, or he can predict configurations where an assembly would no longer qualify as exactly constrained.

Another contribution of this generalized method allows for constraint analysis when the  $C$  matrix becomes singular. Investigating where the equations of equilibrium fail will, at least in part, show why the design no longer meets the criteria for EC design.

Chapter 4 also illustrated a natural tradeoff involving the constraints and the allowable range for the nesting force window. The simple example in Fig. 4.19 showed one example of this tradeoff. Configurations with more distance between constraints 1 and 3 had lower reaction forces, but they also had smaller windows for the nesting force. However, moving the constraints closer together in the  $x$ -direction resulted in higher reaction forces with an expanded nesting force window. The reaction forces continued to increase until the two constraints in question became co-linear, at which point the



assembly became over-constrained in the y-direction, under-constrained in rotation, and the window disappeared for the vertical nesting force.

The discovery of this tradeoff now leads to an investigation into the goodness of an EC design. “Goodness” refers to the fact that while many different configurations of an assembly may constitute an acceptable EC design, there may be some configurations that fulfill design needs better than others.

Just as the general principles of EC design were strengthened by using the equations of equilibrium, the goodness criteria must also be founded on quantitative, mathematical principles. Chapter 5 investigates an approach using quantitative methods to define the goodness in an EC design.

A brief discussion defining the qualitative goodness of an EC design is followed by a brief review of constraint analysis using screw theory. It will show an additional need for a method that will provide a quantitative measure of goodness. Finally, using information from the direct linearization method (DLM), a quantitative method to determine goodness for an EC design will be presented.

## ***5.2 THE GOODNESS OF AN EXACTLY CONSTRAINED DESIGN***

One of the major benefits of EC design is the robust ability of a design to assemble even when variation may enter into the assembly components. This benefit will be further illustrated in Chapter 6 when variation is introduced into assemblies. However,

while the part may assemble under a wide variety of conditions, Chapter 4 illustrated that some configurations may still have certain advantages over others.

The idea that some configurations may pose greater benefits than others will be referred to as the “goodness” of EC design. Goodness, as referred to here, is defined by several aspects.

1. The EC assembly is not on the verge of becoming over-constrained or under-constrained.
2. Variation or tolerances of the parts have little to no effect on the ability of the design to assemble.
3. The assembly offers an acceptable trade-off between the size of the nesting force window and the distance between constraints in order to minimize the magnitudes of the reaction forces.
4. All possible advantages of EC design are utilized and preserved.
5. The overall assembly error is at a minimum.

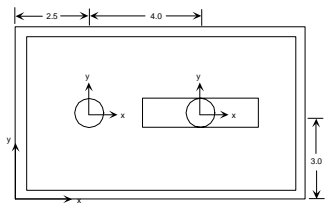
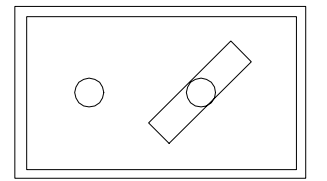
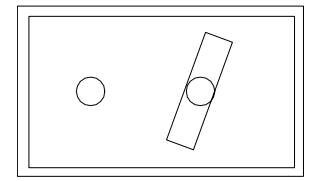
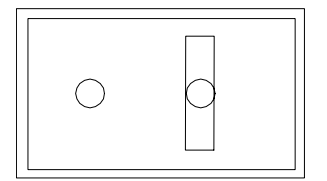
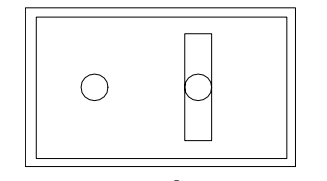
The qualitative goodness criteria defined above for an EC design is not enough to determine which designs would rank higher or better. A mathematical, quantitative method will add strength to any decision. Thus, mathematical techniques will be investigated in this chapter as a way to quantitatively measure the goodness of a design.

### **5.3 USING SCREW THEORY TO QUANTITATIVELY MEASURE GOODNESS**

Recall from Chapter 2 that the screw theory approach to constraint analysis presented by Adams [1998] pinpointed where any over and under-constraint exists in an assembly. One of the criteria defined for goodness of an EC design was that the assembly was not on the verge of becoming over or under-constrained. What happens in screw theory when an assembly transitions from an EC design to a non-EC design?

In Chapter 2, the slotted block example was used to show how screw theory performs. When the slot was at an angle of  $0^\circ$  to the block, screw theory found that the assembly was exactly constrained.

Figure 5.1 shows the final results for the slotted block as the angle of the slot changes. The detailed calculations for Fig. 5.1 can be found in Appendix A. As the angle of the slot changes, the assembly transitions from an EC design to one of over-constraint in the x-direction and under-constraint in rotation. More particularly, this analysis with screw theory shows that up to and through  $89.9^\circ$ , this assembly is exactly constrained. However, when the slot is rotated to  $90^\circ$ , the assembly suddenly becomes over-constrained in the x-direction and under-constrained in rotation.

Angle of the slot	Motion analysis: Under-constrained	Force Analysis: Over-constrained	EC
 <p style="text-align: center;"><math>0^\circ</math></p>	No: empty matrix	<p style="text-align: center;">No</p> $\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$	Yes
 <p style="text-align: center;"><math>45^\circ</math></p>	No: empty matrix	<p style="text-align: center;">No</p> $\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$	Yes
 <p style="text-align: center;"><math>70^\circ</math></p>	No: empty matrix	<p style="text-align: center;">No</p> $\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$	Yes
 <p style="text-align: center;"><math>89.9^\circ</math></p>	No: empty matrix	<p style="text-align: center;">No</p> $\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$	Yes
 <p style="text-align: center;"><math>90^\circ</math></p>	<p>Yes: rotation about the point (2.5, 3)</p> $\begin{bmatrix} 0 & 0 & 1 & 3 & -2.5 & 0 \end{bmatrix}$	<p>Yes: x translation along the axis <math>y = 3.0</math></p> $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & -3 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$	No

**Figure 5.1 – Results of the Screw Theory Analysis. EC means that the design is exactly constrained.**

Screw theory identifies the exact point at which the assembly becomes over and under-constrained. However, it gives no indication that the assembly is approaching such a state. In other words, screw theory acts like a “switch.” It is either “on” or “off.” Either the assembly is exactly constrained, or it is not.

While this method provides a powerful way to perform constraint analysis for a given arrangement in an assembly, screw theory does not appear to provide any indication when the assembly is approaching an over and/or under-constrained state. It does nothing to illustrate a tradeoff between design considerations for an EC design, and it cannot show the effects of variation. It thus shows no difference or superiority between EC designs with differing slot angles.

#### ***5.4 USING THE DIRECT LINEARIZATION METHOD (DLM) TO PROVIDE A QUANTITATIVE GOODNESS CRITERIA***

The direct linearization method (DLM) was developed for tolerance analysis. While it is not traditionally applied to constraint analysis, the work of Daniel Smith [2001] found a link between it and screw theory.

Smith found that the 3D representation of the **B** matrix (a matrix of first order partial derivatives with respect to the dependent variables) from the DLM could be used in place of the initial twist matrices for the under-constrained (motion) analysis in screw theory. The **F** matrix (a matrix of first order partial derivatives with respect to the geometric feature variables—see Chase et. al [1996]) from the DLM could be used in place of the initial twist matrices for the over-constrained (force) analysis. Then, the

same reciprocal operations and steps were applied as in screw theory to find the overall constraint status. Smith called his method the Variation-based Constraint Analysis of Assemblies (VCAA).

Screw theory as applied to constraint analysis and the VCAA show when and where a design is or is not exactly constrained; however, neither of the methods provides a quantitative measure of goodness that signals when the design may be approaching an over or under-constrained state. The measure for goodness must do more than state if the assembly is exactly constrained or not. It must show when a design is close to violating the rules from Chapter 3. It must also provide a means to compare various configurations of the same assembly to determine which configuration may best suit the functional needs of the assembly.

The DLM provides additional information not utilized by other methods that can provide a quantitative measure of goodness. As will be presented in this chapter, the determinant of the  $\mathbf{B}$  matrix can signify when a design is approaching an unstable state. Also, the assembly sensitivities,  $\mathbf{B}^{-1}\mathbf{A}$ , provide a way to quantitatively compare similar EC designs to find which configurations are least affected by variation.

A more in-depth discussion of the variables and the partial derivatives (which are found when the vector loop equations are linearized) must first be discussed to understand why the two matrices of sensitivities work well as a measure for goodness. Then, a discussion of the  $\mathbf{B}$  matrix and its contributions to EC design goodness will be

followed by a discussion on using the assembly sensitivities to find an additional quantitative value for goodness.

#### **5.4.1 SIGNIFICANCE OF PARTIAL DERIVATIVES IN THE DLM**

Before proceeding to find a quantitative measure of goodness for EC design, a more comprehensive look at the variables and the partial derivatives will be presented. This information is explored to give a solid background to the next section, **B** matrix contributions.

Each vector in a vector loop equation is composed of two or more variables, which describe the magnitude and direction of that vector. There are two types of variables: the *length* variable describes the magnitude of a vector, while the *angular* variable relates to the direction of the vector in the assembly. These variables could be either independent or dependent in the vector loop analysis.

In the DLM, the independent variables are the known (or user-defined) dimensions in the analysis. For example, the locations of the constraints or the height and width of parts are independent *length* variables; the angles of surfaces or slots are independent *angular* variables. The independent length variables will be collectively denoted as  $x_i$ , while the independent angular variables will be collectively denoted as  $\phi_i$ .

The dependent variables are the *resulting* dimensions based on the values of the independent variables. They are used in the analysis to absorb changes from the

independent variables to allow the vector loop equations to close. The dependent length variables will be denoted as  $u_i$ , and the dependent angular variables will be denoted as  $\theta_i$ . Vectors that include dependent length variables always go from an unknown position of a joint (where a constraint connects to the main body of the assembly) to a known position in the assembly. These vectors can change in magnitude and direction, according to the variation present in the assembly.

The vectors in the vector loops (Section 2.9.4) are summed together in the vector loop equations. For example, the vector loop equation  $h_x$  sums together the component of each vector in the x-direction. The magnitude of the vector is a length variable, and the direction involves the cosine of the angular variables. Likewise, the  $h_y$  equation sums together the components of each vector in the y-direction. The  $h_\theta$  equation sums together the angles of each vector relative to one another.

While it is possible to find the values of the dependent variables using a non-linear equation solver, the DLM is more interested in quantifying the effects of small changes (variation) in the assembly. As mentioned in Section 2.9.6, the vector loop equations can be linearized, based on a first order Taylor series expansion. The resulting equation takes the form,

$$[A]\{\delta X\} + [B]\{\delta U\} = \{0\} \quad (1)$$



**A** contains the partial derivatives with respect to the independent variables. **B** contains the partial derivatives with respect to the dependent variables.

Variables, vectors, and vector loop equations were presented here because they influence the first order partial derivatives. Information from the partial derivatives with respect to each kind of variable (length or angular) will be applied later in this chapter when the **B** matrix contributions are presented. However, the specific significance of the partial derivatives with respect to the each kind of variable will first be explained.

Taking the partial derivative with respect to a length variable will leave the component of a unit vector in a certain direction. For example, taking the partial derivative of the  $h_x$  equation with respect to a length variable will leave the component of a unit vector in the x-direction, as illustrated in the example below (which only shows the dependent variables from a vector loop equation).

$$h_x = \dots + u_1 \cos(180 + \theta) + u_2 \cos(90 + \theta)$$

$$\frac{\partial h_x}{\partial u_1} = \cos(180 + \theta)$$

$$\frac{\partial h_x}{\partial u_2} = \cos(90 + \theta)$$

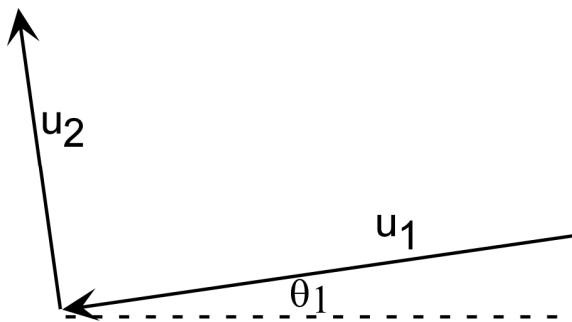
To further illustrate, suppose a term representing the partial derivative of  $h_x$  with respect to some length variable in the **B** matrix gives a value of 0.707.

$$B = \begin{matrix} \partial u_1 & \partial u_2 \\ \left[ \begin{array}{cc} 0.707 & 1 \\ 0.707 & 1 \end{array} \right] \partial h_x \\ \left[ \begin{array}{cc} 0.707 & 1 \\ 0.707 & 1 \end{array} \right] \partial h_y \end{matrix}$$

By virtue of being in the x-direction, this value is related to the cosine of an angle.

Because the partial derivatives are all unit vectors for length variables, the angle of the vector is  $45^\circ$ . The value of the partial derivative taken with respect to a length variable will always be between 0 and 1, as it is the component of a unit vector in some direction.

However, the partial derivative for any angular variable will become the component of a *resultant* vector (not a unit vector) in the opposite direction from the vector loop equation under analysis. For example, Fig. 5.2 shows all vectors in a vector loop associated with the variable  $\theta_1$ .



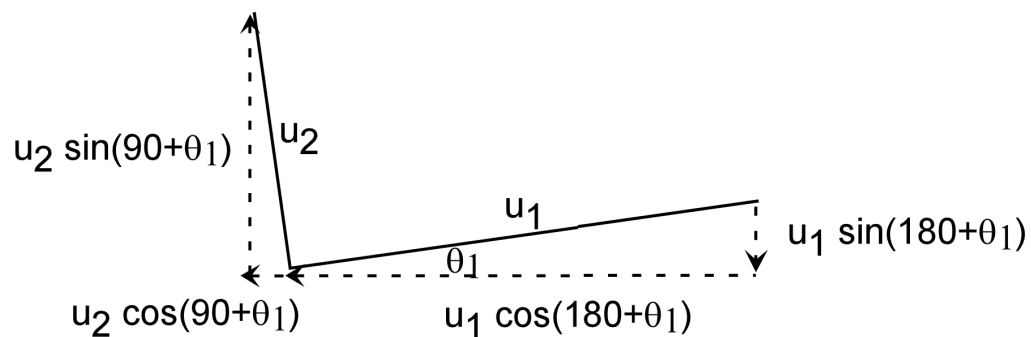
**Figure 5.2 – All vectors associated with  $\theta_1$  in a vector loop sample**

The tail of vector  $u_1$  is the position of one joint connecting a constraint to the main block of an assembly, and the tip of  $u_2$  is another joint connecting a different constraint to the same block. The portion of the  $h_x$  equation relating to these vectors is stated below,

followed by the partial derivative of the equation with respect to  $\theta_1$ . Figure 5.3 shows the components of each vector labeled.

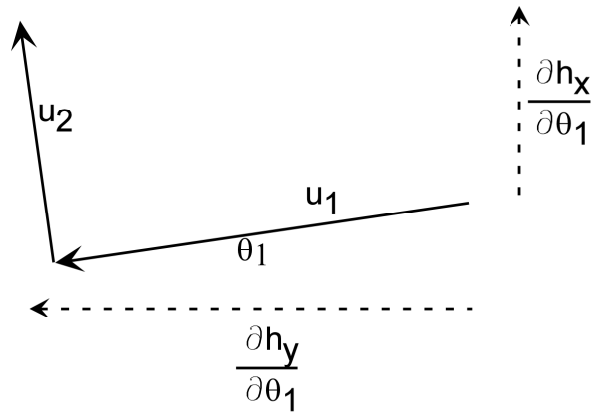
$$h_x = \dots + u_1 \cos(180 + \theta_1) + u_2 \cos(90 + \theta_1)$$

$$\frac{\partial h_x}{\partial \theta_1} = -u_1 \sin(180 + \theta_1) - u_2 \sin(90 + \theta_1)$$



**Figure 5.3 – Components of each vector for the example in Fig. 5.2**

Taking the derivative with respect to  $\theta_1$  shows that the partial derivative of the x-equation with respect to  $\theta_1$  will be the component of a resulting vector (of some non-unit magnitude) in the y-direction. Using the information in Fig. 5.3, it can be seen that the resultant vector for this example will be in the positive direction with a magnitude equal to the y-distance between the tail of  $u_1$  and the tip of  $u_2$  (as shown in Fig. 5.4).



**Figure 5.4 – Resultant vectors representing the values of  $\frac{\partial h_x}{\partial \theta_1}$  and  $\frac{\partial h_y}{\partial \theta_1}$**

In review, the partial derivatives found in the **A** and **B** matrices have meaning that will be used later in this chapter. The partial derivative in some direction with respect to a length variable will be the component of a unit vector in that same direction. The partial derivative in some direction with respect to an angular variable will be the component of a resultant vector in the opposite direction.

The **A** matrix will be momentarily set aside because it will not contribute any additional information until the assembly sensitivities are found. However, with the previous information presented, a deeper understanding of the **B** matrix contributions can begin to be used to define a measure of goodness for EC design. These contributions will also explain why the assembly sensitivities act as a useful indicator for the goodness criterion.

### 5.4.2 [B] MATRIX CONTRIBUTIONS

As previously mentioned, the actual meaning of each term in the **B** matrix depends on which kind of variable the derivative was taken with respect to. If the derivative is taken with respect to  $u_i$ , the value represents the component of a unit vector in a certain direction. Because it is only the direction of a unit vector, the values for the derivatives will always be from 0 to 1. If, however, the derivative is for  $\theta_i$ , the value represents the component of a *resultant* vector in the opposite direction of the row in which the term resides.

In order to find the assembly sensitivities, the **B** matrix must be inverted and multiplied by **A**. In order to invert the **B** matrix, it must be square and non-singular. If the matrix is square, the number of variables equals the number of equations.

Recall that singularity means the determinant of the matrix will go to zero. This singularity can happen due to any of three reasons.

1. A row is zero
2. A column is zero
3. Linear dependence

Just as with the **C** matrix for the equations of equilibrium, a careful examination of the three criteria for a singular **B** matrix in the DLM shows useful information relating to EC design constraint status. Following an explanation of what it means to an assembly when

the **B** matrix goes singular for each case listed, a section will present how to use the **B** matrix as a quantitative measure for goodness.

#### 5.4.2.1 A ROW OF ZEROS

Each row in the **B** matrix represents the partial derivatives in one direction with respect to each dependent variable. Because the dependent variables allow the vector loop equations to sum to zero by absorbing any change from the independent variables, a row of zeros in the **B** matrix shows that any variation of the independent variables will not allow the vector loops to close. The dependent variables can no longer absorb changes from the independent variables in the direction corresponding to the row that went to zero. The assembly is *over-constrained* in the direction corresponding to the row that went to zero. For example, the B matrix shown below illustrates an assembly is over-constrained in the x-direction.

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 1 & 4 \end{bmatrix} \begin{matrix} \partial h_x \\ \partial h_y \end{matrix}$$

To better understand what this means, equation (1) can be rearranged.

$$-[\mathbf{A}]\{\delta X\} = [\mathbf{B}]\{\delta U\} \quad (2)$$

For this equation to be true, each of the vector loop equations must equal zero. If any row in **B** is all zeros, the independent variables in that direction cannot have *any* variation or else the vector loops will not go to zero.



### 5.4.2.2 A COLUMN OF ZEROS

Each column in the **B** matrix represents the partial derivatives in each direction with respect to one dependent variable. When the derivative is taken for  $u_i$  in each of the vector loop equations, the column in the **B** matrix shows the components of a unit vector pointing in the x and y-directions. Because it only involves the components of a unit vector in some direction, a column for a  $u_i$  variable can never have all zeros. The square root of the sum of the squares for the x and y-components in the column will always be one.

A column can only be full of zeros for a  $\theta_i$  variable. Recall that the partial derivatives for  $\theta_i$  involve both a non-unit magnitude and the components of direction; in other words, each term in a column of the **B** matrix for  $\theta_i$  contains the components of a resultant vector in the x or y-directions. A column can only be full of zeros when two constraints that both relate to  $\theta_i$  are placed at the same point. There is no resultant vector between the constraints in this case. Thus, the assembly has become *under-constrained*. However, this is a trivial case because it is not likely to occur often.

### 5.4.2.3 LINEAR DEPENDENCE

Linear dependence will also cause the determinant of the **B** matrix to go to zero. If the **B** matrix does have linear dependence, the vector loop equations must be inspected to determine if they all summed to zero. If the vector loops did sum to zero, there is a constraint problem in the design. If the vector loops did not sum to zero, the design did not assemble. Each case is now explained in further detail.



If each of the vector loop equations sum to zero, the linear dependence signals a constraint problem in the design. The constraint problem, as seen through the **B** matrix, depends on whether the system is linearly dependent due to the columns or the rows.

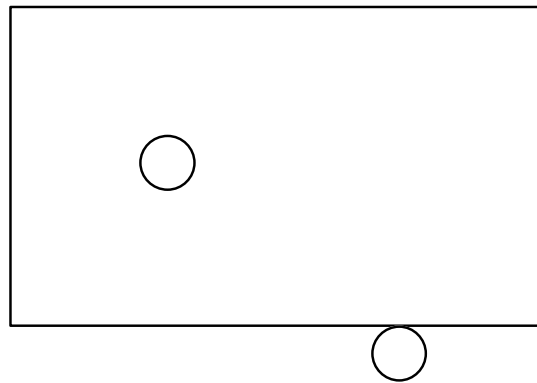
The system is *under-constrained* in rotation if the columns are linearly dependent. The linear dependence in this case will likely occur between a column representing the partial derivatives of a length ( $u_i$ ) variable with respect to each vector loop equation and a column representing the partial derivatives of an angular ( $\theta_i$ ) variable with respect to each vector loop equation (linear dependence between two length columns is a trivial case as it will not occur if there is more than one vector loop or if there is any resultant rotation of the part).

For example, the **B** matrix listed below (and used earlier) shows linear dependence in the columns.

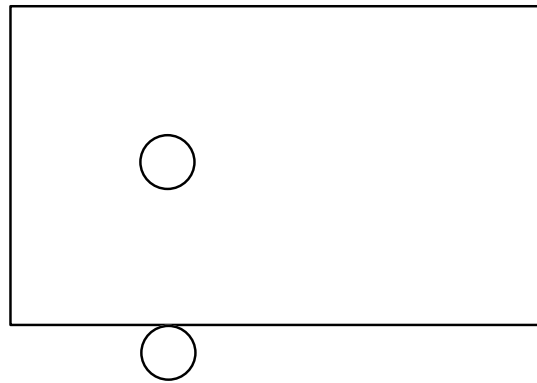
$$\mathbf{B} = \begin{matrix} \partial u_1 & \partial \theta_1 \\ \left[ \begin{array}{cc} 0 & 0 \\ 1 & 4 \end{array} \right] \end{matrix} \begin{matrix} \partial h_x \\ \partial h_y \end{matrix}$$

In this case, the columns show that this assembly will have under-constraint in rotation because the physical joint that provides the constraint associated with  $u_1$ , which once eliminated rotation in its nominal position, has moved in such a manner that it provides no constraint against rotation.

A physical example that would result in the  $\mathbf{B}$  matrix given above is found in Fig. 5.5. The revolute joint provides the translational constraints and the cylinder slider provides the rotational constraint in Fig. 5.5a. However, the slider joint has moved in Fig. 5.5b, and there is no longer constraint against rotation. Linear dependence between columns signals *under-constraint* in rotation.



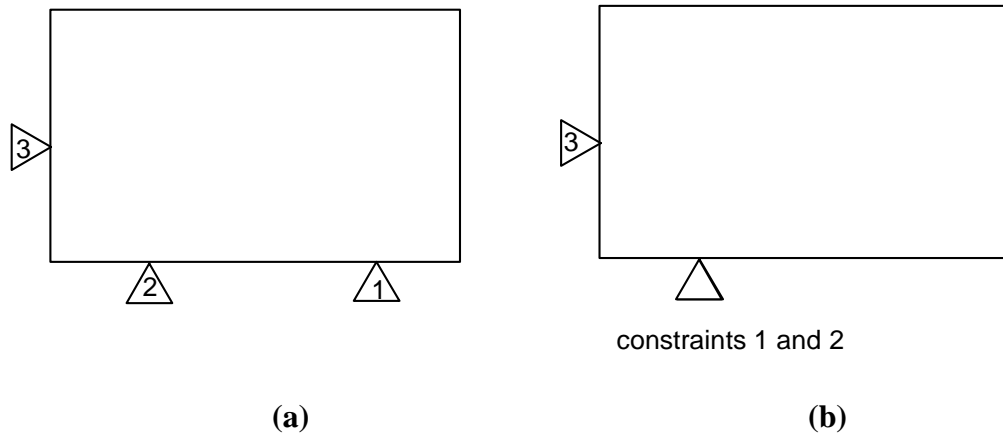
(a)



(b)

**Figure 5.5 – Example showing when the  $\mathbf{B}$  matrix would have linear dependence in the columns (a) EC design (b) Under-constrained in rotation according to the  $\mathbf{B}$  matrix**

If rows are linearly dependent in the  $\mathbf{B}$  matrix, the system is *over-constrained* in the direction relating to the dependent rows. The linear dependence signals that two constraints are competing to constrain the same direction, as shown by the vector loop equations. For example, Fig. 5.6a shows an exactly constrained design that becomes over-constrained (Fig. 5.6b) in the y-direction when the two constraints along the base are located at the same point. The  $\mathbf{B}$  matrix shows this over-constraint by the linear dependence in the second and fourth rows. Linear dependence between rows in the  $\mathbf{B}$  matrix signals *over-constraint*.



$$\mathbf{B} = \begin{matrix} & \begin{matrix} \partial u_1 & \partial u_2 & \partial u_3 & \partial \theta_1 \end{matrix} \\ \begin{bmatrix} -1 & 0 & 0 & -2.5 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & -1 & -2.5 \\ 0 & 1 & 0 & -2 \end{bmatrix} & \begin{matrix} \partial h_{x1} \\ \partial h_{y1} \\ \partial h_{x2} \\ \partial h_{y2} \end{matrix} \end{matrix}$$

**Figure 5.6 – Example showing when the  $\mathbf{B}$  matrix would have linear dependence in the rows (a) EC design (b) Over-constrained in the y-direction**

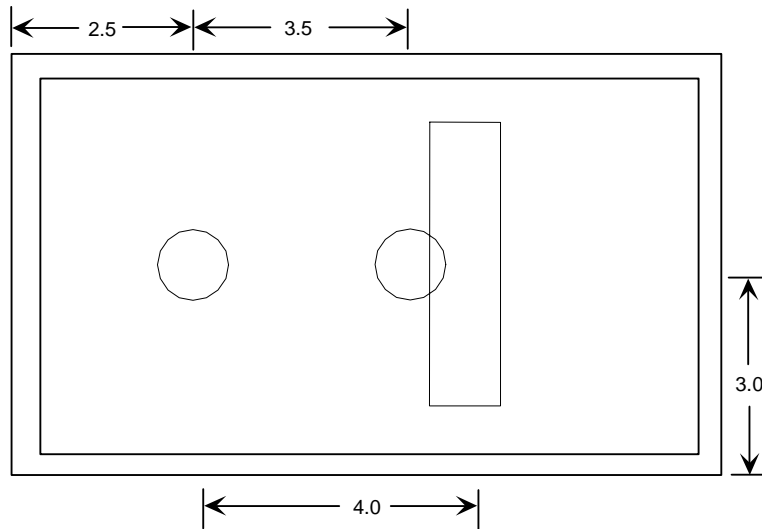
If any one of the vector loop equations do not sum to zero, the linear dependence signals that the design did not assemble. An important piece of predictive information can be gained from the **B** matrix when this happens.

An interesting physical interpretation from terms in the **B** matrix can be used to determine why a design will not assemble. Recall that the partial derivatives associated with the  $\theta_i$  variables represent the component of a resultant vector in a specified direction. As illustrated in Section 5.4.1, the partial derivative of  $h_x$  with respect to  $\theta_i$  will be the y-distance between the tip and tail of the string of vectors associated with that variable. Each end of this string of vectors will be at either a DRF or a constraint. If the tip is at one constraint and the tail is at another constraint (as was discussed in Fig. 5.2), the term in the **B** matrix for the partial derivative of  $h_x$  with respect to  $\theta_i$  will be equivalent to the y-distance between the constraints. Likewise, the partial derivative for  $\theta_i$  relating to  $h_y$  will be the x-distance between the constraints. It should be noted that while this information was independently discovered by the author, it has been more fully described in Gao et. al [1998].

This physical interpretation can now be applied to discover why a design did not assemble (as shown when at least one vector loop equation does not sum to zero). When solving the vector loop equations with an optimization routine or an equation solver such as Excel's Solver<sup>®</sup>, the last iteration before the routine terminates will show the minimum distance required between the constraints (or between the constraint and DRF, as the case

may be) for the design to assemble. For example, the **B** matrix shown below relates to Fig. 5.7, which shows a variation of the slotted block example from Fig. 5.1.

$$\mathbf{B} = \begin{array}{cc|cc} & & u_1 & \text{theta} \\ & & -0 & 0 \\ \mathbf{B} = & & 1 & 4 \end{array} \begin{array}{l} h_x \\ h_y \end{array} \quad \text{determinant} = -0$$



**Figure 5.7 – Example assembly to show how to use the **B** matrix to predict why a design did not assemble**

In order for this design to assemble, the two constraints must be 4 units apart in the x-direction (as shown by the term in the **B** matrix for the partial derivative of  $h_y$  with respect to  $\theta$ ); however, they are only 3.5 units apart. For the design to assemble, the constraints would have to be moved apart another 0.5 units, or the dysfunctional part would have to be re-manufactured to fit the dimensional needs found in the analysis.

#### 5.4.2.4 USING THE [B] MATRIX AS A MEASURE OF GOODNESS

Using the determinant of **B** proves useful for EC design. When the determinant for the **B** matrix goes to zero, the assembly has become over-constrained, under-constrained, or it does not assemble. The reason the assembly does not perform as an EC design can be inferred from the cause of the singularity in the **B** matrix.

One of the criteria for goodness is that the design is not approaching an over and/or under-constrained state. A quick inspection of the determinant of the **B** matrix is one measurement of goodness because information from the **B** matrix can detect stability of the assembly. When the matrix is singular, the determinant is zero. As the assembly approaches a state of over and/or under-constraint, the determinant gets smaller. Very small values for the determinant show that the design is approaching an undesired state.

The next task is to define “small values of the determinant.” Obviously the determinant depends upon the values found in the **B** matrix. Recall that all partial derivatives with respect to length variables have values between zero and one. However, the angular variables have partial derivative values that could be of any magnitude, depending on the dimensions of parts and locations of joints in the assembly. Therefore, the determinant could have many orders of magnitude, or it could have very few. A “small value”, therefore, ranges in orders of magnitude according to the dimensions of the assembly.

A general guideline can be followed to determine what a small value is for a given design. If the highest dimension involved in any vector loop is a value between one and ten, the designer can begin to be wary when the determinant is below an absolute value of one. If the highest dimension involved in any vector loop is between 0.1 and 1, the absolute value of the determinant should not go below 0.1. If the highest dimension in the assembly is on the order of 10 (such as 21), the determinant should not go below 10. The same general pattern can be applied to any given order of magnitude. This guideline shows where the assembly is beginning to approach an unstable state. It has not yet reached the over and/or under-constrained state, but the determinant warns that the assembly is approaching that status. The constraints should be altered to avoid a determinant that approaches zero, if possible.

With a square and non-singular **B** matrix, the assembly is exactly constrained. In that case, the sensitivities can also be used to quantify the goodness of similar designs.

### 5.4.3 USING ASSEMBLY SENSITIVITIES TO QUANTIFY GOODNESS

Now the discussion will return to the DLM procedure to find the assembly sensitivities. Once the **A** and **B** matrices have been found, the equation can be solved to find the sensitivities of the dependent variables with respect to the independent variables.

$$\{\delta U\} = -[B^{-1}A]\{\delta X\} \quad (3)$$

The matrix **B<sup>-1</sup>A** is the matrix of assembly sensitivities. They show what effect small changes of the independent variables will have on all the dependent variables in the assembly.

The effects of change are very important in analyzing an EC design because one of the benefits to EC design is that variation in parts or dimensions has little effect on the overall function of the assembly. Thus, another quantitative measurement of goodness for an EC design can be found using the assembly sensitivities.

The sensitivities show how the dependent variables will change with any change from the independent variables. For example, a sensitivity of one shows that changing the independent variable by some amount will change the dependent variable by the same amount. A sensitivity of two will force the dependent variables to change twice as much as the independent variables were changed.

The sensitivity matrix can be evaluated to ensure variation will not have a great effect on the assembly. The sensitivities should stay low in magnitude, as it would not be good to have the changes in dependent variables magnified when the independent variables are changed. Any sensitivity above two should be avoided, although doubling the changes in dependent variables compared to independent variables is still high.

If there is a high sensitivity somewhere in the matrix, the designer should avoid any variation related to that variable, or the design may not assemble. An alternative solution would be to change the locations of the constraints to lower the sensitivities. Note that when the determinant of the **B** matrix is zero, the sensitivities on the critical variables (the variables that cannot change because of any over-constraint) will go to infinity.



The sensitivities can also be compared between configurations for the same assembly to determine which designs may show a better measure for goodness. Comparing the assembly sensitivities among various configurations will give the designer a clearer idea for which designs will absorb variation better.

In addition to minimizing the effects of variation on an assembly, another criteria defined earlier for goodness includes a good tradeoff between the locations of the constraints for good reaction forces, and the size of the nesting force window. The assembly sensitivities can provide a quantitative way to show the effects of changing constraint locations. The sensitivities will change as the constraint locations change. Better sensitivities will lead to more robust designs. While the DLM as presented in this thesis does not bring the actual nesting force window into the analysis, the assembly sensitivities for various configurations of an assembly can be compared based on the constraint placements. The configurations that yield the best goodness values can be analyzed using the equations of equilibrium to find if the nesting force window is acceptable. This comparison is easily done in a spreadsheet or similar tool.

Using sensitivities is not new to engineering design. For example, Wittwer [2002] uses sensitivities from both the DLM and the force equations for micro-compliant mechanisms. However, using the sensitivities to derive a measure of goodness for EC design is unique.

#### 5.4.4 USING THE GOODNESS VALUES FOUND IN THE DLM

With all the information from this chapter in mind, a method can now be presented using the DLM to find a measure of goodness. The steps themselves are no different than performing the DLM on an assembly; however, the information within the DLM at various stages can now be used to form a mathematical or quantitative measure of goodness.

1. Find the **A** and **B** matrices as outlined by the DLM.
2. Examine the **B** matrix.
3. Find the assembly sensitivities,  $[\mathbf{B}^{-1}\mathbf{A}]$ .
4. Evaluate or compare the assembly sensitivities between configurations.

#### STEP 1: FIND THE [A] AND [B] MATRICES

Recall from the discussion in Chapter 2 that the DLM begins by creating vector loops and formulating vector loop equations. The three vector loop equations ( $h_x$ ,  $h_y$ ,  $h_\theta$ ) allow for up to three unknown variables per loop.

It should be noted that in many instances the  $h_\theta$  equation either falls out completely or it solves in terms of a user-defined value, allowing a substitution to eliminate one angular variable. In such circumstances, only two unknown dimensions can be found through the vector loop equations.

Regardless, solving the vector loop equations (which ordinarily requires a non-linear equation solver) finds the values for the unknown dimensions. While the DLM is more interested in small changes, it is necessary to find the unknown variables, as the values must be used to numerically evaluate the partial derivatives.

The next step in the method is to linearize the equations by taking partial derivatives of the vector loop equations. The **A** and **B** matrices are formulated after taking the partial derivatives of each equation with respect to each dependent and independent variable.

## **STEP 2: INSPECT THE [B] MATRIX**

At this stage, the **B** matrix must be inspected to ensure that it is both square and non-singular. It must meet these stipulations in order to get useful information out of the assembly sensitivities. Section 5.4.2 can be referenced to determine how the assembly will behave for a non-singular matrix.

If a design assembles, the value of the determinant for the **B** matrix is assigned to the assembly as a measure of goodness. The further away from zero the absolute value for the determinant is, the better the design. Values approaching zero (as defined in Section 5.4.2.4) should be avoided, if possible.

If a design does not assemble, the information in the **B** matrix can be used to make the appropriate changes. Then, the determinant can be evaluated once more to establish a specific measurement for goodness.

Thus, when inspecting the **B** matrix for an EC design, not only will the determinant be used as a measure of goodness, but the **B** matrix will also be used to predict what changes must happen in an assembly to fix a design that did not assemble. These two benefits alone provide a very powerful argument for using the DLM to analyze EC designs.

### **STEP 3: FIND THE ASSEMBLY SENSITIVITIES**

The  $\mathbf{B}^{-1}\mathbf{A}$  matrix contains the assembly sensitivities. These sensitivities show what effect changing one independent variable will have on each of the dependent variables.

### **STEP 4: EVALUATE OR COMPARE ASSEMBLY SENSITIVITIES BETWEEN CONFIGURATIONS**

Recall from an earlier discussion that part of the definition for goodness of an EC design is that variation in the parts has little to no effect on the function of the assembly. Assembly sensitivities show what effect small changes will have on the assembly. They can also show which constraints and variables have the greatest influence on the assembly. Assembly sensitivities above two should be avoided, if possible.

When comparing the assembly sensitivities between configurations of a design, the assemblies with lower sensitivities are better. The lower sensitivities show an ability of the design to absorb variation that may enter from part to part. Thus, using the assembly sensitivities also provides a quantitative measure for goodness of EC design.

#### 5.4.5 A RETURN TO THE SLOTTED BLOCK EXAMPLE

The simple slotted block example shown earlier in Fig. 5.1 can be used to illustrate the information presented in this chapter. Figure 5.8 shows the slotted block with the vector loop used in the DLM. The vector loop equations are expressed after the figure, and the **A** and **B** matrices are computed.

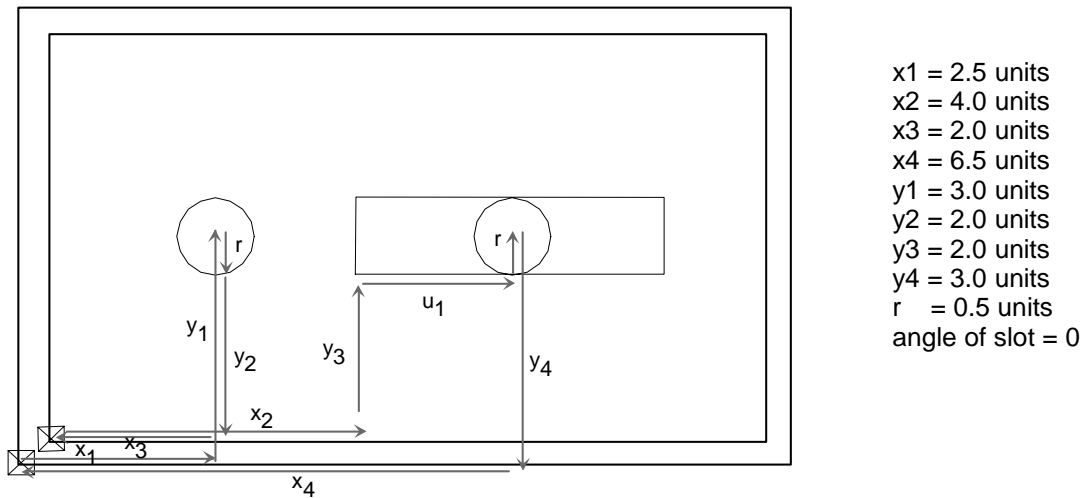


Figure 5.8 – DLM for the slotted block example

$$h_x = x_1 \cos(0) + y_1 \cos(90) + r \cos(270 + \theta) + y_2 \cos(270 + \theta) + x_3 \cos(180 + \theta) + x_2 \cos(\theta) + y_3 \cos(90 + \theta) + u_1 \cos(\theta + \phi) + r \cos(90 + \theta + \phi) + y_4 \cos(270) + x_4 \cos(180) = 0$$

$$h_y = x_1 \sin(0) + y_1 \sin(90) + r \sin(270 + \theta) + y_2 \sin(270 + \theta) + x_3 \sin(180 + \theta) + x_2 \sin(\theta) + y_3 \sin(90 + \theta) + u_1 \sin(\theta + \phi) + r \sin(90 + \theta + \phi) + y_4 \sin(270) + x_4 \sin(180) = 0$$

$$h_\theta = 0 + 90 + 180 + \theta + 0 - 90 - 180 + 90 - \phi + 90 + 180 - \theta - \phi - 90 - 180 = 0$$

$$\mathbf{A} = \begin{array}{cccccccccccc|c} & x_1 & x_2 & x_3 & x_4 & y_1 & y_2 & y_3 & y_4 & r & \text{phi} & \\ \hline & 1 & 1 & -1 & -1 & 0 & -0 & 0 & -0 & -0 & -0.5 & h_x \\ & 0 & -0 & 0 & 0 & 1 & -1 & 1 & -1 & 0 & 2 & h_y \end{array}$$

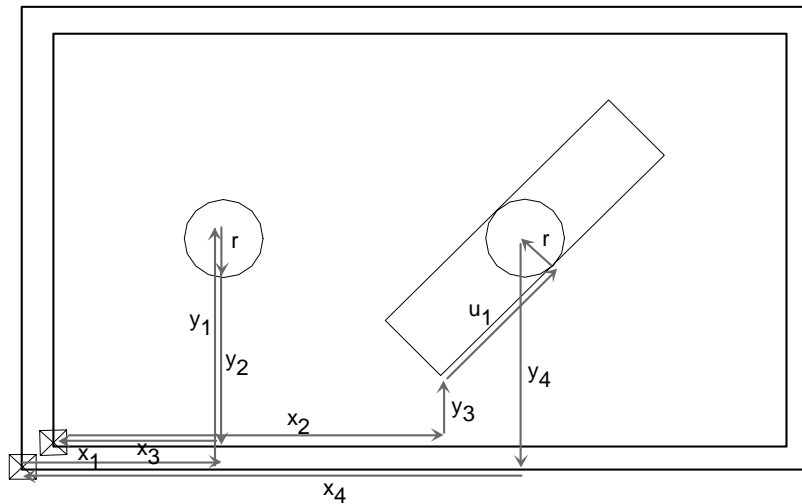
$$\mathbf{B} = \begin{array}{cc|c} & u_1 & \text{theta} & \\ \hline & 1 & 0 & h_x \\ & -0 & 4 & h_y \end{array} \quad \text{determinant} = 4$$

The determinant of the **B** matrix is four, which is greater than zero. There appear to be no signs that the assembly is on the verge of becoming over-constrained. The assembly is exactly constrained at 0°.

The overall sensitivities are very low (as shown below), no more than a one to one ratio of change between the independent and dependent variables. Variation in the dimensions can be absorbed. This set-up for the assembly will be quite robust, and it will stay exactly constrained.

$$\text{Sensitivities} = -(\mathbf{B}^{-1}\mathbf{A}) = \begin{array}{cccccccccccc|c} & x_1 & x_2 & x_3 & x_4 & y_1 & y_2 & y_3 & y_4 & r & \text{phi} & \\ \hline & -1 & -1 & 1 & 1 & 0 & 0 & 0 & -0 & 0 & 0.5 & u_1 \\ & -0 & 0 & -0 & 0 & -0 & 0.25 & -0.3 & 0.3 & 0 & -1 & \text{theta} \end{array}$$

Now, the slot will be rotated to  $45^\circ$  (Fig. 5.9), with the rest of the assembly remaining at the same nominal dimensions as previously used. The same vector loop equations apply, only the  $\phi$  has changed from  $0^\circ$  to  $45^\circ$ .



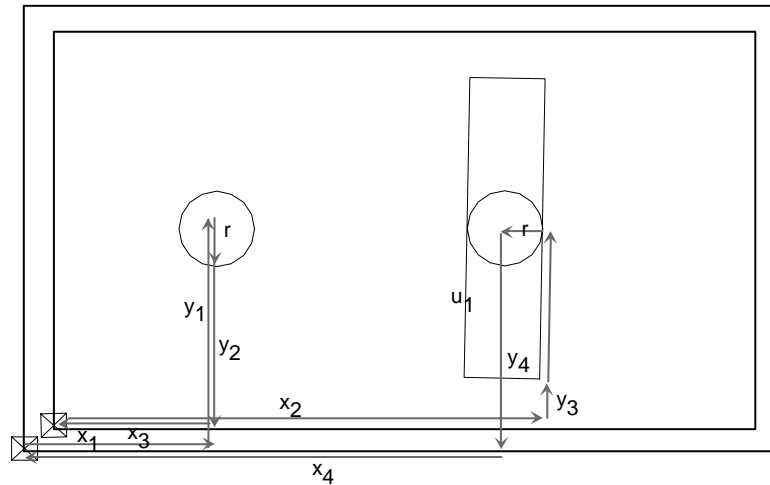
**Figure 5.9 – Slotted block with the slot at  $45^\circ$**

$$\mathbf{B} = \begin{array}{cc|cc} & u_1 & \text{theta} & \\ \mathbf{B} = & \left| \begin{array}{cc} 0.7 & 0 \\ 0.7 & 4 \end{array} \right| & \begin{array}{c} h_x \\ h_y \end{array} & \text{determinant} = 2.8 \end{array}$$

$$\text{Sensitivities} = -(\mathbf{B}^{-1}\mathbf{A}) = \begin{array}{cccccccccc|c} & x_1 & x_2 & x_3 & x_4 & y_1 & y_2 & y_3 & y_4 & r & \text{phi} & \\ \text{Sensitivities} = & \left| \begin{array}{cccccccccc} -1.4 & -1.4 & 1.4 & 1.41 & 0 & 0 & 0 & -0 & 1 & 2.5 \\ 0.2 & 0.25 & -0.3 & -0.2 & -0 & 0.25 & -0.3 & 0.3 & -0.1 & -1 \end{array} \right| & \begin{array}{c} u_1 \\ \text{theta} \end{array} \end{array}$$

Again, for the given dimensions and configuration, the assembly stays exactly constrained. The determinant is still greater than zero, although it is closer to zero than the slot at  $0^\circ$ . The sensitivities are also higher than when the slot is at  $0^\circ$ . The effect of change on the design will be greater in this set-up than in Fig 5.8, but the design will still assemble.

Figure 5.10 now shows the block at 89°. Screw theory indicates that this assembly is exactly constrained, but it gives no indication or warning of how close the assembly is to becoming over and under-constrained.



**Figure 5.10 – Slotted block with the slot at 89°**

$$\mathbf{B} = \begin{array}{cc|cc} & & u_1 & \text{theta} \\ & & 0.017 & 0 \\ \hline & & 1 & 4 \\ \hline & & h_x & \text{determinant=} 0.07 \\ & & h_y & \end{array}$$

$$\text{Sensitivities} = -(\mathbf{B}^{-1}\mathbf{A}) = \begin{array}{cccccccccc|cc} & x_1 & x_2 & x_3 & x_4 & y_1 & y_2 & y_3 & y_4 & r & \text{phi} & & \\ \hline & -57.3 & -57.3 & 57.299 & 57.299 & 9\text{E-}11 & 0 & 0 & -0 & 57.29 & 115.08 & u_1 & \\ \hline & 14.32 & 14.322 & -14.32 & -14.322 & -0.25 & 0.25 & -0.3 & 0.25 & -14.075 & -28.65 & \text{theta} & \end{array}$$

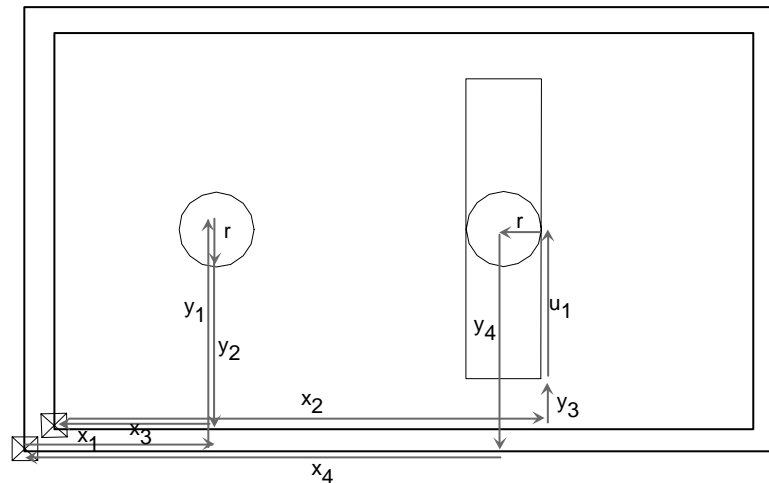
The determinant of this **B** matrix is 0.07, which is less than one. The simple indicator of the determinant shows that while this system is indeed exactly constrained, it is dangerously close to becoming over and/or under-constrained. If there is much variation in the right peg, the block will not properly assemble unless there is some deformation involved.



The sensitivities are also very high. In some instances, the dependent variables would have to change 57 times as much as the independent variables changed. The sensitivities quickly indicate that the design cannot absorb variation from most of the variables.

The sensitivities can be compared between configurations of this design. Based on the effects of variation, the  $0^\circ$  or  $45^\circ$  slotted block assemblies would be chosen over the  $89^\circ$  assembly because the goodness shown by the assembly sensitivities of the other two designs are better. All three designs are exactly constrained, but the  $89^\circ$  design is on the verge of not being exactly constrained.

Figure 5.11 shows the same assembly. However, this time the slot is at  $90^\circ$ .



**Figure 5.11 – Slotted block with slot at  $90^\circ$**

$$\mathbf{B} = \begin{array}{cc|cc} & & u_1 & \text{theta} \\ & & 0 & 0 \\ & & 1 & 4 \end{array} \begin{array}{l} h_x \\ h_y \end{array} \quad \text{determinant} = -0$$

$$\text{Sensitivities} = -(\mathbf{B}^{-1}\mathbf{A}) = \begin{array}{c} x_1 \quad x_2 \quad x_3 \quad x_4 \quad y_1 \quad y_2 \quad y_3 \quad y_4 \quad r \quad \text{phi} \\ \left| \begin{array}{cccccccccc} 5\text{E}+15 & 5\text{E}+15 & -5\text{E}+15 & -5\text{E}+15 & -7831 & -0.7 & -1 & 7830 & -5\text{E}+15 & -1\text{E}+16 \\ -1\text{E}+15 & -1\text{E}+15 & 1\text{E}+15 & 1\text{E}+15 & 1957 & 0.43 & 0 & -1957 & 1\text{E}+15 & 2\text{E}+15 \end{array} \right| \begin{array}{l} u_1 \\ \text{theta} \end{array} \end{array}$$

The determinant of the **B** matrix is zero. Inspecting the matrix can show the reason why: the x-equation in the **B** matrix has gone to zero, and over-constraint in the x-direction has resulted. Also, there is linear dependence between  $u_1$  and  $\theta$ , which shows under-constraint in rotation.

The sensitivities have nearly all exploded. This assembly cannot absorb any changes in the variables, except for  $y_2$  or  $y_3$ , and still assemble.

The results from screw theory concur with these findings. Also, a visual inspection indeed shows that the constraint lines of action do line up in the x-direction.

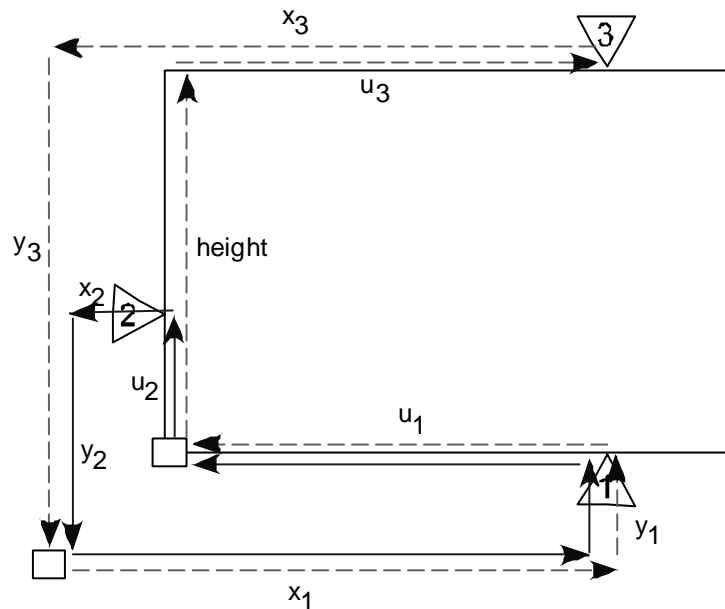
### 5.5 THE [B] MATRIX FROM THE DLM AND THE [C] MATRIX FROM THE EQUATIONS OF EQUILIBRIUM

Recall that screw theory begins in twist space. Reciprocal operations are performed to take the twists into wrench space (the force domain), and then the operations are applied again to bring the analysis back into twist space (the motion domain) to find possible motion in the assembly. Likewise, to obtain results from the constraint analysis, the reciprocal operations are applied to the unionized twists in order to enter into the wrench space. From this information, the constraint status is found. In

short, the motion and force domains are reciprocal to each other through the “reciprocal operation” defined for screw theory.

Section 4.4.2 Step 2 showed that the  $\mathbf{C}$  matrix in the force analysis goes singular when an assembly, which was exactly constrained given constraints at some distance apart, became over and under-constrained when two of the constraints became co-linear. In the force analysis, the moment equation (first row) went to zero, thus showing that the assembly became under-constrained in rotation.

The DLM can also be applied to this example, shown in Fig. 5.12. The vector loop equations and  $\mathbf{B}$  matrix follow the figure.



**Figure 5.12 – Exactly constrained block with three constraints**

$h_1 = \text{solid line}$        $h_2 = \text{dashed line}$

$$h_{x1} = x_1 \cos(0) + y_1 \cos(90) + u_1 \cos(180 + \theta) + u_2 \cos(90 + \theta) + x_2 \cos(180) + y_2 \cos(270) = 0$$

$$h_{y1} = x_1 \sin(0) + y_1 \sin(90) + u_1 \sin(180 + \theta) + u_2 \sin(90 + \theta) + x_2 \sin(180) + y_2 \sin(270) = 0$$

$$h_{\theta1} = 0 + 90 + 90 + \theta - 90 + 90 - \phi + 90 + 90 = 0$$

$$h_{x2} = x_1 \cos(0) + y_1 \cos(90) + u_1 \cos(180 + \theta) + height \cos(90 + \theta) + u_3 \cos(\theta) + x_3 \cos(180) + y_3 \cos(270) = 0$$

$$h_{y2} = x_1 \sin(0) + y_1 \sin(90) + u_1 \sin(180 + \theta) + height \sin(90 + \theta) + u_3 \sin(\theta) + x_3 \sin(180) + y_3 \sin(270) = 0$$

$$h_{\theta2} = 0 + 90 + 90 + \theta - 90 - 90 + 180 - \phi + 90 + 90 = 0$$

$$\mathbf{B} = \begin{array}{cccc|ccc} & u_1 & u_2 & u_3 & \text{theta} & & & \\ & -1 & 0 & 0 & -2.5 & h_{x1} & & \\ & 0 & 1 & 0 & -7.64 & h_{y1} & \text{determinant=} & 0 \\ & -1 & 0 & 1 & -6.67 & h_{x2} & & \\ & 0 & 0 & -0 & 0 & h_{y2} & & \end{array}$$

Notice the last row of the **B** matrix. It shows that the assembly is over-constrained in the y-direction for the second vector loop (the dashed vector loop). That loop shows that the over-constraint happens between constraints 1 and 3. Upon visual inspection and in agreement with the **C** matrix from the equations of equilibrium, at the point when the two constraints line up, it is indeed the case that the assembly has become over-constrained in the y-direction and under-constrained in rotation.

The **B** matrix from the DLM and the **C** matrix from the equations of equilibrium simply show two sides to the same analysis. They appear to be reciprocal to each other.

Appendix G gives a table showing the apparent relationship for Fig. 5.12. Thus, through the equations of equilibrium and the DLM, the status of the assembly (exactly constrained, over-constrained, or under-constrained) can be found.

## **5.6 CONCLUSIONS**

This chapter showed how to use the DLM to find two measures of goodness for an EC design. The determinant of the **B** matrix can help the designer understand when an EC design is approaching an unstable condition. The rows and columns in the **B** matrix can also be used for constraint analysis. The assembly sensitivities can be used to quantify the effects of variation on an assembly. These values for goodness can help a designer make a decision on appropriate configurations for an assembly.

However, as will be shown in Chapter 6, the DLM also provides additional understanding for EC design, especially as it relates to variation in positioning of the constraints. Chapter 6 will use the DLM to show any effects of variation, and it will illustrate the robustness of an EC design vs. an over-constrained design.

## **CHAPTER 6 USING EXACTLY CONSTRAINED DESIGN AS A ROBUST DESIGN METHOD**

### ***6.1 INTRODUCTION***

Each of the previous chapters discussed one tool that strengthens the quantitative foundation for exactly constrained (EC) design. Chapter 3 validated the heuristic rules of EC design using the equations of equilibrium as a foundation. Chapter 4 developed a generalized method to use the equations of equilibrium to inspect and/or predict the effects of different configurations for an EC assembly. Chapter 5 introduced the direct linearization method (DLM) as a means to determine a measurement of goodness for varying arrangements of the same design, while also providing warning signals when the assembly is approaching an under or over-constrained status.

Each chapter thus demonstrates a quantitative evaluation for EC design, and each method adds insight for when certain configurations approach or have become over and/or under-constrained. Avoiding the configurations that could lead to over or under-constrained assemblies preserves the advantages described in Chapters 1 and 2 for EC design.

The ability of an EC design to assemble under a wide variety of conditions is one advantage that has been referenced over and over again in this thesis. However, the mere

suggestion that such an advantage exists does not substantiate the claim. Chapter 6 now quantitatively explores the robust nature of EC design in greater detail.

This chapter begins with a brief description of the method developed to show robustness. That explanation is followed by examples that show the robust nature of EC design vs. similarly over-constrained (OC) designs.

## **6.2 *MONTE CARLO SIMULATION TO SHOW THE ROBUST NATURE OF EC DESIGN***

As with all other methods presented previously in this thesis, the robust nature of EC design must be shown through a quantitative means. This section will show how a Monte Carlo simulation provides a quantitative method to effectively illustrate the robust nature of EC design.

First, it must be understood that robust designs will assemble under a wide variety of conditions, although there may still be error. They do this by absorbing the variation allowed by the dimensional tolerances. As variation inevitably arises in real world assembling processes, it must be included in any analysis.

A Monte Carlo simulation will be used to show the effects of variation on assemblies. Each run of the Monte Carlo simulation will vary the position of specific constraints to determine the effects of that variation. Following the variation, two questions will be answered.

1. Will the design assemble?
2. If it does assemble, what is the overall error?

### **6.2.1 WILL THE DESIGN ASSEMBLE?**

The vector loop equations will be used in the Monte Carlo simulation to determine whether the design assembles. Recall that vector loop equations must sum to zero; and if they do, it can be stated that the vector loop equations “close.” (Physically, if something assembles, it means that all parts make contact with each other.) If any equation does not close, the design fails to assemble.

The simulation determines if the loops close by using a Newton-Raphson (NR) routine. If the NR routine converges, the design assembles. The NR is considered to have converged if the residuals for the loops fall below a value of 0.000001 in less than 15 iterations (Examples of the actual code can be found in Appendix E). Otherwise, the NR routine fails, and the design does not assemble.

A design will be considered robust if it consistently assembles, even after variation is introduced. Designs that assemble even when subjected to variation are more robust than those designs that do not assemble when variation is present.

### **6.2.2 WHAT IS THE OVERALL ERROR?**

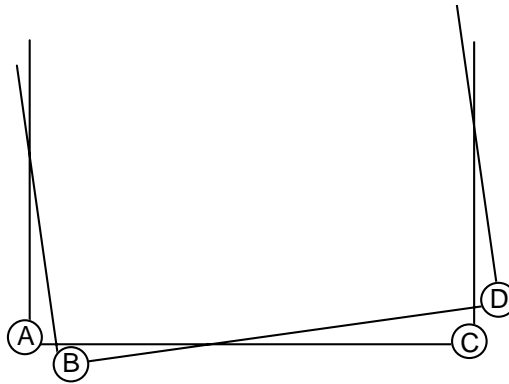
As variation enters into the components of an assembly, the possibility of error associated with the position arises. It is assumed that in some cases, the assembled



position of the design is important. Therefore, the error found in the Monte Carlo simulation will reflect the error in the position of the assembly.

When all of the parts in an assembly are at their nominal positions, the error in the assembly is zero. As positions or dimensions change, error may increase in the overall assembly. Therefore, if the Monte Carlo finds that the design assembles, the possible error in the assembly is found.

A root-sum-squared method is used to find the error. For example, consider Fig. 6.1 below. The error in this figure is found by subtracting the resulting position of the left bottom corner of the assembly (B) from the nominal position of the left corner (A). Likewise, the resulting position of the right bottom corner (D) will be subtracted from the nominal position of the right corner (C). Each term is squared, and all terms are then added together. The square root is taken to find the overall error for each assembly. Appendix E includes the error calculations in the C program for each example presented in this chapter.



$$error = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2 + (x_C - x_D)^2 + (y_C - y_D)^2}$$

**Figure 6.1 – Example calculation for the error of an assembly**

### 6.3 EXAMPLES

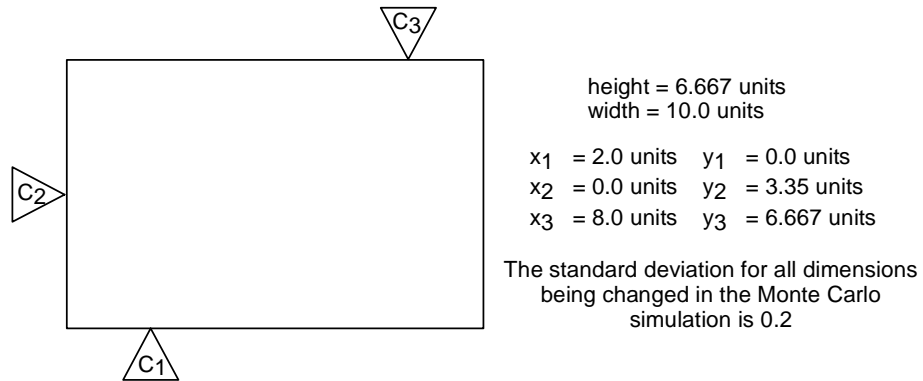
The Monte Carlo simulation will be used to simulate the effects of variation on assemblies. It will show how robust an EC design can be compared to a similar over-constrained design. During the course of each simulation, two main details are recorded: the number of runs that assemble and the overall average error for the assembly.

Several examples will now be presented to show the effects of variability on EC and OC designs. The first example will show the effects of variation on the block assembly used in Chapters 3, 4, and 5. The slotted block example, from Chapters 2 and 5, follows thereafter.

#### 6.3.1 EC BLOCK WITH THREE CONSTRAINTS

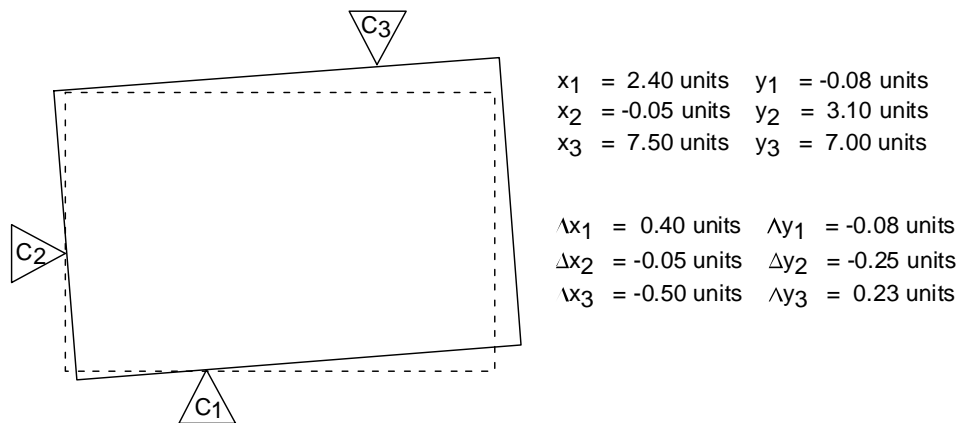
Figure 6.2 shows the familiar block assembly used in previous chapters. For this example, the Monte Carlo simulation varies the positions of all three constraints in the x and y-directions. The block is allowed to rotate in order to restore contact with the three

constraints. The nominal dimensions for each constraint are shown alongside the graphic.



**Figure 6.2 – Block assembly with three constraints**

Figure 6.3 shows an example for one run of the Monte Carlo simulation. All the dimensional changes are noted in the figure. Note that the block has rotated in order to maintain contact with the constraints.



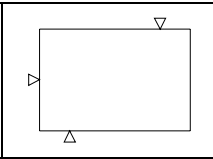
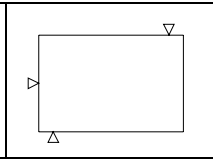
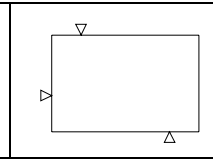
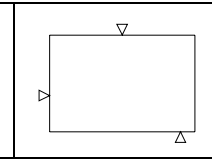
**Figure 6.3 – Block assembly with constraints at varying positions from the nominal**

Taking into account the variation in Fig. 6.3, the block still assembles with an error of 0.64 units. The overall angle of the block is now  $4.41^\circ$ . Even with the introduction of variability, the design assembles, albeit with some error.

After simulating 100,000 designs based on the information in Fig. 6.2, the results show that the EC block assembles 100% of the time with an average error of 0.46 units. The C code and Excel<sup>®</sup> spreadsheet with the DLM and error calculations used to find all the results can be found in Appendix E.

Recall from Chapter 5 that as the constraints are moved around, the goodness of the design changes. The goodness of an EC design can get better or worse depending on how the constraints are placed. One facet involved in the definition of goodness was that the error should be kept to a minimum. Table 6.1 shows the % assembled and error results of several configurations (different starting points) for this assembly. It also shows the goodness values. Recall that designs with higher absolute values of goodness were better EC designs.

**Table 6.1 – Block assembly with different starting points for the constraints**

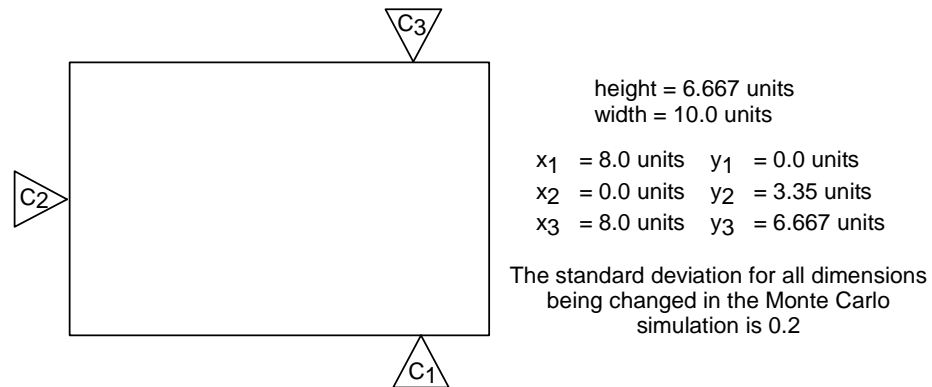
				
	Fig 6.2			
Height	6.67 units	6.67 units	6.67 units	6.67 units
length	10 units	10 units	10 units	10 units
x1, y1	2.0, 0.0 units	1.0, 0.0 units	8.0, 0.0	9.0, 0.0
x2, y2	0.0, 3.35 units	0.0, 3.35 units	0.0, 2.5	0.0, 2.5
x3, y3	8.0, 6.67 units	9.0, 6.67 units	2.0, 6.67	5.0, 6.67
Standard deviation for all variables	0.2	0.2	0.2	0.2
% assembled	100	100	100	100
Average error	0.46 units	0.41 units	0.46 units	0.61 units
Goodness ( <b>B</b> matrix determinant)	-6	-8	6	4

From Table 6.1, the same trends for goodness exist as had been found in Chapter 5. The further apart the constraints are from each other, the better the error. As the constraints get closer together, the error increases. However, as expected, all configurations assembled for these EC designs.

### 6.3.2 NON-EC BLOCK WITH THREE CONSTRAINTS

Now, attention will turn to when the assembly from Fig. 6.2 is no longer exactly constrained. This happens when constraints 1 and 3 line up. Several previous chapters explained why Fig. 6.4 does not qualify as an EC assembly. From a force perspective, there is no constraint in place that can overcome the resultant moment that would occur when the nesting force is applied, and constraints 1 and 3 are competing to constrain the

y-direction. That leaves the block with play and looseness if everything is not perfectly assembled. From a DLM perspective, the vector loops may not close if any of the independent variables in the y-direction change.



**Figure 6.4 – Over/Under-constrained block assembly with three constraints**

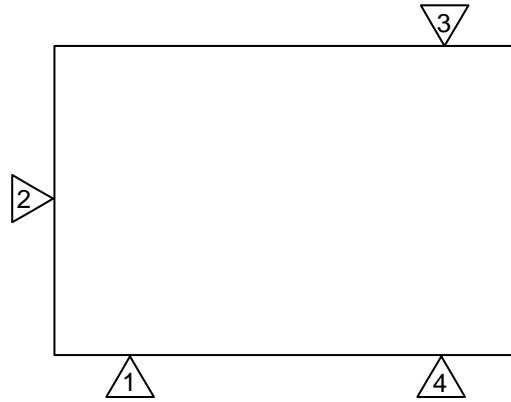
100,000 runs of the Monte Carlo simulation show the effects of variation on this non-EC design. It reveals that only 50% of the runs assembled, and they had an average error of 3.02 units! The error is significantly higher due to the negative effects of variation—this block must rotate significantly for even small variations in the y-direction on constraints 1 and 3 just to maintain contact with all the parts. Over-constraining the design in this manner significantly reduced the ability of the block to assemble, and it significantly increased the error. It is clearly not as robust as the EC design.

### 6.3.3 OC BLOCK WITH FOUR CONSTRAINTS

Now, another constraint will be added along the bottom of the block (Fig. 6.5).

Note from discussions in earlier chapters that this assembly is over-constrained in several

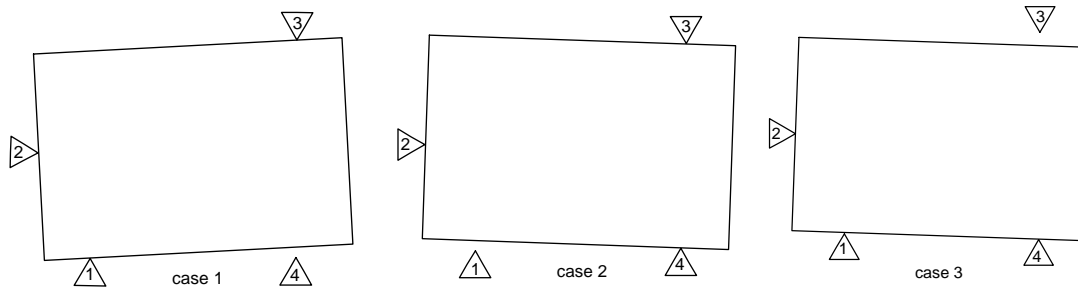
ways: there are four constraints, the two constraints to the right of the block (constraints 3 and 4) are co-linear, and constraints 2, 3, and 4 intersect at the same point.



**Figure 6.5 – Block assembly with four constraints**

The fourth constraint adds new complexity to the analysis of this example. Two conditions must be tested to determine if the block assembles: the vector loop equations must close, and the fourth constraint must not interfere with the block.

To learn if the vector loops close requires a closer look at the behavior of this assembly. When variability enters into the assembly, only three of the four constraints will stay in contact with the block. There are three possible arrangements (Fig. 6.6) for the assembly if the constraints are not in the nominal position shown in Fig. 6.5. It is uncertain which of the three options the assembly will assume. Thus, each case must be checked in each simulation of the Monte Carlo. If any one of the three cases assembles (the vector loops close), then the design is considered to have passed the first assembly test.



**Figure 6.6 – Three possible assembly cases for the over-constrained block**

However, after it is determined if the vector loops close, the simulation must check to ensure the fourth constraint does not interfere with the assembly. It must stay clear of the block. If it is out of the way of the block, the design assembles. Otherwise, the assembly fails.

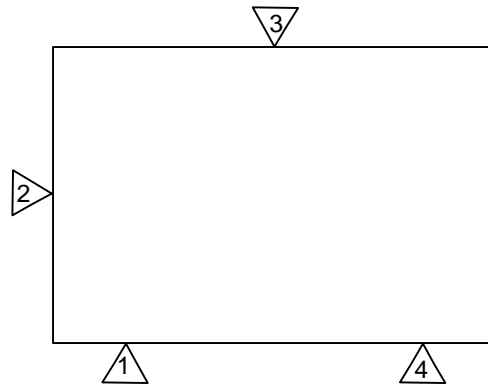
The error primarily comes from the play in the assembly due to the over-constraint present. For each run of the Monte Carlo simulation that does assemble, the program finds two values for the assembly error: average error and maximum error. To find the average error, the program averages the error of all the cases that assembled. The maximum error is found by comparing the errors of each case that assembled and keeping the maximum value.

The reported average error is the overall average of the individual average errors found above. The reported average maximum error is the average of all the maximum errors found above. The maximum error is found because the worst case assembly is just as likely as any other case to assemble. To provide a fair measure of the error, both the average and maximum errors are reported.



Including variation, the block in Fig. 6.5 will only assemble 50% of the time. The average error is 0.67 units, and the average maximum error is 1.23 units. It is interesting to note that when the design assembles to case 1 or case 3, the average error stands at 0.31 units. However, when the design assembles to case 2, the average error is significantly greater at 3.12 units. These results concur with earlier examples in Sections 6.3.1 and 6.3.2.

Recall that Fig. 6.5 had three different reasons that the block was over-constrained. If constraint 3 is moved over to the center of the block (Fig. 6.7), that eliminates two of the reasons for the over-constraint in this assembly. Now, the block is over-constrained only because there are four constraints.



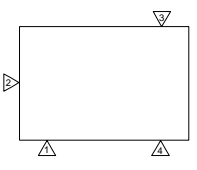
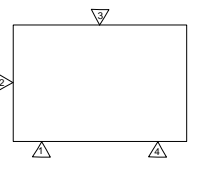
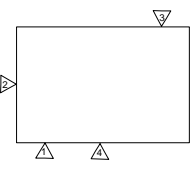
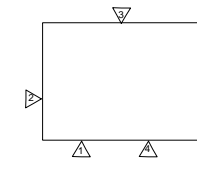
**Figure 6.7 – Alternate configuration for the over-constrained block**

The design will still only assemble 50% of the time. However, the error is considerably more reasonable. The average error is 0.54 units, and the average maximum error is 0.74 units. Each individual case has an average error of 0.60, 0.60, and 0.41 units, respectively. Eliminating two forms of over-constraint in this block still led to 50%

assembly (as has been the case for all over-constrained examples to this point), but the error present in this example is closer to that obtained by the EC example.

The results of Fig. 6.5 and 6.7 suggest that there are also varying degrees of goodness for over-constrained designs as well. The goodness for an over-constrained design is beyond the scope of this thesis; however, varied configurations show that as long as there is the fourth constraint, the EC block will always be more robust than any OC block. Table 6.2 shows the results for various nominal configurations of the assembly with four constraints.

**Table 6.2 – Several over-constrained examples**

				
	Fig. 6.5	Fig. 6.7		
Height	6.67 units	6.67 units	6.67 units	6.67 units
length	10 units	10 units	10 units	10 units
x1, y1	1.0, 0.0	1.0, 0.0	1.0, 0.0	2.0, 0.0
x2, y2	0.0, 3.35	0.0, 3.35	0.0, 3.35	0.0, 2.5
x3, y3	9.0, 6.67	5.0, 6.67	9.0, 6.67	4.0, 6.67
x4, y4	9.0, 0.0	9.0, 0.0	5.0, 0.0	5.5, 0.0
Standard deviation for all variables	0.2	0.2	0.2	0.2
% assembled	50	50	99	51
Average error	0.67	0.54	0.59	0.83
Average maximum error	1.23	0.74	0.66	1.16

The two new examples in the far right columns of Table 6.2 move the vertical constraints closer together. As constraints 1 and 4 move closer together (as in the fourth column of Table 6.2), the results become very comparable to an EC design. Recall that if any one of the three cases in Fig. 6.6 assembles, the design assembles. With the fourth constraint placed to the inside of the third constraint, the block has a greater range for rotation, and the extra constraint interferes less (it still interferes in some cases, but it never interferes in all cases in any run of the simulation). Hence, the results are similar to that of an EC design. It can be noted that when constraint 3 is at (8.0, 6.67) instead of (9.0, 6.67) for this example, the % assembly goes down to 90%, and the average and maximum errors increase to 0.68 and 0.75 units, respectively.

The example in the fifth column of the table shows that as constraint 3 is also moved closer to constraints 1 and 4, the percentage of successful assemblies plummets again, and the error escalates. Thus, although some OC designs exhibit good robustness, the placement of the constraints is very important. OC designs cannot absorb variation as well as EC designs, and they generally have more error.

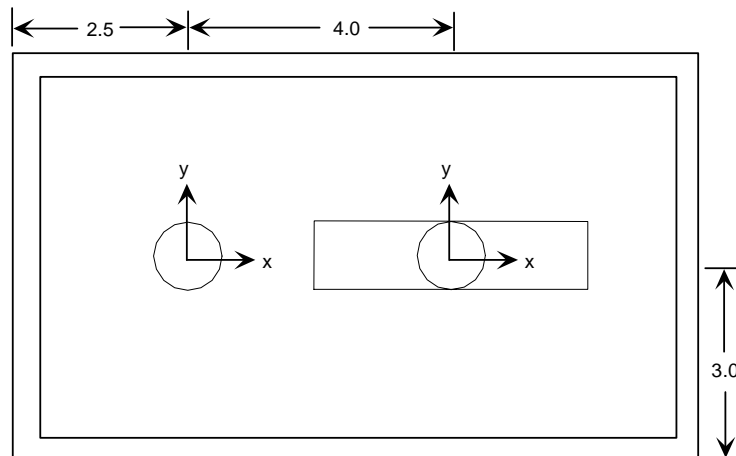
#### **6.3.4 SUMMARY OF THE BLOCK ASSEMBLIES WITH THREE OR FOUR CONSTRAINTS**

In summary, the exactly constrained assemblies in the previous examples have lower error with higher overall assembly rates than similar designs that are over-constrained. This observation, based on the results from the Monte Carlo simulation, provides key insight into the strength of EC design.

To re-emphasize, there is a very powerful benefit to EC design that is found from the Monte Carlo simulation. In the world of manufacturing, it is generally observed that assemblies with broader tolerances (and thus greater possibility for variation among assemblies) will lead to higher assembly rates, although the error also increases. Here it is shown that with an EC design, there need not be a negative tradeoff. Indeed, compared to the over-constrained designs, EC designs have greater assembly rates with lower overall error!

### 6.3.5 SLOTTED BLOCK ASSEMBLY

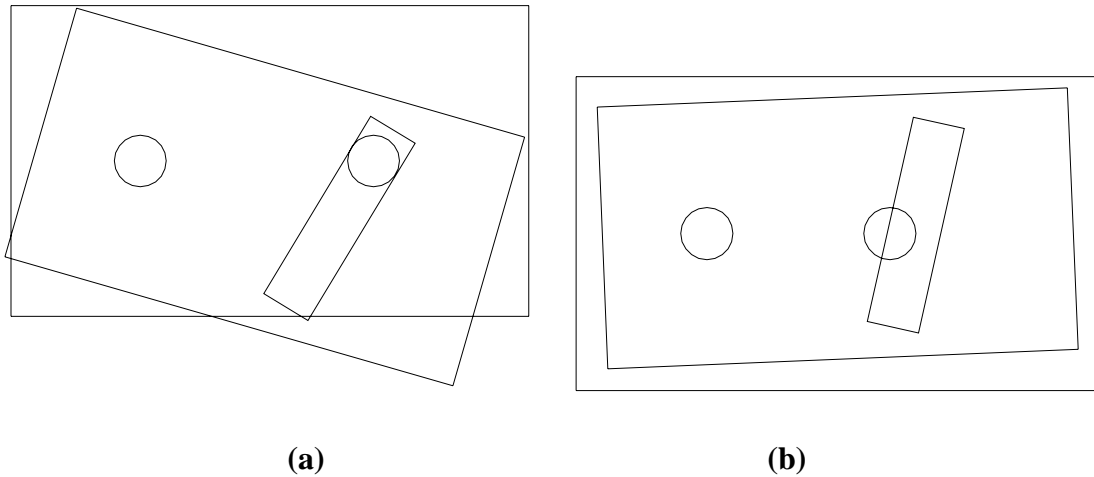
Figure 6.8 shows the slotted block assembly which was first introduced in Chapter 2. The nominal dimensions are listed in the figure.



**Figure 6.8 – Slotted block example**

The angle of the slot is a user defined input which can vary from  $0^\circ$  to  $90^\circ$ . The variation for this example is limited to the x-location of the right peg.

Due to the variation in the right peg, the top plate may have to rotate in order to allow the design to assemble. However, just because the top plate can rotate does not guarantee that the design will assemble. For example, Fig. 6.9 shows possible configurations for the slotted block when the slot has an angle of  $75^\circ$  and the right peg rests at various positions. The parts would assemble in Fig. 6.9a, but they would not assemble in 6.9b.



**Figure 6.9 – Slotted block assembly with the slot at  $75^\circ$  (a) the right peg at 7.0 units (b) the right peg at 6.0 units**

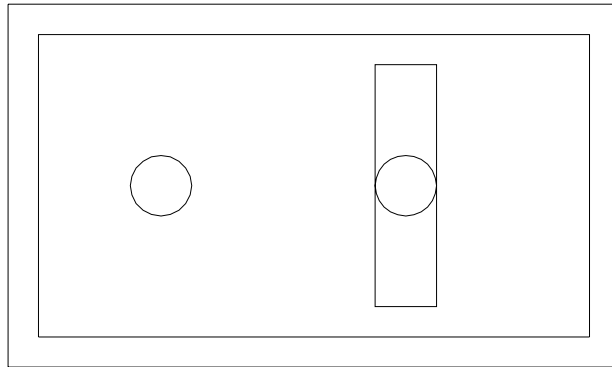
A Monte Carlo simulation will produce assembly results for the slotted block example at various slot angles. The right peg will be varied in the x-direction for each run of the simulation. Again, if the vector loops close after the variation has been introduced, the block assembles. If the block assembles, the program calculates the average error for all successful assemblies per slot angle. The error is again defined as the RSS displacement of the bottom two corners from their nominal positions (as defined in section 6.2.2).

### 6.3.5.1 SLOT ANGLE: 0°

With a 0° slot angle, previous chapters showed this block assembly to be exactly constrained. After 100,000 runs of the Monte Carlo simulation, the block assembled 100% of the time with no error. The slot absorbed the variation in the right peg, and all possible configurations assembled.

### 6.3.5.2 SLOT ANGLE: 90°

Chapter 5 showed the block assembly in Fig. 6.10 to be over-constrained using both the screw theory approach to constraint analysis and the **B** matrix in the DLM. After 100,000 runs of the Monte Carlo simulation, the total number of configurations that assembled totaled 48%, which is drastically lower than the 100% assembly rate for the same block with the slot manufactured at 0°.



**Figure 6.10 – Assembly with the slot angle at 90°**

Not only is the slotted block assembly less robust at 90°, it shows significant increases in error. The average error of this assembly is 3.27 units as compared to 0.00 units of error when the slot is at 0°.

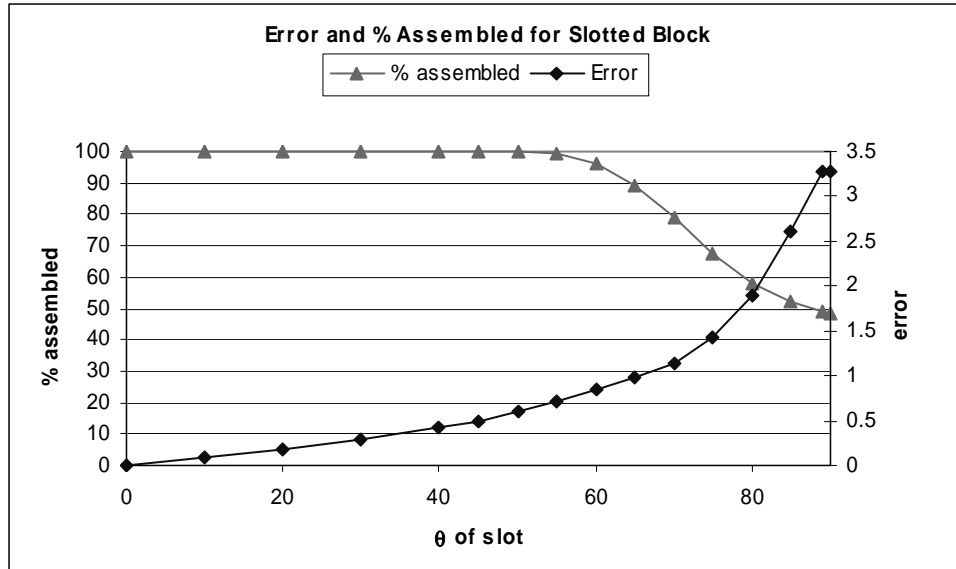
### 6.3.5.3 SLOT ANGLE: VARIED

Recall from Chapter 5 that screw theory considered all slotted block assemblies exactly constrained up to and through a slot angle of 89.9°. However, the goodness factor developed in Chapter 5 also showed that as the angle of the slot increases, the designs tend to be more sensitive to change, as evidenced by higher sensitivities. Thus, the goodness level decreases as the slot angle increases.

Table 6.3 and Fig. 6.11 combine to show the results of a 100,000 run Monte Carlo simulation for each listed angle. The goodness factor listed is found from the determinant of the **B** matrix in the DLM.

**Table 6.3 – Table of Monte Carlo results for various slot angles**

Slot angle	% assembled	Error	Goodness
0	100	0	4
10	100	0.086	3.94
20	100	0.179	3.76
30	100	0.284	3.46
40	100	0.417	3.06
45	100	0.501	2.83
50	99.8	0.603	2.57
55	99.2	0.724	2.29
60	96.3	0.848	2
65	89.4	0.973	1.69
70	78.9	1.14	1.37
75	67.5	1.44	1.04
80	58.0	1.90	0.69
85	51.9	2.60	0.35
89	48.7	3.27	0.07
90	48.2	3.27	0



**Figure 6.11 – Chart of results for slotted block assembly showing % assembled and error**

Figure 6.11 shows that 100% of the configurations assembled up to and through a slot angle of  $45^\circ$ . However, the error steadily increased for the same slot angles. The results for this range of slot angles correspond to the results for the block with three constraints in section 6.3.1. All configurations assemble, although those shown to have a lower goodness value have greater error.

Beyond  $45^\circ$  for the slot angle, both the error and the number of failures increase. While under screw theory these designs are called exactly constrained, it would perhaps be more correct to use Kamm's [1993] description for these designs: semi-MinCD or semi-exactly constrained. This description gives the designer a more realistic idea that while the design is not over-constrained, it really does not hold all of the benefits associated with an EC design.



## **6.4 CONCLUSIONS**

By using a Monte Carlo simulation, EC designs are found to be more robust than over-constrained designs. Not only do EC designs have a much greater percentage of successful assemblies, they also have lower error. These results concur with the constraint analysis from Chapters 3 and 4, and they strengthen the goodness findings of Chapter 5.

## **CHAPTER 7 CONTRIBUTIONS, CONCLUSIONS, AND RECOMMENDATIONS**

The purpose of this thesis was to establish a quantitative foundation for exactly constrained design. This chapter will explain the contributions and conclusions made in order to establish that foundation. Finally, recommendations will be made for further research that may be performed in this field of study.

### ***7.1 CONTRIBUTIONS OF THIS THESIS***

Exactly constrained (EC) design is a powerful and robust design method for mechanical assemblies. While many have defined it through heuristics or experience, this thesis begins to establish a quantitative foundation to both understand and use EC design in mechanical assemblies.

EC designs can be defined in quantitative terms by noting that they are statically determinate; therefore, the rules established by researchers and practitioners through years of experience are easily validated using the equations of equilibrium. Chapter 3 shows this contribution.

In addition to validating existing rules, this thesis also presents a quantitative method to analyze EC designs based on the location of the constraints. The equations of equilibrium can be used to determine if a design is exactly constrained. When poor

placement of the constraints leads to an over and/or under-constrained assembly, the **C** matrix can be used to see what over and/or under-constraint is present. The equations can also be used to predict which locations must be avoided so the design can stay exactly constrained. This method is presented in Chapter 4.

Another contribution of this thesis shows how the equations of equilibrium can be used to find the nesting force window. Through a quantitative method, the acceptable and unacceptable regions for the nesting force can be found in a simple and concise way. Finding the window is also presented in Chapter 4.

This work includes the development of a quantitative process to find the “goodness” of EC designs. As all designs are not created equal, this method could help a designer quantitatively compare similar designs to make an informed decision on which configuration would be best. Using the DLM, assembly sensitivities can be compared between designs, or the determinant of the **B** matrix can be inspected to make decisions on designs that would best suit the needs of the designer. Also, if a design has become unstable, the **B** matrix can be inspected to determine why.

By using a Monte Carlo simulation, the robust nature of EC design was clearly demonstrated over similar designs that were over-constrained. The EC designs consistently had 100% assembly rates with relatively low error. The OC designs had assembly rates of approximately 50%, with greater error.

## 7.2 CONCLUSIONS OF THIS THESIS

Exactly constrained design can effectively be described and analyzed using quantitative means. Both the equations of equilibrium and the DLM successfully provide ways to do so.

The equations of equilibrium can be used to validate, inspect or predict the behavior of an assembly based on the constraints in an EC design. The locations of the constraints can be monitored through the reaction forces. High reaction forces (especially reaction forces leading to infinity) must be avoided. Also, the nesting force can be appropriately placed after using the equations of equilibrium to find the window.

The **C** matrix (from the matrix form of the equations of equilibrium) can give a general overview for constraint analysis. It is important that it not be singular.

Using information from the DLM provides a method whereby a quantitative measure of goodness can be assigned to various EC designs. The determinant of the **B** matrix can indicate how close a design may be to approaching an over and/or under-constrained design. The determinant must not approach zero (a singular matrix), or the assembly will lose the benefits of being exactly constrained (inspecting the **B** matrix when it does go singular can be used for constraint analysis). The assembly sensitivities also provide a measurement that can be compared between designs to show which design in question may provide the best possible assembly under consideration. Lower sensitivities generally lead to “better” EC designs.

In addition to showing a measure of goodness for EC design, the DLM provides the means to show how robust an EC design can be. By implementing the DLM into a Monte Carlo simulation, EC designs are found to be more robust than similar OC designs.

Throughout the thesis, edge slider, cylinder slider, and revolute joints are used to constrain the assemblies. With a basic understanding of how degrees of freedom work, any type of joint that will allow the assembly to properly function can be used to constrain motion. The analysis method used to analyze the design, whether it be the equations of equilibrium or the direct linearization method (DLM), can find the necessary information, regardless of the type of joint used.

### **7.3 RECOMMENDATIONS FOR FUTURE WORK**

One area that could possibly be developed further is Section 5.5. As mentioned there, the **C** matrix from the equations of equilibrium and the **B** matrix from the DLM appear to tell two sides to the same story. Just as screw theory found a link through the reciprocal operation between the twist space and the wrench space, there may be a similar link by way of matrix operations between the **B** and **C** matrices.

For example, it could be stated that variation is analogous to velocity [Faerber 1999]. Therefore, through mechanical advantage, the force and velocity are reciprocal.

In addition to these considerations, the force analysis can be further extended to include sensitivity. The matrix equation employed in this thesis is very basic.

$$\mathbf{Cr} = \mathbf{b}$$

The vector  $\mathbf{b}$  could actually be separated to make the equation more descriptive.

$$\mathbf{Cr} = \mathbf{Df}$$

$\mathbf{C}$  is still a matrix of coefficients for the reaction forces in  $\mathbf{r}$ , and  $\mathbf{D}$  contains the coefficients for the nesting forces in  $\mathbf{f}$ . If this equation is linearized (as was done for the vector loop equations in the DLM), the equation begins to resemble the assembly sensitivity matrix in the DLM.

$$\{\delta r\} = [C^{-1} D] \{\delta f\}$$

$$\{\delta U\} = [B^{-1} A] \{\delta X\}$$

The matrix  $[\mathbf{C}^{-1}\mathbf{D}]$  now also provides a matrix of assembly sensitivities. The method presented in Section 5.4.4 on using the goodness values could now be extended to include two additional steps.

5. Formulate force equilibrium equations and linearize them to find  $\mathbf{C}$  and  $\mathbf{D}$ .
6. Examine the  $\mathbf{C}$  determinant and the  $[\mathbf{C}^{-1}\mathbf{D}]$  sensitivities.

The information from this thesis could be further implemented or linked into CAD systems to evaluate designs for over or under-constraint. If necessary, the automated process could fix the designs to achieve an exactly constrained assembly, and it would further optimize the configuration (through “goodness” values).

This thesis only treats 2D assemblies. Further research should extend this information into 3D.

The definition for a quantitative measure of goodness was begun in this thesis. Work should continue to further examine this idea for EC design. In addition, the results for over-constrained designs suggested that there may also be a value for goodness for OC designs as well. Research could be extended to further define these considerations.

Most of the examples in this thesis used symmetrical configurations. It may be interesting to investigate non-symmetrical configurations of the constraints and nesting force(s) to learn more about their error or assembly rates.

Non-normal nesting forces should be further examined. It has been observed throughout the research process of this thesis that the nesting force window will change according to the number, position, and angle of the nesting force(s). It should be further explored. In addition, nesting moments should also be explored.

Further exploration is needed to investigate the tradeoff between the locations of constraints and the nesting force window. In particular, Pearce [2003] developed a method to analyze the placement of the nesting force using the DLM. By generalizing that method to find the nesting force window, it could be used in conjunction with the current method presented in this thesis to find a value for goodness that includes the

nesting force window. That investigation may show a better measure for goodness relating to the tradeoff found in Chapter 4.

The effects of clearance need to be investigated further. As long as a nesting force is present to ensure that the constraint stays in contact with the main assembly, clearance may not have any effect on the analysis. However, if the nesting force does not provide the seating necessary, clearance may become an issue. The amount of clearance needed in an over-constrained system is unclear.

A few other considerations could also be investigated as they relate to EC design. The effects of elastic deformation, such as press fit bearings, in an EC design could be investigated further. It is important to note that an EC assembly does not have to be comprised of parts that are exactly constrained. For example, ball bearings could be used as a component of an EC design.

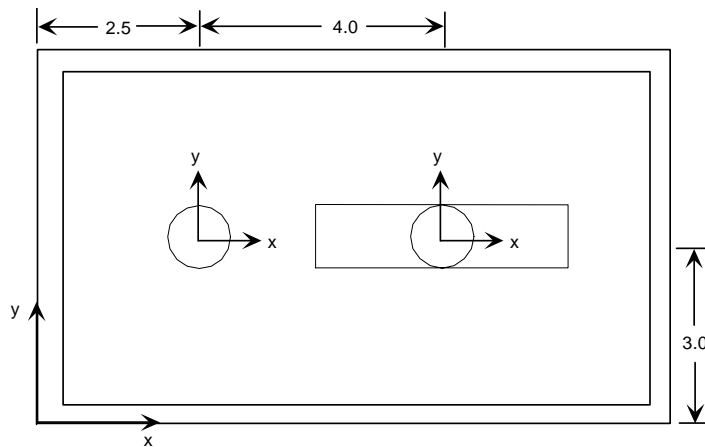
Finally, all mechanical assemblies under consideration in this thesis had no motion. Exactly constrained designs do not have to be immobile; therefore, future work should include extending the quantitative foundation to mechanisms that allow motion as well.





## APPENDIX A      CONSTRAINT ANALYSIS USING SCREW THEORY

This appendix will show the details involved to use screw theory as a constraint analysis method. The topic will be presented in outline form. All steps will be done on the following example.



### ***A.1 FIND THE TRANSFORMATION MATRIX***

The transformation matrix simply relates each individual joint in its local coordinate frame to the global coordinate frame. There is one transformation matrix per joint. The transformation matrix has the same form for each joint.

$$F = \begin{bmatrix} \mathbf{A} & \mathbf{d} \\ \mathbf{0} & 1 \end{bmatrix}$$

$\mathbf{A}$  is a 3x3 rotation matrix—based on direction cosines

$\mathbf{d}$  is a 3x1 displacement vector

$\mathbf{0}$  is a 1x3 row of zeros

The transformation matrix will be found for both joints in the example. As the local axis is the same as the global axis,  $\mathbf{A}$  is the identity matrix. When the slot is at  $0^\circ$ , the  $\mathbf{A}$  matrix is the identity matrix.

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

If the slot were at an angle of  $90^\circ$  to its current position,  $\mathbf{A}$  would have to properly define rotation between the two axes. The proper matrix for such a case is shown below.

$$\mathbf{A} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The  $\mathbf{d}$  vector is simply the displacement from the global coordinate zero. In this example, the left peg is 2.5 units away in the x-direction and 3.0 units in the y-direction. The right peg is 6.5 units in the x-direction and 3.0 units in the y-direction. Thus, the transformation matrix can be written as below.

$$F_{\text{left}} = \begin{bmatrix} 1 & 0 & 0 & 2.5 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad F_{\text{right}} = \begin{bmatrix} 1 & 0 & 0 & 6.5 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## A.2 FIND THE TWISTMATRIX FOR EACH FEATURE

The information from the transformation matrix can be used to find the necessary twists for each joint. There is one twistmatrix for each joint, and there are the same number of rows as there are degrees of freedom allowed by a joint. For example, the right peg's twist matrix will have two rows because it will allow both rotation and translation.

Translational motion is described in the twistmatrix as shown below.

$$T = [\mathbf{0} \quad \mathbf{v}]$$

$\mathbf{0}$  is a 1x3 vector of zeros

$\mathbf{v}$  is a 1x3 vector where  $\mathbf{v} = (\mathbf{A}\mathbf{k})^T$

$\mathbf{A}$  is defined in the transformation matrix

$\mathbf{k}$  is a 3x1 vector, representing the local axis along which the joint can translate

Rotational motion is described in the twistmatrix as shown below.

$$T = [\omega \quad \mathbf{v}]$$

$\omega$  is a 1x3 vector, where  $\omega = (\mathbf{A}\mathbf{w})^T$

$\mathbf{A}$  is defined in the transformation matrix

$\mathbf{w}$  is a 3x1 vector, which defines the allowed joint rotation, such as (0 0 1)

$\mathbf{v}$  is a 1x3 vector, where  $\mathbf{v} = \mathbf{r} \times \omega$

$\mathbf{r}$  is the 1x3 vector  $\mathbf{d}^T$ , as defined in the transformation matrix

$\omega$  is a 1x3 vector defined above

The twistmatrix for the right peg will be shown here. It has two degrees of freedom: one rotation and one translation. Thus, it will have two rows. The elements for rotation will be found first. Then, the elements will be found for the translation.

$$\omega = \left( \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right)^T = [0 \quad 0 \quad 1]$$

$$\mathbf{v}_{rot} = \begin{vmatrix} 6.5 & 3 & 0 \\ 0 & 0 & 1 \end{vmatrix} = 3\mathbf{i} - 6.5\mathbf{j} = [3 \quad -6.5 \quad 0]$$

$$\mathbf{0} = [0 \quad 0 \quad 0]$$

$$\mathbf{v}_{trans} = \left( \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right)^T = [1 \quad 0 \quad 0]$$

Finally, the twistmatrix for the right peg is shown below.

$$Twist_{right} = \begin{bmatrix} 0 & 0 & 1 & 3 & -6.5 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

### **A.3 DETERMINE IF THE ASSEMBLY IS UNDER-CONSTRAINED**

Now that the twistmatrices have been found, the constraint analysis can begin. Finding whether an assembly is under-constrained begins by applying what is called a reciprocal operation to each twist. After this stage, the matrices are unionized and the system is row reduced. Finally, the resulting matrix is sent back through the reciprocal operation to find if and where the assembly is under-constrained.

#### **STEP 1: RECIPROCAL OPERATION APPLIED TO EACH TWIST**

The reciprocal operation entails taking the null space of each twistmatrix, which is easily done in Matlab<sup>®</sup>. After taking the null space, the matrix is transposed. Finally, a flip function is applied, such that columns 1 and 4 are swapped, 2 and 5 are swapped, and 3 and 6 are swapped. Each matrix is now a wrench.

#### **STEP 2: UNIONIZE THE MATRICES**

After the reciprocal operation is applied to each twist matrix, the wrench matrices are combined through a union. This operation simply means that all matrices are stacked together into one matrix.

#### **STEP 3: ROW REDUCED ECHELON FORM**

The row reduced echelon form of the unionized wrench matrix is found.

#### STEP 4: RECIPROCAL OPERATION APPLIED TO THE WRENCH

Finally, the reciprocal operation can be performed on the unionized reduced wrench matrix. The results of the reciprocal operation will show whether there is any under-constraint in the assembly. If the resulting twistmatrix is anything other than empty, there is motion.

If there is under-constraint, the point of motion can also be found through a “point algorithm” as illustrated in Adams [1998]. As this composition only relates to 2D models, only the point algorithm for rotation about the z-axis will be given.

$$\text{if } \omega_z \neq 0 \text{ then the point of rotation} = \begin{bmatrix} -\frac{v_y}{\omega_z} \\ \frac{v_x}{\omega_z} \\ 0 \end{bmatrix}$$

#### MATLAB<sup>®</sup> AUTOMATION OF THE PROCESS

The whole process can be easily implemented into Matlab<sup>®</sup>, as also shown in Adams and Whitney [2001]. The twistmatrices are found by hand, and they are entered as T1, T2, etc. into the command window. An m-file called “run.m” finds the resultant twistmatrix.

### **m-file – run.m**

```
W1=recip(T1);  
W2=recip(T2);  
WU=[W1;W2];  
WU=rref(WU);  
Twist=recip(WU)
```

### **m-file – recip.m**

```
function R = recip(T)  
p=(null(T))';  
R=flip(p);  
  
end
```

### **m-file – flip.m**

```
function W=flip(p)  
  
[i,j] = size(p);  
if j==6  
    for l=1:i  
        for k=1:3  
            W(l,k)=p(l,k+3);  
            W(l,k+3)=p(l,k);  
        end  
    end  
  
end  
  
end  
W;
```

The example shown in the beginning of this appendix results in an empty matrix;  
therefore, there is no under-constraint in the assembly.



#### ***A.4 DETERMINE IF THE ASSEMBLY IS OVER-CONSTRAINED***

To learn whether the assembly is over-constrained, the twistmatrices are first unionized. Then, the row reduced echelon form of the unionized twistmatrix is found, and that step is followed by the reciprocal operation. This leads to a wrench matrix.

##### **STEP 1: UNIONIZE THE TWISTMATRICES**

To find if there is over-constraint in an assembly requires the twistmatrices to be unionized before they are sent through the reciprocal operation.

##### **STEP 2: ROW REDUCED ECHELON FORM**

The row reduced echelon form of the resulting unionized matrix is found.

##### **STEP 3: RECIPROCAL OPERATION APPLIED TO THE TWIST**

The reciprocal operation, as described in section A.3.1, leads to a resultant wrench matrix which can show whether an assembly is over-constrained. The first three columns show if the body is over-constrained in the x, y or z translational directions. The last three columns show whether the body is over-constrained in the x, y, or z rotational directions.

If an assembly is over-constrained, the point algorithm mentioned in section A.3.4 can again be applied to find where the over-constraint happened.

## MATLAB<sup>®</sup> AUTOMATION OF THE PROCESS

As before, the process can be automated in Matlab<sup>®</sup>. The same m-files can be linked to a run file to find the over-constraint. To maximize efficiency, it is best to put both the motion (under-constraint) and force (over-constraint) analysis in the same run file. Only the additional information to run the over-constrained analysis is shown below.

### **m-file – run.m**

```
TU=[T1;T2];  
T=rref(TU);  
Wrench=recip(T)
```

For the example, the resulting wrench matrix is shown below. The explanation follows the matrix.

$$\text{Wrench} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

This matrix shows that the assembly is over-constrained in z-translation, x-rotation, and y-rotation. However, as the assembly is only 2D, these can be ignored. All possible degrees of freedom are constrained. This assembly is exactly constrained.

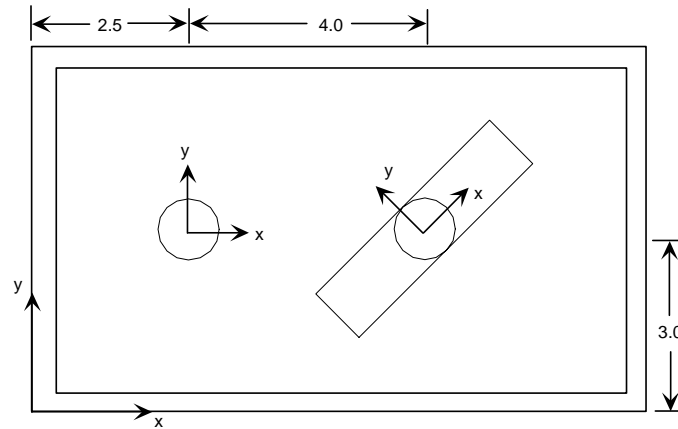
### ***A.5 DETAILED EXAMPLES FROM CHAPTER 5***

This section will show the development of the twist matrices and the results after running the slotted block example through the procedure described above. The slot will change in each example. Please note that  $F_{\text{left}}$  and  $\text{Twist}_{\text{left}}$  will always be the same in each example.

$$F_{\text{left}} = \begin{bmatrix} 1 & 0 & 0 & 2.5 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{Twist}_{\text{left}} = [0 \quad 0 \quad 1 \quad 3 \quad -2.5 \quad 0]$$

### SLOT AT 45°



$$Twist_{right} = \begin{bmatrix} \boldsymbol{\omega} & \mathbf{v} \\ \mathbf{0} & \mathbf{v} \end{bmatrix} \quad F_{right} = \begin{bmatrix} 0.707 & -0.707 & 0 & 6.5 \\ 0.707 & 0.707 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{\omega} = \left( \begin{bmatrix} 0.707 & -0.707 & 0 \\ 0.707 & 0.707 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right)^T = [0 \quad 0 \quad 1]$$

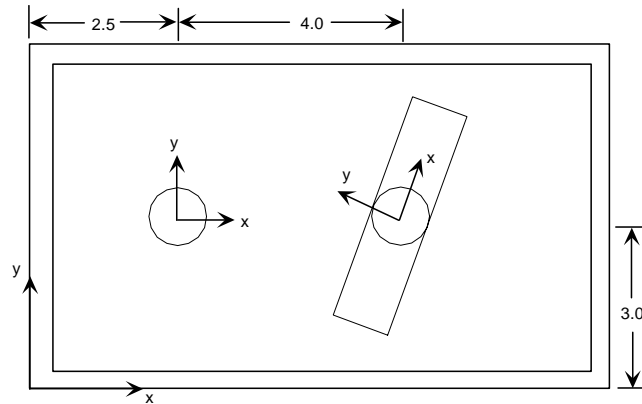
$$\mathbf{v}_{rot} = \begin{bmatrix} 6.5 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} = 3\mathbf{i} - 6.5\mathbf{j} = [3 \quad -6.5 \quad 0]$$

$$\mathbf{0} = [0 \quad 0 \quad 0] \quad \mathbf{v}_{trans} = \left( \begin{bmatrix} 0.707 & -0.707 & 0 \\ 0.707 & 0.707 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right)^T = [0.707 \quad 0.707 \quad 0]$$

$$Twist_{right} = \begin{bmatrix} 0 & 0 & 1 & 3 & -6.5 & 0 \\ 0 & 0 & 0 & 0.707 & 0.707 & 0 \end{bmatrix}$$

$$Twist = \text{empty matrix} \quad Wrench = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

### SLOT AT 70°



$$Twist_{right} = \begin{bmatrix} \boldsymbol{\omega} & \mathbf{v} \\ \mathbf{0} & \mathbf{v} \end{bmatrix} \quad F_{right} = \begin{bmatrix} 0.342 & -0.9397 & 0 & 6.5 \\ 0.9397 & 0.342 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{\omega} = \left( \begin{bmatrix} 0.342 & -0.9397 & 0 \\ 0.9397 & 0.342 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right)^T = [0 \ 0 \ 1]$$

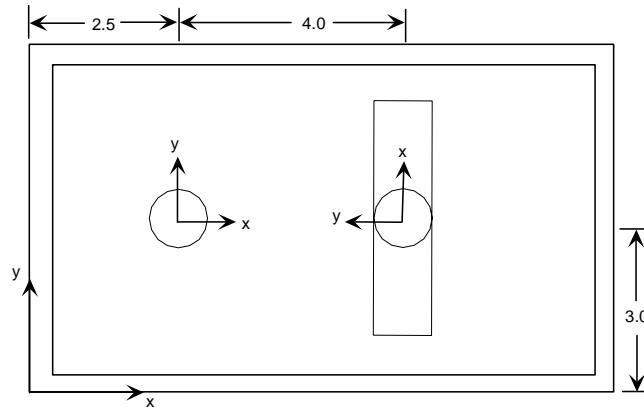
$$\mathbf{v}_{rot} = \begin{bmatrix} 6.5 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} = 3\mathbf{i} - 6.5\mathbf{j} = [3 \ -6.5 \ 0]$$

$$\mathbf{0} = [0 \ 0 \ 0] \quad \mathbf{v}_{trans} = \left( \begin{bmatrix} 0.342 & -0.9397 & 0 \\ 0.9397 & 0.342 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right)^T = [0.342 \ 0.9397 \ 0]$$

$$Twist_{right} = \begin{bmatrix} 0 & 0 & 1 & 3 & -6.5 & 0 \\ 0 & 0 & 0 & 0.342 & 0.9397 & 0 \end{bmatrix}$$

$$Twist = \text{empty matrix} \quad Wrench = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

### SLOT AT 89.9°



$$Twist_{right} = \begin{bmatrix} \omega & \mathbf{v} \\ \mathbf{0} & \mathbf{v} \end{bmatrix} \quad F_{right} = \begin{bmatrix} 0.00175 & -0.999 & 0 & 6.5 \\ 0.999 & 0.00175 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\omega = \left( \begin{bmatrix} 0.00175 & -0.999 & 0 \\ 0.999 & 0.00175 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right)^T = [0 \quad 0 \quad 1]$$

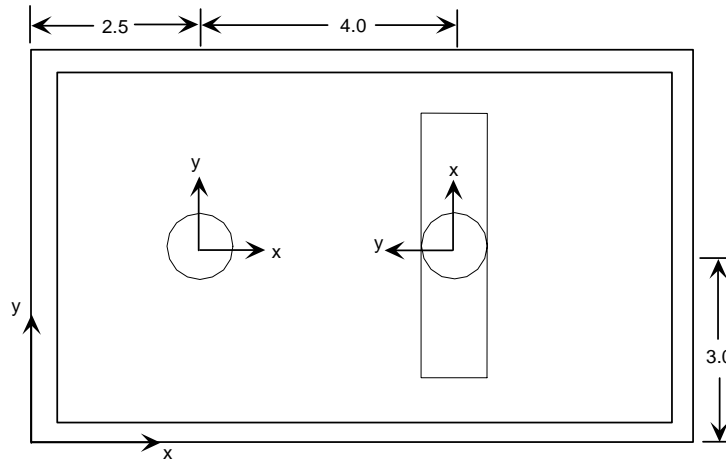
$$\mathbf{v}_{rot} = \begin{bmatrix} 6.5 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} = 3\mathbf{i} - 6.5\mathbf{j} = [3 \quad -6.5 \quad 0]$$

$$\mathbf{0} = [0 \quad 0 \quad 0] \quad \mathbf{v}_{trans} = \left( \begin{bmatrix} 0.00175 & -0.999 & 0 \\ 0.999 & 0.00175 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right)^T = [0.00175 \quad 0.999 \quad 0]$$

$$Twist_{right} = \begin{bmatrix} 0 & 0 & 1 & 3 & -6.5 & 0 \\ 0 & 0 & 0 & 0.00175 & 0.999 & 0 \end{bmatrix}$$

$$Twist = \text{empty matrix} \quad Wrench = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

### SLOT AT 90°



$$Twist_{right} = \begin{bmatrix} \boldsymbol{\omega} & \mathbf{v} \\ \mathbf{0} & \mathbf{v} \end{bmatrix} \quad \mathbf{F}_{right} = \begin{bmatrix} 0 & -1 & 0 & 6.5 \\ 1 & 0 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{\omega} = \left( \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right)^T = [0 \ 0 \ 1]$$

$$\mathbf{v}_{rot} = \begin{bmatrix} 6.5 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} = 3\mathbf{i} - 6.5\mathbf{j} = [3 \ -6.5 \ 0]$$

$$\mathbf{0} = [0 \ 0 \ 0] \quad \mathbf{v}_{trans} = \left( \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right)^T = [0 \ 1 \ 0]$$

$$Twist_{right} = \begin{bmatrix} 0 & 0 & 1 & 3 & -6.5 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$Twist = [0 \ 0 \ 1 \ 3 \ -2.5 \ 0] \quad Wrench = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & -3 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\text{rotation point} = \begin{bmatrix} \frac{2.5}{1} \\ \frac{3}{1} \\ \frac{1}{0} \end{bmatrix} = \begin{bmatrix} 2.5 \\ 3 \\ 0 \end{bmatrix}$$

$$\text{coordinate of overconstraint} = \begin{bmatrix} \frac{3}{1} \\ \frac{0}{1} \\ \frac{1}{0} \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}$$



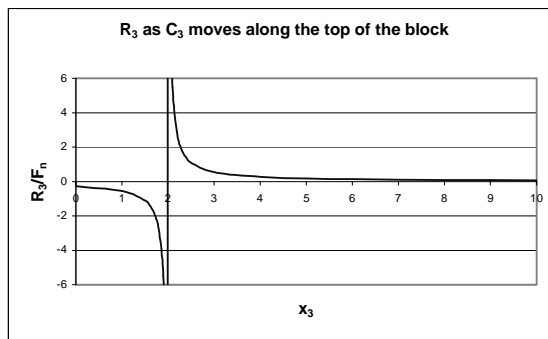


**APPENDIX B**

**EXCEL® ANALYSIS FOR EXAMPLES USING THE EQUATIONS OF EQUILIBRIUM**

***B.1 NO TWO CONSTRAINTS SHOULD BE CO-LINEAR ANALYSIS (FIG. 3.3)***

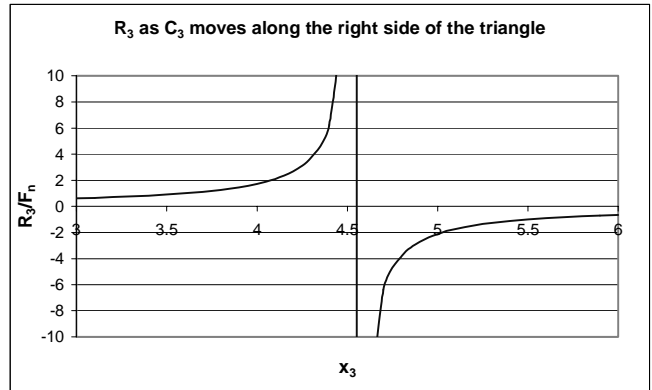
$F_n$	$x_3$	$R_3$
$x_1$	0	-0.27471098
$y_1$	1	-0.54942197
$x_2$	1.5	-1.09884394
$y_2$	1.6	-1.37355492
$x_3$	1.7	-1.83140656
$y_3$	1.8	-2.74710984
$x_n$	1.9	-5.49421969
$y_n$	1.95	-10.9884394
$x_{12}$	1.96	-13.7355492
$y_{12}$	1.97	-18.3140656
$0_{F_n}$	1.98	-27.4710984
	1.99	-54.9421969
	2	#DIV/0!
	2.01	54.9421969
	2.02	27.4710984
	2.03	18.3140656
	2.04	13.7355492
	2.05	10.9884394
	2.1	5.49421969
	2.2	2.74710984
	2.3	1.83140656
	2.4	1.37355492
	2.5	1.09884394
	3	0.54942197
	4	0.27471098
	5	0.18314066
	6	0.13735549
	7	0.10988439
	8	0.09157033
	9	0.07848885
	10	0.06867775





**B.2 NO THREE CONSTRAINTS SHOULD INTERSECT AT A POINT ANALYSIS  
(FIG. 3.10)**

x3	y3	R3
3	6	0.610843918
3.1	5.8	0.652913594
3.2	5.6	0.701206679
3.3	5.4	0.757214429
3.4	5.2	0.822945903
3.5	5	0.901174073
3.6	4.8	0.995837053
3.7	4.6	1.11272187
3.8	4.4	1.26069384
3.9	4.2	1.45405727
4	4	1.717482259
4.1	3.8	2.097471742
4.2	3.6	2.693375673
4.3	3.4	3.762253908
4.4	3.2	6.237721845
4.5	3	18.23760431
4.51	2.98	22.58180203
4.52	2.96	29.64269183
4.53	2.94	43.12796752
4.54	2.92	79.12335979
4.55	2.9	478.4288389
4.551	2.898	965.8632625
4.5511	2.8978	1075.430636
4.5512	2.8976	1213.037404
4.5519	2.8962	11628.6299
4.552	2.896	-51312.25259
4.553	2.894	-930.8211091
4.554	2.892	-469.6705435
4.555	2.89	-314.0719485
4.556	2.888	-235.9149584
4.557	2.886	-188.9056705
4.558	2.884	-157.5180424
4.559	2.882	-135.0747008
4.56	2.88	-118.2292666
4.57	2.86	-52.61368957
4.58	2.84	-33.83547674
4.59	2.82	-24.93573094
4.6	2.8	-19.74278579
4.7	2.6	-6.40473071
4.8	2.4	-3.822370384
4.9	2.2	-2.724046401
5	2	-2.116025404
5.1	1.8	-1.729902399
5.2	1.6	-1.462949764
5.3	1.4	-1.267373075
5.4	1.2	-1.117921959
5.5	1	-1
5.6	0.8	-0.904581835
5.7	0.6	-0.825786776
5.8	0.4	-0.759618943
5.9	0.2	-0.703268162
6	0	-0.654700538



$F_n = 1$   
 $x_{12} = 3$   
 $y_{12} = 2$   
 $x_3 = \text{variable}$   
 $y_3 = \text{variable}$   
 $x_n = 5.5$   
 $y_n = 1$   
 $\theta_{C_3} = 30$   
 $\theta_{F_n} = 30$

$x_{12}$  is the x-coordinate of the instant center (IC<sub>12</sub>)  
 $y_{12}$  is the y-coordinate of the instant center (IC<sub>12</sub>)

x3  
3  
=A12+0.1  
=A13+0.1  
=A14+0.1  
=A15+0.1  
=A16+0.1  
=A17+0.1  
=A18+0.1  
=A19+0.1  
=A20+0.1  
=A21+0.1  
=A22+0.1  
=A23+0.1  
=A24+0.1  
=A25+0.1  
=A26+0.1  
=A27+0.01  
=A28+0.01  
=A29+0.01  
=A30+0.01  
=A31+0.01  
=A32+0.001  
=A33+0.0001  
=A34+0.0001  
=A35+0.0007  
=A33+0.001  
=A37+0.001  
=A38+0.001  
=A39+0.001  
=A40+0.001  
=A41+0.001  
=A42+0.001  
=A43+0.001  
=A32+0.01  
=A45+0.01  
=A46+0.01  
=A47+0.01  
=A27+0.1  
=A49+0.1  
=A50+0.1  
=A51+0.1  
=A52+0.1  
=A53+0.1  
=A54+0.1  
=A55+0.1  
=A56+0.1  
=A57+0.1  
=A58+0.1  
=A59+0.1  
=A60+0.1  
=A61+0.1  
=A62+0.1

y3  
=-2\*A12+12  
=-2\*A13+12  
=-2\*A14+12  
=-2\*A15+12  
=-2\*A16+12  
=-2\*A17+12  
=-2\*A18+12  
=-2\*A19+12  
=-2\*A20+12  
=-2\*A21+12  
=-2\*A22+12  
=-2\*A23+12  
=-2\*A24+12  
=-2\*A25+12  
=-2\*A26+12  
=-2\*A27+12  
=-2\*A28+12  
=-2\*A29+12  
=-2\*A30+12  
=-2\*A31+12  
=-2\*A32+12  
=-2\*A33+12  
=-2\*A34+12  
=-2\*A35+12  
=-2\*A36+12  
=-2\*A37+12  
=-2\*A38+12  
=-2\*A39+12  
=-2\*A40+12  
=-2\*A41+12  
=-2\*A42+12  
=-2\*A43+12  
=-2\*A44+12  
=-2\*A45+12  
=-2\*A46+12  
=-2\*A47+12  
=-2\*A48+12  
=-2\*A49+12  
=-2\*A50+12  
=-2\*A51+12  
=-2\*A52+12  
=-2\*A53+12  
=-2\*A54+12  
=-2\*A55+12  
=-2\*A56+12  
=-2\*A57+12  
=-2\*A58+12  
=-2\*A59+12  
=-2\*A60+12  
=-2\*A61+12  
=-2\*A62+12  
=-2\*A63+12





LEFT SIDE

xn	yn	R3	R2	R1	R3	R2	R1
0	0	-1.463411332	-2.463411332	-1.4634113	FAILED	FAILED	FAILED
0.1	0.2	-1.362348303	-2.362348303	-1.3623483	FAILED	FAILED	FAILED
0.2	0.4	-1.261285274	-2.261285274	-1.2612853	FAILED	FAILED	FAILED
0.3	0.6	-1.160222245	-2.160222245	-1.1602222	FAILED	FAILED	FAILED
0.4	0.8	-1.059159216	-2.059159216	-1.0591592	FAILED	FAILED	FAILED
0.5	1	-0.958096187	-1.958096187	-0.9580962	FAILED	FAILED	FAILED
0.6	1.2	-0.857033158	-1.857033158	-0.8570332	FAILED	FAILED	FAILED
0.7	1.4	-0.755970129	-1.755970129	-0.7559701	FAILED	FAILED	FAILED
0.8	1.6	-0.6549071	-1.6549071	-0.6549071	FAILED	FAILED	FAILED
0.9	1.8	-0.553844071	-1.553844071	-0.5538441	FAILED	FAILED	FAILED
1	2	-0.452781042	-1.452781042	-0.452781	FAILED	FAILED	FAILED
1.1	2.2	-0.351718013	-1.351718013	-0.351718	FAILED	FAILED	FAILED
1.2	2.4	-0.250654984	-1.250654984	-0.250655	FAILED	FAILED	FAILED
1.3	2.6	-0.149591955	-1.149591955	-0.149592	FAILED	FAILED	FAILED
1.4	2.8	-0.048528926	-1.048528926	-0.0485289	FAILED	FAILED	FAILED
1.5	3	0.052534103	-0.947465897	0.0525341	ok	FAILED	ok
1.6	3.2	0.153597132	-0.846402868	0.15359713	ok	FAILED	ok
1.7	3.4	0.254660161	-0.745339839	0.25466016	ok	FAILED	ok
1.8	3.6	0.35572319	-0.64427681	0.35572319	ok	FAILED	ok
1.9	3.8	0.456786219	-0.543213781	0.45678622	ok	FAILED	ok
2	4	0.557849248	-0.442150752	0.55784925	ok	FAILED	ok
2.1	4.2	0.658912277	-0.341087723	0.65891228	ok	FAILED	ok
2.2	4.4	0.759975306	-0.240024694	0.75997531	ok	FAILED	ok
2.3	4.6	0.861038335	-0.138961665	0.86103834	ok	FAILED	ok
2.4	4.8	0.962101364	-0.037898636	0.96210136	ok	FAILED	ok
2.41	4.82	0.972207667	-0.027792333	0.97220767	ok	FAILED	ok
2.42	4.84	0.98231397	-0.01768603	0.98231397	ok	FAILED	ok
2.43	4.86	0.992420273	-0.007579727	0.99242027	ok	FAILED	ok
2.431	4.862	0.993430903	-0.006569097	0.9934309	ok	FAILED	ok
2.432	4.864	0.994441533	-0.005558467	0.99444153	ok	FAILED	ok
2.433	4.866	0.995452164	-0.004547836	0.99545216	ok	FAILED	ok
2.434	4.868	0.996462794	-0.003537206	0.99646279	ok	FAILED	ok
2.435	4.87	0.997473424	-0.002526576	0.99747342	ok	FAILED	ok
2.436	4.872	0.998484055	-0.001515945	0.99848405	ok	FAILED	ok
2.437	4.874	0.999494685	-0.000505315	0.99949468	ok	FAILED	ok
2.4371	4.8742	0.999595748	-0.000404252	0.99959575	ok	FAILED	ok
2.4372	4.8744	0.999696811	-0.000303189	0.99969681	ok	FAILED	ok
2.4373	4.8746	0.999797874	-0.000202126	0.99979787	ok	FAILED	ok
2.4374	4.8748	0.999898937	-0.000101063	0.99989894	ok	FAILED	ok
2.4375	4.875	1	6.40988E-16	1	ok	ok	ok
2.4376	4.8752	1.000101063	0.000101063	1.00010106	ok	ok	ok
2.4377	4.8754	1.000202126	0.000202126	1.00020213	ok	ok	ok
2.4378	4.8756	1.000303189	0.000303189	1.00030319	ok	ok	ok
2.4379	4.8758	1.000404252	0.000404252	1.00040425	ok	ok	ok
2.438	4.876	1.000505315	0.000505315	1.00050532	ok	ok	ok
2.44	4.88	1.002526576	0.002526576	1.00252658	ok	ok	ok
2.45	4.9	1.012632879	0.012632879	1.01263288	ok	ok	ok
2.5	5	1.063164393	0.063164393	1.06316439	ok	ok	ok
2.6	5.2	1.164227422	0.164227422	1.16422742	ok	ok	ok
2.7	5.4	1.265290451	0.265290451	1.26529045	ok	ok	ok
2.8	5.6	1.36635348	0.36635348	1.36635348	ok	ok	ok
2.9	5.8	1.467416509	0.467416509	1.46741651	ok	ok	ok
3	6	1.568479538	0.568479538	1.56847954	ok	ok	ok



BASE SIDE

xn	yn	R3	R2	R1	R3	R2	R1
6	0	-1.358343125	-1.358343125	-2.358343125	FAILED	FAILED	FAILED
5.9	0	-1.313065021	-1.313065021	-2.313065021	FAILED	FAILED	FAILED
5.8	0	-1.267786917	-1.267786917	-2.267786917	FAILED	FAILED	FAILED
5.7	0	-1.222508813	-1.222508813	-2.222508813	FAILED	FAILED	FAILED
5.6	0	-1.177230709	-1.177230709	-2.177230709	FAILED	FAILED	FAILED
5.5	0	-1.131952604	-1.131952604	-2.131952604	FAILED	FAILED	FAILED
5.4	0	-1.0866745	-1.0866745	-2.0866745	FAILED	FAILED	FAILED
5.3	0	-1.041396396	-1.041396396	-2.041396396	FAILED	FAILED	FAILED
5.2	0	-0.996118292	-0.996118292	-1.996118292	FAILED	FAILED	FAILED
5.1	0	-0.950840188	-0.950840188	-1.950840188	FAILED	FAILED	FAILED
5	0	-0.905562084	-0.905562084	-1.905562084	FAILED	FAILED	FAILED
4.9	0	-0.860283979	-0.860283979	-1.860283979	FAILED	FAILED	FAILED
4.8	0	-0.815005875	-0.815005875	-1.815005875	FAILED	FAILED	FAILED
4.7	0	-0.769727771	-0.769727771	-1.769727771	FAILED	FAILED	FAILED
4.6	0	-0.724449667	-0.724449667	-1.724449667	FAILED	FAILED	FAILED
4.5	0	-0.679171563	-0.679171563	-1.679171563	FAILED	FAILED	FAILED
4.4	0	-0.633893458	-0.633893458	-1.633893458	FAILED	FAILED	FAILED
4.3	0	-0.588615354	-0.588615354	-1.588615354	FAILED	FAILED	FAILED
4.2	0	-0.54333725	-0.54333725	-1.54333725	FAILED	FAILED	FAILED
4.1	0	-0.498059146	-0.498059146	-1.498059146	FAILED	FAILED	FAILED
4	0	-0.452781042	-0.452781042	-1.452781042	FAILED	FAILED	FAILED
3.9	0	-0.407502938	-0.407502938	-1.407502938	FAILED	FAILED	FAILED
3.8	0	-0.362224833	-0.362224833	-1.362224833	FAILED	FAILED	FAILED
3.7	0	-0.316946729	-0.316946729	-1.316946729	FAILED	FAILED	FAILED
3.6	0	-0.271668625	-0.271668625	-1.271668625	FAILED	FAILED	FAILED
3.5	0	-0.226390521	-0.226390521	-1.226390521	FAILED	FAILED	FAILED
3.4	0	-0.181112417	-0.181112417	-1.181112417	FAILED	FAILED	FAILED
3.3	0	-0.135834313	-0.135834313	-1.135834313	FAILED	FAILED	FAILED
3.2	0	-0.090556208	-0.090556208	-1.090556208	FAILED	FAILED	FAILED
3.1	0	-0.045278104	-0.045278104	-1.045278104	FAILED	FAILED	FAILED
3	0	-4.0215E-16	-4.0215E-16	-1	FAILED	FAILED	FAILED
2.9	0	0.045278104	0.045278104	-0.954721896	ok	ok	FAILED
2.8	0	0.090556208	0.090556208	-0.909443792	ok	ok	FAILED
2.7	0	0.135834313	0.135834313	-0.864165687	ok	ok	FAILED
2.6	0	0.181112417	0.181112417	-0.818887583	ok	ok	FAILED
2.5	0	0.226390521	0.226390521	-0.773609479	ok	ok	FAILED
2.4	0	0.271668625	0.271668625	-0.728331375	ok	ok	FAILED
2.3	0	0.316946729	0.316946729	-0.683053271	ok	ok	FAILED
2.2	0	0.362224833	0.362224833	-0.637775167	ok	ok	FAILED
2.1	0	0.407502938	0.407502938	-0.592497062	ok	ok	FAILED
2	0	0.452781042	0.452781042	-0.547218958	ok	ok	FAILED
1.9	0	0.498059146	0.498059146	-0.501940854	ok	ok	FAILED
1.8	0	0.54333725	0.54333725	-0.45666275	ok	ok	FAILED
1.7	0	0.588615354	0.588615354	-0.411384646	ok	ok	FAILED
1.6	0	0.633893458	0.633893458	-0.366106542	ok	ok	FAILED
1.5	0	0.679171563	0.679171563	-0.320828437	ok	ok	FAILED
1.4	0	0.724449667	0.724449667	-0.275550333	ok	ok	FAILED
1.3	0	0.769727771	0.769727771	-0.230272229	ok	ok	FAILED
1.2	0	0.815005875	0.815005875	-0.184994125	ok	ok	FAILED
1.1	0	0.860283979	0.860283979	-0.139716021	ok	ok	FAILED
1	0	0.905562084	0.905562084	-0.094437916	ok	ok	FAILED
0.9	0	0.950840188	0.950840188	-0.049159812	ok	ok	FAILED
0.8	0	0.996118292	0.996118292	-0.003881708	ok	ok	FAILED
0.799	0	0.996571073	0.996571073	-0.003428927	ok	ok	FAILED
0.798	0	0.997023854	0.997023854	-0.002976146	ok	ok	FAILED
0.797	0	0.997476635	0.997476635	-0.002523365	ok	ok	FAILED
0.796	0	0.997929416	0.997929416	-0.002070584	ok	ok	FAILED
0.795	0	0.998382197	0.998382197	-0.001617803	ok	ok	FAILED
0.794	0	0.998834978	0.998834978	-0.001165022	ok	ok	FAILED
0.793	0	0.999287759	0.999287759	-0.000712241	ok	ok	FAILED
0.792	0	0.99974054	0.99974054	-0.00025946	ok	ok	FAILED
0.7919	0	0.999785818	0.999785818	-0.000214182	ok	ok	FAILED
0.7918	0	0.999831096	0.999831096	-0.000168904	ok	ok	FAILED
0.7917	0	0.999876375	0.999876375	-0.000123625	ok	ok	FAILED
0.7916	0	0.999921653	0.999921653	-7.83474E-05	ok	ok	FAILED
0.7915	0	0.999966931	0.999966931	-3.30693E-05	ok	ok	FAILED
0.7914	0	1.000012209	1.000012209	1.22088E-05	ok	ok	ok
0.7913	0	1.000057487	1.000057487	5.74869E-05	ok	ok	ok
0.7912	0	1.000102765	1.000102765	0.000102765	ok	ok	ok
0.7911	0	1.000148043	1.000148043	0.000148043	ok	ok	ok
0.791	0	1.000193321	1.000193321	0.000193321	ok	ok	ok
0.79	0	1.000646102	1.000646102	0.000646102	ok	ok	ok
0.7	0	1.041396396	1.041396396	0.041396396	ok	ok	ok
0.6	0	1.0866745	1.0866745	0.0866745	ok	ok	ok
0.5	0	1.131952604	1.131952604	0.131952604	ok	ok	ok
0.4	0	1.177230709	1.177230709	0.177230709	ok	ok	ok
0.3	0	1.222508813	1.222508813	0.222508813	ok	ok	ok
0.2	0	1.267786917	1.267786917	0.267786917	ok	ok	ok
0.1	0	1.313065021	1.313065021	0.313065021	ok	ok	ok
0	0	1.358343125	1.358343125	0.358343125	ok	ok	ok

Example code from Excel<sup>®</sup>. This spreadsheet is actually set up columns A-E, where column A is xn, B is yn, C is R3, D is R2, and E is R1. It is simply shown here in this manner as an illustration of the formulas.

LEFT SIDE			
	xn	yn	R3
0	=A6*2		=(fn*COS(theta*PI()/180)*(y.12-B6)+fn*SIN(theta*PI()/180)*(x.12-A6))/(SIN(theta*PI()/180)*(x_3-x.12)-COS(theta*PI()/180)-R3*COS(theta*PI()/180))/-COS(theta*PI()/180)
			R2
			=(fn*COS(theta*PI()/180)-R3*COS(theta*PI()/180))/-COS(theta*PI()/180)
			R1
			=fn*SIN(theta*PI()/180)+D6*SIN(theta*PI()/180)+C6*SIN(theta*PI()/180)
=A6+0.1	=A13*2		R3
			=(fn*COS(theta*PI()/180)*(y.12-B13)+fn*SIN(theta*PI()/180)*(x.12-A13))/(SIN(theta*PI()/180)*(x_3-x.12)-COS(theta*PI()/180)-R2*COS(theta*PI()/180))/-COS(theta*PI()/180)
			R2
			=(fn*COS(theta*PI()/180)-C13*COS(theta*PI()/180))/-COS(theta*PI()/180)
			R1
			=fn*SIN(theta*PI()/180)+D13*SIN(theta*PI()/180)+C13*SIN(theta*PI()/180)
RIGHT SIDE			
	xn	yn	R3
3		6	=(fn*COS(theta*PI()/180)*(B22-y.12)-fn*SIN(theta*PI()/180)*(A22-x.12))/(SIN(theta*PI()/180)*(x_3-x.12)-COS(theta*PI()/180)-R2*COS(theta*PI()/180))/-COS(theta*PI()/180)
			R2
			=(fn*COS(theta*PI()/180)+C22*COS(theta*PI()/180))/COS(theta*PI()/180)
			R1
			=fn*SIN(theta*PI()/180)+D22*SIN(theta*PI()/180)+C22*SIN(theta*PI()/180)
=A105+0.1	=-2*A30+		R3
			=(fn*COS(theta*PI()/180)*(B30-y.12)-fn*SIN(theta*PI()/180)*(A30-x.12))/(SIN(theta*PI()/180)*(x_3-x.12)-COS(theta*PI()/180)-R2*COS(theta*PI()/180))/-COS(theta*PI()/180)
			R2
			=(fn*COS(theta*PI()/180)+C30*COS(theta*PI()/180))/COS(theta*PI()/180)
			R1
			=fn*SIN(theta*PI()/180)+D30*SIN(theta*PI()/180)+C30*SIN(theta*PI()/180)
BASE SIDE			
	xn	yn	R3
6		0	=(fn*(A39-x.12))/(SIN(theta*PI()/180)*(x_3-x.12)-COS(theta*PI()/180)*(y_3-y.12))
			R2
			=C39
			R1
			=(D39*SIN(theta*PI()/180)+C39*SIN(theta*PI()/180)-fn)
=A39-0.1		0	R3
			=(fn*(A47-x.12))/(SIN(theta*PI()/180)*(x_3-x.12)-COS(theta*PI()/180)*(y_3-y.12))
			R2
			=C47
			R1
			=(D47*SIN(theta*PI()/180)+C47*SIN(theta*PI()/180)-fn)



## **APPENDIX C      COMPARING THE GRAPHICAL NESTING FORCE WINDOW TO THE QUANTITATIVE NESTING FORCE WINDOW**

Object: To determine if the equations of equilibrium and the graphical method to find the nesting force window agree

### ***C.1 BRIEF DESCRIPTION AND BASELINE RESULTS***

First method—equations of equilibrium (Section C.2):

- Use the equations of equilibrium to find the reaction forces at each constraint, given a certain nesting force position.
- Find the transition points, which happen at the point when the reaction forces are all positive and then one or more reaction forces become negative.

Second method—graphical approach (Section C.3):

- Follow Blanding's rules for finding the nesting force window.
- Find the equation of a line for all sides of the assembly.
- Find the equation of a line for the constraint lines of action and the perpendicular intersections of the transition points.
- Find the intersections of the necessary lines to find the transition points.

**Transition points according to the first method (equations of equilibrium):**

Along the left side

$x_n$	$y_n$	$R_3$	$R_2$	$R_1$
2.4376	4.8752	1.000101063	0.000101063	1.000101

Along the right side

$x_n$	$y_n$	$R_3$	$R_2$	$R_1$
4.553	2.894	0.001029302	1.001029302	1.001029

Along the base

$x_n$	$y_n$	$R_3$	$R_2$	$R_1$
0.7914	0	1.000102765	1.000102765	0.000001

**Transition points according to the second method (graphical approach):**

Along the left side

$x_n$	$y_n$
2.40	4.80

Along the right side

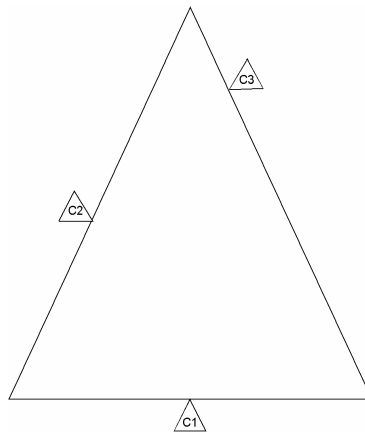
$x_n$	$y_n$
4.60	2.80

Along the base

$x_n$	$y_n$
0.75	0

They are essentially the same. The graphical method is limited by round-off error.

***C.2 FIRST METHOD: EQUATIONS OF EQUILIBRIUM***



Given the following information, the reaction forces at each constraint can be found:

$$\begin{aligned}
F_n &= 1 \\
\theta &= 30^\circ \\
x_{12} &= 3 \\
y_{12} &= 2 \\
x_3 &= 3.5625 \\
y_3 &= 4.875
\end{aligned}$$

Find  $R_3$ :

+ ↻  $\Sigma M_{12}=0$  counterclockwise positive

Along the left side of the triangle:

$$R_3 = \frac{F_n * \cos(\theta) * (y_{12} - y_n) + F_n * \sin(\theta) * (x_{12} - x_n)}{\sin(\theta) * (x_3 - x_{12}) - \cos(\theta) * (y_3 - y_{12})}$$

Along the right side of the triangle:

$$R_3 = \frac{F_n * \cos(\theta) * (y_n - y_{12}) - F_n * \sin(\theta) * (x_n - x_{12})}{\sin(\theta) * (x_3 - x_{12}) - \cos(\theta) * (y_3 - y_{12})}$$

Along the bottom of the triangle:

$$R_3 = \frac{F_n * \cos(\theta) * (x_n - x_{12})}{\sin(\theta) * (x_3 - x_{12}) - \cos(\theta) * (y_3 - y_{12})}$$

Find  $R_2$ :

+ →  $\Sigma F_x=0$

Along the left side of the triangle:

$$R_2 = \frac{F_n * \cos(\theta) - R_3 * \cos(\theta)}{-\cos(\theta)}$$

Along the right side of the triangle:

$$R_2 = \frac{F_n * \cos(\theta) + R_3 * \cos(\theta)}{\cos(\theta)}$$

Along the bottom of the triangle:

$$R_2 = R_3$$

Find  $R_1$ :

$$+\uparrow \Sigma F_y = 0$$

Along the left side of the triangle:

$$R_1 = F_n * \sin(\theta) + R_2 * \sin(\theta) + R_3 * \sin(\theta)$$

Along the right side of the triangle:

$$R_1 = F_n * \sin(\theta) + R_2 * \sin(\theta) + R_3 * \sin(\theta)$$

Along the bottom of the triangle:

$$R_1 = R_2 * \sin(\theta) + R_3 * \sin(\theta) - F_n$$

Overall results (in numbers):

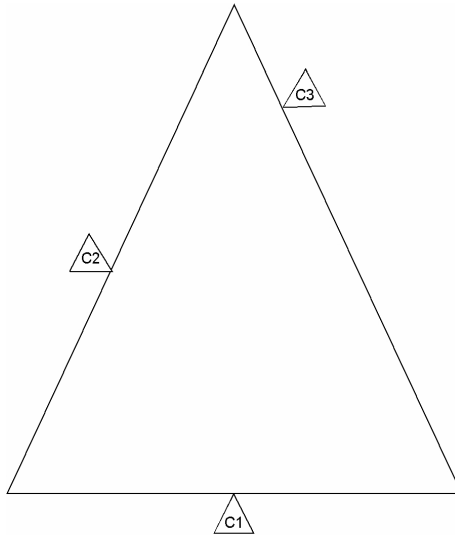
	xn	yn	R3	R2	R1
0	0	-1.463411332	-2.463411332	-1.463411	Left side of the triangle
	0.1	0.2	-1.362348303	-2.362348303	-1.362348
	0.2	0.4	-1.261285274	-2.261285274	-1.261285
	0.3	0.6	-1.160222245	-2.160222245	-1.160222
	0.4	0.8	-1.059159216	-2.059159216	-1.059159
	0.5	1	-0.958096187	-1.958096187	-0.958096
	0.6	1.2	-0.857033158	-1.857033158	-0.857033
	0.7	1.4	-0.755970129	-1.755970129	-0.75597
	0.8	1.6	-0.6549071	-1.6549071	-0.654907
	0.9	1.8	-0.553844071	-1.553844071	-0.553844
	1	2	-0.452781042	-1.452781042	-0.452781
	1.1	2.2	-0.351718013	-1.351718013	-0.351718
	1.2	2.4	-0.250654984	-1.250654984	-0.250655
	1.3	2.6	-0.149591955	-1.149591955	-0.149592
	1.4	2.8	-0.048528926	-1.048528926	-0.048529
	1.5	3	0.052534103	-0.947465897	0.0525341
	1.6	3.2	0.153597132	-0.846402868	0.1535971
	1.7	3.4	0.254660161	-0.745339839	0.2546602
	1.8	3.6	0.35572319	-0.64427681	0.3557232
	1.9	3.8	0.456786219	-0.543213781	0.4567862
	2	4	0.557849248	-0.442150752	0.5578492
	2.1	4.2	0.658912277	-0.341087723	0.6589123
	2.2	4.4	0.759975306	-0.240024694	0.7599753
	2.3	4.6	0.861038335	-0.138961665	0.8610383
	2.4	4.8	0.962101364	-0.037898636	0.9621014
	2.41	4.82	0.972207667	-0.027792333	0.9722077
	2.42	4.84	0.98231397	-0.01768603	0.982314
	2.43	4.86	0.992420273	-0.007579727	0.9924203
	2.437	4.874	0.999494685	-0.000505315	0.9994947
	2.4371	4.8742	0.999595748	-0.000404252	0.9995957

	2.4374	4.8748	0.999898937	-0.000101063	0.9998989
	2.4375	4.875	1	6.40988E-16	1
2.4376	4.8752	1.000101063	0.000101063	1.0001011	Approximate transition point
	2.4377	4.8754	1.000202126	0.000202126	1.0002021
	2.4378	4.8756	1.000303189	0.000303189	1.0003032
	2.4379	4.8758	1.000404252	0.000404252	1.0004043
	2.438	4.876	1.000505315	0.000505315	1.0005053
	2.44	4.88	1.002526576	0.002526576	1.0025266
	2.45	4.9	1.012632879	0.012632879	1.0126329
	2.5	5	1.063164393	0.063164393	1.0631644
	2.6	5.2	1.164227422	0.164227422	1.1642274
	2.7	5.4	1.265290451	0.265290451	1.2652905
	2.8	5.6	1.36635348	0.36635348	1.3663535
	2.9	5.8	1.467416509	0.467416509	1.4674165
	3	6	1.568479538	0.568479538	1.5684795
3	6	-1.568479538	-0.568479538	-0.56848	Right side of the triangle
	3.1	5.8	-1.467416509	-0.467416509	-0.467417
	3.2	5.6	-1.36635348	-0.36635348	-0.366353
	3.3	5.4	-1.265290451	-0.265290451	-0.26529
	3.4	5.2	-1.164227422	-0.164227422	-0.164227
	3.5	5	-1.063164393	-0.063164393	-0.063164
	3.6	4.8	-0.962101364	0.037898636	0.0378986
	3.7	4.6	-0.861038335	0.138961665	0.1389617
	3.8	4.4	-0.759975306	0.240024694	0.2400247
	3.9	4.2	-0.658912277	0.341087723	0.3410877
	4	4	-0.557849248	0.442150752	0.4421508
	4.1	3.8	-0.456786219	0.543213781	0.5432138
	4.2	3.6	-0.35572319	0.64427681	0.6442768
	4.3	3.4	-0.254660161	0.745339839	0.7453398
	4.4	3.2	-0.153597132	0.846402868	0.8464029
	4.5	3	-0.052534103	0.947465897	0.9474659
	4.51	2.98	-0.0424278	0.9575722	0.9575722
	4.52	2.96	-0.032321497	0.967678503	0.9676785
	4.53	2.94	-0.022215194	0.977784806	0.9777848
	4.54	2.92	-0.012108892	0.987891108	0.9878911
	4.55	2.9	-0.002002589	0.997997411	0.9979974
	4.551	2.898	-0.000991958	0.999008042	0.999008
	4.552	2.896	1.86719E-05	1.000018672	1.0000187
4.553	2.894	0.001029302	1.001029302	1.0010293	Approximate transition point
	4.554	2.892	0.002039932	1.002039932	1.0020399
	4.555	2.89	0.003050563	1.003050563	1.0030506
	4.56	2.88	0.008103714	1.008103714	1.0081037
	4.6	2.8	0.048528926	1.048528926	1.0485289
	4.7	2.6	0.149591955	1.149591955	1.149592
	4.8	2.4	0.250654984	1.250654984	1.250655
	4.9	2.2	0.351718013	1.351718013	1.351718
	5	2	0.452781042	1.452781042	1.452781
	5.1	1.8	0.553844071	1.553844071	1.5538441
	5.2	1.6	0.6549071	1.6549071	1.6549071
	5.3	1.4	0.755970129	1.755970129	1.7559701
	5.4	1.2	0.857033158	1.857033158	1.8570332
	5.5	1	0.958096187	1.958096187	1.9580962
	5.6	0.8	1.059159216	2.059159216	2.0591592
	5.7	0.6	1.160222245	2.160222245	2.1602222
	5.8	0.4	1.261285274	2.261285274	2.2612853
	5.9	0.2	1.362348303	2.362348303	2.3623483
	6	0	1.463411332	2.463411332	2.4634113
6	0	-1.358343125	-1.358343125	-2.358343	Base of the triangle
	5.9	0	-1.313065021	-1.313065021	-2.313065
	5.8	0	-1.267786917	-1.267786917	-2.267787
	5.7	0	-1.222508813	-1.222508813	-2.222509
	5.6	0	-1.177230709	-1.177230709	-2.177231
	5.5	0	-1.131952604	-1.131952604	-2.131953
	5.4	0	-1.0866745	-1.0866745	-2.086675
	5.3	0	-1.041396396	-1.041396396	-2.041396
	5.2	0	-0.996118292	-0.996118292	-1.996118
	5.1	0	-0.950840188	-0.950840188	-1.95084
	5	0	-0.905562084	-0.905562084	-1.905562
	4.9	0	-0.860283979	-0.860283979	-1.860284
	4.8	0	-0.815005875	-0.815005875	-1.815006
	4.7	0	-0.769727771	-0.769727771	-1.769728
	4.6	0	-0.724449667	-0.724449667	-1.72445



4.5	0	-0.679171563	-0.679171563	-1.679172
4.4	0	-0.633893458	-0.633893458	-1.633893
4.3	0	-0.588615354	-0.588615354	-1.588615
4.2	0	-0.54333725	-0.54333725	-1.543337
4.1	0	-0.498059146	-0.498059146	-1.498059
4	0	-0.452781042	-0.452781042	-1.452781
3.9	0	-0.407502938	-0.407502938	-1.407503
3.8	0	-0.362224833	-0.362224833	-1.362225
3.7	0	-0.316946729	-0.316946729	-1.316947
3.6	0	-0.271668625	-0.271668625	-1.271669
3.5	0	-0.226390521	-0.226390521	-1.226391
3.4	0	-0.181112417	-0.181112417	-1.181112
3.3	0	-0.135834313	-0.135834313	-1.135834
3.2	0	-0.090556208	-0.090556208	-1.090556
3.1	0	-0.045278104	-0.045278104	-1.045278
3	0	-4.0215E-16	-4.0215E-16	-1
2.9	0	0.045278104	0.045278104	-0.954722
2.8	0	0.090556208	0.090556208	-0.909444
2.7	0	0.135834313	0.135834313	-0.864166
2.6	0	0.181112417	0.181112417	-0.818888
2.5	0	0.226390521	0.226390521	-0.773609
2.4	0	0.271668625	0.271668625	-0.728331
2.3	0	0.316946729	0.316946729	-0.683053
2.2	0	0.362224833	0.362224833	-0.637775
2.1	0	0.407502938	0.407502938	-0.592497
2	0	0.452781042	0.452781042	-0.547219
1.9	0	0.498059146	0.498059146	-0.501941
1.8	0	0.54333725	0.54333725	-0.456663
1.7	0	0.588615354	0.588615354	-0.411385
1.6	0	0.633893458	0.633893458	-0.366107
1.5	0	0.679171563	0.679171563	-0.320828
1.4	0	0.724449667	0.724449667	-0.27555
1.3	0	0.769727771	0.769727771	-0.230272
1.2	0	0.815005875	0.815005875	-0.184994
1.1	0	0.860283979	0.860283979	-0.139716
1	0	0.905562084	0.905562084	-0.094438
0.9	0	0.950840188	0.950840188	-0.04916
0.8	0	0.996118292	0.996118292	-0.003882
0.799	0	0.996571073	0.996571073	-0.003429
0.798	0	0.997023854	0.997023854	-0.002976
0.796	0	0.997929416	0.997929416	-0.002071
0.7915	0	0.999966931	0.999966931	-3.31E-05
0.7914	0	1.000012209	1.000012209	1.221E-05
0.7913	0	1.000057487	1.000057487	5.749E-05
0.7912	0	1.000102765	1.000102765	0.0001028
				Approximate transition point
0.7911	0	1.000148043	1.000148043	0.000148
0.791	0	1.000193321	1.000193321	0.0001933
0.79	0	1.000646102	1.000646102	0.0006461
0.7	0	1.041396396	1.041396396	0.0413964
0.6	0	1.0866745	1.0866745	0.0866745
0.5	0	1.131952604	1.131952604	0.1319526
0.4	0	1.177230709	1.177230709	0.1772307
0.3	0	1.222508813	1.222508813	0.2225088
0.2	0	1.267786917	1.267786917	0.2677869
0.1	0	1.313065021	1.313065021	0.313065
0	0	1.358343125	1.358343125	0.3583431

**C.3 SECOND METHOD: EQUATION OF A LINE TO FIND INTERSECTION/TRANSITION POINTS**



For the given triangle, the equation of a line is found for each side.

Left side:

$$y = 2x$$

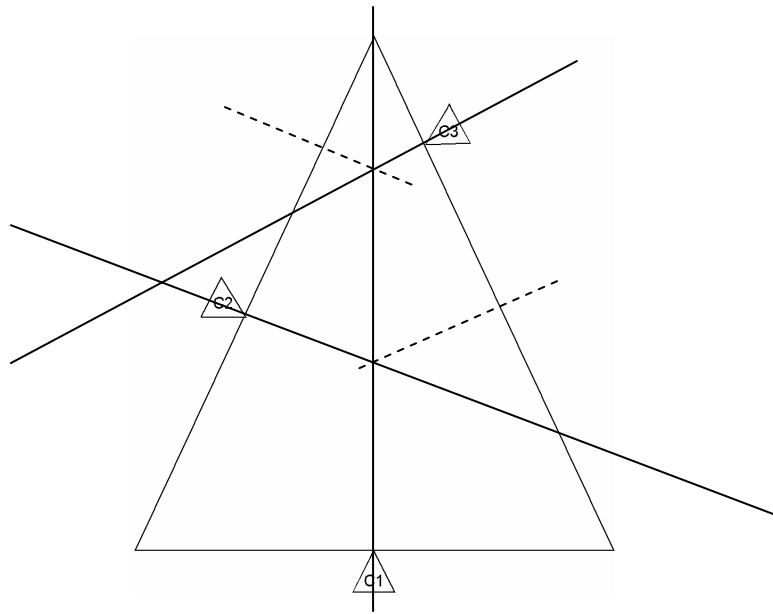
Right side:

$$y = -2x + 12$$

Along the bottom:

$$y = 0$$

Next, the intersection points where the transition points lie must be found. This can be done by constructing a line perpendicular to each side of the triangle that runs through the necessary instant center (dashed lines).



Perpendicular (dashed) line for the transition point on the left side of the triangle:

$$y = -\frac{1}{2}x + 6$$

To find the transition point, set the two equations equal:

$$\begin{aligned} 2 * x &= -\frac{1}{2}x + 6 \\ x &= 2.4 \\ y &= 4.8 \end{aligned}$$

Perpendicular (dashed) line for the transition point on the right side of the triangle:

$$y = \frac{1}{2}x + 0.5$$

To find the transition point, set the two equations equal:

$$\begin{aligned} -2 * x + 12 &= \frac{1}{2}x + 0.5 \\ x &= 4.6 \\ y &= 2.8 \end{aligned}$$

To find the transition point along the bottom of the assembly, take the equation of a line for the lines of action on constraints 2 and 3, and set them equal to each other (solid lines).

Equation of a line for the line of action extending from the top right constraint:

$$y = \frac{2}{3}x + 2.5$$

Equation of a line for the line of action extending from the left side constraint:

$$y = -\frac{4}{9}x + 3.333$$

To find the intersection point, set the two equations equal:

$$\frac{2}{3}x + 2.5 = -\frac{4}{9}x + 3.333$$

$$x = 0.75$$

$$y = 3$$

The transition point is found by simply projecting the x-value onto the x-axis:

$$x = 0.75$$

$$y = 0$$



## APPENDIX D      DETAILED ANALYSIS USING THE EQUATIONS OF EQUILIBRIUM FOR FIGURE 4.19

This appendix first shows the set-up in Excel<sup>®</sup> for the equations of equilibrium.

Afterwards, each of the examples in Fig. 4.19 will be solved.

### *D.1 FORMULAS USED IN EXCEL<sup>®</sup> FOR FIG. 4.19*

	C		r=
Vertical force only			
0	0	=-COS(phi*PI()/180)*(x_3-x_12)-SIN(phi*PI()/180)*(y_3-y_12)	R1
=-SIN(phi*PI()/180)	=COS(phi*PI()/180)	=SIN(phi*PI()/180)	R2
=COS(phi*PI()/180)	=SIN(phi*PI()/180)	=-COS(phi*PI()/180)	R3
=MINVERSE(M72:O74)	=MINVERSE(M72:O74)	=MINVERSE(M72:O74)	
=MINVERSE(M72:O74)	=MINVERSE(M72:O74)	=MINVERSE(M72:O74)	
=MINVERSE(M72:O74)	=MINVERSE(M72:O74)	=MINVERSE(M72:O74)	
Add the horizontal force			
0	0	=-COS(phi*PI()/180)*(x_3-x_12)-SIN(phi*PI()/180)*(y_3-y_12)	R1
=-SIN(phi*PI()/180)	=COS(phi*PI()/180)	=SIN(phi*PI()/180)	R2
=COS(phi*PI()/180)	=SIN(phi*PI()/180)	=-COS(phi*PI()/180)	R3
=MINVERSE(M82:O84)	=MINVERSE(M82:O84)	=MINVERSE(M82:O84)	
=MINVERSE(M82:O84)	=MINVERSE(M82:O84)	=MINVERSE(M82:O84)	
=MINVERSE(M82:O84)	=MINVERSE(M82:O84)	=MINVERSE(M82:O84)	

b

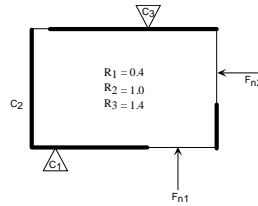
$$\begin{aligned} &= -F_{n1} \cdot \cos(\phi) \cdot (x_{n1} - x_{n2}) + F_{n1} \cdot \sin(\phi) \cdot (y_{n2} - y_{n1}) \\ &= F_{n1} \cdot \sin(\phi) \\ &= -F_{n1} \cdot \cos(\phi) \end{aligned}$$

$$\begin{aligned} &= \text{MMULT}(M76:O78, S72:S74) \\ &= \text{MMULT}(M76:O78, S72:S74) \\ &= \text{MMULT}(M76:O78, S72:S74) \end{aligned}$$

$$\begin{aligned} &= -F_{n1} \cdot \cos(\phi) \cdot (x_{n1} - x_{n2}) + F_{n1} \cdot \sin(\phi) \cdot (y_{n2} - y_{n1}) - F_{n2} \cdot \cos(\phi) \cdot (y_{n2} - y_{n1}) + F_{n2} \cdot \sin(\phi) \cdot (x_{n2} - x_{n1}) \\ &= F_{n1} \cdot \sin(\phi) + F_{n2} \cdot \cos(\phi) \\ &= -F_{n1} \cdot \cos(\phi) + F_{n2} \cdot \sin(\phi) \end{aligned}$$

$$\begin{aligned} &= \text{MMULT}(M86:O88, S82:S84) \\ &= \text{MMULT}(M86:O88, S82:S84) \\ &= \text{MMULT}(M86:O88, S82:S84) \end{aligned}$$

## D.2 THE RESULTS FOR FIG. 4.19



### Force Analysis

$x_{1} = 1$   
 $y_{1} = 0$   
 $x_{2} = 0$   
 $y_{2} = 2.5$   
 $x_{3} = 6$   
 $y_{3} = 6.6667$

$F_{n1} = 1$   
 $F_{n2} = 1$   
 $x_{n1} = 7$   
 $y_{n1} = -4.64E-16$   
 $x_{n2} = 10$   
 $y_{n2} = 3.5$   
 $x_{12} = 1$   
 $y_{12} = 2.5$

Vertical force only (Nesting force is along the base of the block)

$$\begin{vmatrix} 0 & 0 & -5 \\ 7.733E-17 & 1 & -7.73E-17 \\ 1 & -7.73E-17 & -1 \end{vmatrix} \begin{vmatrix} -6 \\ -7.7E-17 \\ -1 \end{vmatrix}$$

$$\begin{vmatrix} -0.2 & 7.73E-17 & 1 \\ 0 & 1 & -7.73E-17 \\ -0.2 & 0 & 0 \end{vmatrix} \begin{vmatrix} 0.2 \\ 0 \\ 1.2 \end{vmatrix}$$

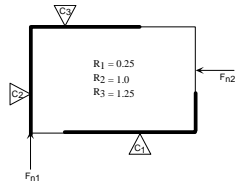
Add on the horizontal force (Vertical nesting force along the base, horizontal along the right)

$$\begin{vmatrix} 0 & 0 & -5 \\ 7.733E-17 & 1 & -7.73E-17 \\ 1 & -7.73E-17 & -1 \end{vmatrix} \begin{vmatrix} -7 \\ 1 \\ -1 \end{vmatrix}$$

$$\begin{vmatrix} -0.2 & 7.73E-17 & 1 \\ 0 & 1 & -7.73E-17 \\ -0.2 & 0 & 0 \end{vmatrix} \begin{vmatrix} 0.4 \\ 1 \\ 1.4 \end{vmatrix} \begin{matrix} R1 \\ R2 \\ R3 \end{matrix}$$

Transition points:

$x_{n1\_top} = 1$   
 $x_{n1\_base} = 6$   
 $y_{n2\_right} = 2.5$



**Force Analysis**

x\_1= 6  
 y\_1= 0  
 x\_2= 0  
 y\_2= 2.5  
 x\_3= 2  
 y\_3= 6.6667

Vertical force only (Nesting force is along the base of the block)

$$\begin{bmatrix} 0 & 0 & 4 \\ 7.733E-17 & 1 & -7.73E-17 \\ 1 & -7.73E-17 & -1 \end{bmatrix} \begin{bmatrix} 6 \\ -7.7E-17 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} 0.25 & 7.73E-17 & 1 \\ 0 & 1 & -7.73E-17 \\ 0.25 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0 \\ 1.5 \end{bmatrix}$$

Fn\_1= 1  
 Fn\_2= 1  
 xn\_1= 0  
 yn\_1= 7.73E-17  
 xn\_2= 15  
 yn\_2= 3.5  
 x\_12= 6  
 y\_12= 2.5

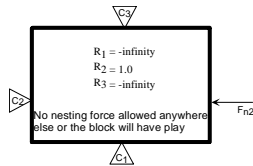
Add on the horizontal force (Vertical nesting force along the base, horizontal along the right)

$$\begin{bmatrix} 0 & 0 & 4 \\ 7.733E-17 & 1 & -7.73E-17 \\ 1 & -7.73E-17 & -1 \end{bmatrix} \begin{bmatrix} 5 \\ 1 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} 0.25 & 7.73E-17 & 1 \\ 0 & 1 & -7.73E-17 \\ 0.25 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.25 \\ 1 \\ 1.25 \end{bmatrix} \begin{matrix} R1 \\ R2 \\ R3 \end{matrix}$$

Transition points:

Xn1\_top = 6  
 Xn1\_base = 2  
 Yn2\_right = 2.5



**Force Analysis**

x\_1= 5  
 y\_1= 0  
 x\_2= 0  
 y\_2= 2.5  
 x\_3= 5  
 y\_3= 6.6667

Vertical force only (Nesting force is along the base of the block)

$$\begin{bmatrix} 0 & 0 & 3.22E-16 \\ 7.733E-17 & 1 & -7.73E-17 \\ 1 & -7.73E-17 & -1 \end{bmatrix} \begin{bmatrix} 5 \\ -7.7E-17 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} 3.104E+15 & 7.73E-17 & 1 \\ 0 & 1 & -7.73E-17 \\ 3.104E+15 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1.55E+16 \\ 0 \\ 1.55E+16 \end{bmatrix}$$

Fn\_1= 1  
 Fn\_2= 1  
 xn\_1= 0  
 yn\_1= 7.73E-17  
 xn\_2= 14  
 yn\_2= 2.5  
 x\_12= 5  
 y\_12= 2.5

Add on the horizontal force (Vertical nesting force along the base, horizontal along the right)

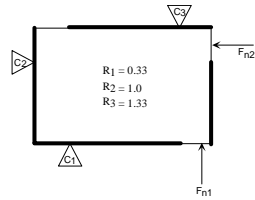
$$\begin{bmatrix} 0 & 0 & 3.22E-16 \\ 7.733E-17 & 1 & -7.73E-17 \\ 1 & -7.73E-17 & -1 \end{bmatrix} \begin{bmatrix} 5 \\ 1 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} 3.104E+15 & 7.73E-17 & 1 \\ 0 & 1 & -7.73E-17 \\ 3.104E+15 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1.55E+16 \\ 1 \\ 1.55E+16 \end{bmatrix} \begin{matrix} R1 \\ R2 \\ R3 \end{matrix}$$

Transition points:

Xn1\_top = ---  
 Xn1\_base = ---  
 Yn2\_right = 2.5





**Force Analysis**

x\_1= 2  
y\_1= 0  
  
x\_2= 0  
y\_2= 4.5  
  
x\_3= 8  
y\_3= 6.6667

Fn\_1= 1  
Fn\_2= 1  
xn\_1= 9.5  
yn\_1= -6.57E-16  
xn\_2= 11  
yn\_2= 5  
x\_12= 2  
y\_12= 4.5

Vertical force only (Nesting force is along the base of the block)

$$\begin{bmatrix} 0 & 0 & -6 \\ 7.733E-17 & 1 & -7.73E-17 \\ 1 & -7.73E-17 & -1 \end{bmatrix} \begin{bmatrix} -7.5 \\ -7.7E-17 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} -0.166667 & 7.73E-17 & 1 \\ 0 & 1 & -7.73E-17 \\ -0.166667 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.25 \\ 0 \\ 1.25 \end{bmatrix}$$

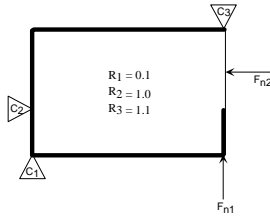
Add on the horizontal force (Vertical nesting force along the base, horizontal along the right)

$$\begin{bmatrix} 0 & 0 & -6 \\ 7.733E-17 & 1 & -7.73E-17 \\ 1 & -7.73E-17 & -1 \end{bmatrix} \begin{bmatrix} -8 \\ 1 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} -0.166667 & 7.73E-17 & 1 \\ 0 & 1 & -7.73E-17 \\ -0.166667 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.333333 \\ 1 \\ 1.333333 \end{bmatrix} \begin{matrix} R1 \\ R2 \\ R3 \end{matrix}$$

Transition points:

Xn1\_top = 2  
Xn1\_base = 8  
Yn2\_right = 4.5



**Force Analysis**

x\_1= 0  
y\_1= 0  
  
x\_2= 0  
y\_2= 2.5  
  
x\_3= 10  
y\_3= 6.6667

Fn\_1= 1  
Fn\_2= 1  
xn\_1= 10  
yn\_1= -6.96E-16  
xn\_2= 9  
yn\_2= 3.5  
x\_12= 7.73E-17  
y\_12= 2.5

Vertical force only (Nesting force is along the base of the block)

$$\begin{bmatrix} 0 & 0 & -10 \\ 7.733E-17 & 1 & -7.73E-17 \\ 1 & -7.73E-17 & -1 \end{bmatrix} \begin{bmatrix} -10 \\ -7.7E-17 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} -0.1 & 7.73E-17 & 1 \\ 0 & 1 & -7.73E-17 \\ -0.1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Add on the horizontal force (Vertical nesting force along the base, horizontal along the right)

$$\begin{bmatrix} 0 & 0 & -10 \\ 7.733E-17 & 1 & -7.73E-17 \\ 1 & -7.73E-17 & -1 \end{bmatrix} \begin{bmatrix} -11 \\ 1 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} -0.1 & 7.73E-17 & 1 \\ 0 & 1 & -7.73E-17 \\ -0.1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.1 \\ 1 \\ 1.1 \end{bmatrix} \begin{matrix} R1 \\ R2 \\ R3 \end{matrix}$$

Transition points:

Xn1\_top = 0  
Xn1\_base = 10  
Yn2\_right = 2.5

## **APPENDIX E            DETAILS FOR THE MONTE CARLO SIMULATION EXAMPLES IN CHAPTER 6**

This appendix contains the detailed analysis and programs used to find the results for the Monte Carlo simulations performed in Chapter 6. Each section contains the C program developed for the example and the Excel<sup>®</sup> spreadsheet (if one exists) used to verify the results. Further description for the development of the slotted block example is also presented.

### ***E.1 ALL BLOCKS WITH THREE CONSTRAINTS (SECTIONS 6.3.1 AND 6.3.2)***

The C program shown below allows for all three constraints to be varied in both directions. The starting points are changed for each set-up presented.

```
#include <math.h>
#include <stdio.h>
#include "LUD.h"

#define EPS 0.000001
#define PERTURB 0.000001
#define PI 3.14159265

void func(double d[], double f[]);
double resid(double f[], int nVar);
double rad(double deg);
double nr(double angle);

double x1, z1, x2, y2, x3, y3;
double xp, yp, h, w;
int nfail;

float ran1(int *idum);
float gasdev(int *idum);
```

```

/*-----
                        MONTE CARLO
-----*/
main ()
{
    double error1, error;
    int iSeed, i, nloop;

    iSeed = -5;
    nfail = 0;
    error = 0.;
    printf ("Enter number of loops\n");
    scanf ("%d", &nloop);

    for (i=1; i<=nloop; i++)
    {
        /* generate random deviates from the starting points */

        x1 = gasdev(&iSeed) * 0.2 + 2.0;
        z1 = gasdev(&iSeed) * 0.2 + 0.0;
        x2 = gasdev(&iSeed) * 0.2 + 0.0;
        y2 = gasdev(&iSeed) * 0.2 + 3.35;
        x3 = gasdev(&iSeed) * 0.2 + 8.0;
        y3 = gasdev(&iSeed) * 0.2 + 6.667;

        printf ("x1,y1:%lf %lf\nx2,y2:%lf %lf\nx3,y3:%lf %lf\n", x1,
z1, x2, y2, x3, y3);

        error1 = nr(0.);
        error += fabs(error1);
    }

    /* output scalar AF values */

    printf ("nfail = %d\n", nfail);
    printf ("error average = %lf\n", error/(nloop-nfail));
}

/*-----
                        NR routine
-----*/

double nr(double angle)
{
    int i, j, nRow, nCol, nB, nVar, count, pvt[MAX_ROWS];
    double a[MAX_ROWS][MAX_COLS], b[MAX_ROWS], det;
    double d[MAX_ROWS], f1[MAX_ROWS], f[MAX_ROWS], totalResid;
    double error, xd1, xd2, yd1, yd2;

    /* initialize x */

```

```

nVar = 4;
d[0] = angle;
d[1] = y2;
d[2] = x1;
d[3] = x3;

/* call functions */
func(d,f);

/* compute residuals */
totalResid = resid(f,nVar);

/* enter main loop */
count = 0;

while (fabs(totalResid) > EPS)
{
    count++;

    /* Evaluate the Jacobian */
    for (i=0; i<nVar; i++)
    {
        /* perturb x */
        d[i] = d[i] + PERTURB;
        func(d, f1);

        for (j = 0; j<nVar; j++)
        {
            a[j][i] = (f1[j]-f[j]) / PERTURB;
        }
        d[i] = d[i] - PERTURB;
    }

    /* printf ("matrix a:\n");
    for (i=0; i<nVar; i++)
    {
        for (j=0; j<nVar; j++)
        {
            printf("%lf ", a[i][j]);
        }
        printf("\n");
    }
    printf("\n"); */

    /* Make sure the functions are current */
    func(d, f);

    /* load b vector */
    for (i=0; i<nVar; i++)
    {
        b[i] = -f[i];
    }
}

```

```

/*printf ("vector b:\n");
for (i=0; i<nVar; i++)
{
    printf("%lf ", b[i]);
}
printf("\n"); */

/* call LU DECOMPOSITION routine */
det = LUdecomp(a, nVar, pvt);
LUSolve(a, nVar, b, pvt);

/* printf ("matrix a:\n");
for (i=0; i<nVar; i++)
{
    for (j=0; j<nVar; j++)
    {
        printf("%lf ", a[i][j]);
    }
}
printf("\n");
} */

/* compute new value for x */
for (i=0; i<nVar; i++)
{
    d[i] = d[i] + b[i];
}

/*printf ("new d:\n");
for (i=0; i<nVar; i++)
{
    printf("%lf ", d[i]);
}
printf("\n"); */

/* evaluate the function's residuals */
func(d, f);
totalResid = resid(f, nVar);
if (count > 15)
{
    nfail +=1;
    printf("Failure in NR\n");
    break;
}
}

/*-----
                                EVALUATE ERROR
-----*/
if (count <= 15)
{

```

```

        xd1=x2 + sin(rad(d[0]))*d[1];
        yd1=y2 - cos(rad(d[0]))*d[1];
        xd2=xd1 + w*cos(rad(d[0]));
        yd2=yd1 + w*sin(rad(d[0]));

        error = pow((xp-xd1),2.)+pow((yp-yd1),2.)+pow(((xp+w)-
xd2),2.)+pow((yp-yd2),2.);
        error=sqrt(error);
        printf("u1: %lf\nu2: %lf\nu3: %lf\nphi: %lf\n",
d[2],d[1],d[3],d[0]);
        /*printf("xd1, yd1, xd2, yd2, angle: %lf %lf %lf %lf
%lf\n",xd1, yd1, xd2, yd2, d[0]);
        printf("Count: %d, Error: %lf\n", count, error);*/
        return(error);
    }
}

/*-----
                                DLM Equations
-----*/
void func(double d[], double f[])
{
    h=6.667;
    w=10.0;
    xp=0.;
    yp=0.;

    f[0] = x1*cos(rad(0.)) + z1*cos(rad(90.)) +
d[2]*cos(rad(180.+d[0])) + d[1]*cos(rad(90.+d[0])) +
x2*cos(rad(180.)) + y2*cos(rad(270.));
    f[1] = x1*sin(rad(0.)) + z1*sin(rad(90.)) +
d[2]*sin(rad(180.+d[0])) + d[1]*sin(rad(90.+d[0])) +
x2*sin(rad(180.)) + y2*sin(rad(270.));
    f[2] = x1*cos(rad(0.)) + z1*cos(rad(90.)) +
d[2]*cos(rad(180.+d[0])) + h*cos(rad(90.+d[0])) +
d[3]*cos(rad(d[0])) + x3*cos(rad(180.)) + y3*cos(rad(270.));
    f[3] = x1*sin(rad(0.)) + z1*sin(rad(90.)) +
d[2]*sin(rad(180.+d[0])) + h*sin(rad(90.+d[0])) +
d[3]*sin(rad(d[0])) + x3*sin(rad(180.)) + y3*sin(rad(270.));
}

/*-----
                                RESIDUALS
-----*/
double resid(double f[], int nVar)
{
    int i;
    double tot;

    tot = 0;
    for (i=0; i<nVar; i++)

```

```

    {
        tot = tot + f[i]*f[i];
    }
    return (sqrt(tot));
}

/*-----
      RADIANS/DEGREES CONVERSION
-----*/

double rad(double deg)
{
    return(deg*PI/180.);
}

/*-----
      GASDEV routine (for monte carlo)
-----*/

float gasdev(int *idum)
{
    static int iset=0;
    static float gset;
    float fac, r, v1, v2;
    float ran1();

    if (iset == 0)
    {
        do
        {
            v1 = 2.0 * ran1(idum) - 1.0;
            v2 = 2.0 * ran1(idum) - 1.0;
            r = v1 * v1 + v2 * v2;
        }
        while(r >= 1.0 || r == 0);
        fac = sqrt(-2.0*log(r)/r);
        gset = v1 * fac;
        iset = 1;
        return v2 * fac;
    }
    else
    {
        iset = 0;
        return gset;
    }
}

/*-----
      RAN1 routine
-----*/
#define M1 259200

```

```

#define IA1 7141
#define IC1 54773
#define RM1 (1.0/M1)
#define M2 134456
#define IA2 8121
#define IC2 28411
#define RM2 (1.0/M2)
#define M3 24300
#define IA3 4561
#define IC3 51349

float ran1(int *idum)
{
    static long ix1, ix2, ix3;
    static double r[98];
    double temp;
    static int iff=0;
    int j;

    if (*idum < 0 || iff == 0)
    {
        iff = 1;
        ix1 = (IC1-(*idum)) % M1;
        ix1 = (IA1*ix1+IC1) % M1;
        ix2 = ix1 % M2;
        ix1 = (IA1*ix1+IC1) % M1;
        ix3 = ix1 % M3;
        for (j=1; j<=97; j++)
        {
            ix1 = (IA1*ix1+IC1) % M1;
            ix2 = (IA2*ix2+IC2) % M2;
            r[j] = (ix1+ix2+RM2)* RM1;
        }
        *idum=1;
    }
    ix1 = (IA1*ix1+IC1) %M1;
    ix2 = (IA2*ix2+IC2) %M2;
    ix3 = (IA3*ix3+IC3) %M3;
    j = 1+((97*ix3)/M3);
    if (j>97 || j<1) printf("RAN1: This cannot happen\n");
    r[j] = (ix1+ix2*RM2)*RM1;
    temp = r[j];

    /*printf("the ran1: %d %d\n", ix1, ix2);*/

    return temp;
}

```



## The formulas used in the spreadsheet.

```

length= 10
height= 6.6667

x_base= 0
y_base= 0

x_1= 2.4
y_1= -0.08

x_2= -0.05
y_2= 3.1

x_3= 7.5
y_3= 7
    
```

### VECTOR LOOPS

```

k_1= =x_1+x_base
k_2= =y_1+y_base
k_3= =x_2+x_base
k_4= =y_2+y_base
k_5= =height
k_6= =(x_3+x_base)
k_7= =(y_3+y_base)
u_1= 2.19811669790986
u_2= 3.35904495092889
u_3= 7.82763809053195
phi= 4.41187319831158

hx1= =k_1*COS(0*PI()/180)+k_2*COS(90*PI()/180)+u_1*COS((180+phi)*PI()/180)+u_2*COS((90+phi)*PI()/180)+k_3*COS(180*PI()/180)+k_4*COS(270*PI()/180)
hy1= =k_1*SIN(0*PI()/180)+k_2*SIN(90*PI()/180)+u_1*SIN((180+phi)*PI()/180)+u_2*SIN((90+phi)*PI()/180)+k_3*SIN(180*PI()/180)+k_4*SIN(270*PI()/180)
htheta1= =0+90+90+phi-90+90-phi+90+90

hx2= =k_1*COS(0*PI()/180)+k_2*COS(90*PI()/180)+u_1*COS((180+phi)*PI()/180)+k_5*COS((90+phi)*PI()/180)+u_3*COS(phi*PI()/180)+k_6*COS(180*PI()/180)+k_7*COS(270*PI()/180)
hy2= =k_1*SIN(0*PI()/180)+k_2*SIN(90*PI()/180)+u_1*SIN((180+phi)*PI()/180)+k_5*SIN((90+phi)*PI()/180)+u_3*SIN(phi*PI()/180)+k_6*SIN(180*PI()/180)+k_7*SIN(270*PI()/180)
htheta2= =0+90+90+phi-90+90+180+90-phi+90

Error
xd1= =x_2 + SIN(phi*PI()/180)*u_2
yd1= =y_2 - COS(phi*PI()/180)*u_2
xd2= =xd_1 + length*COS(phi*PI()/180)
yd2= =yd_1 +length*SIN(phi*PI()/180)

error = =(x_base-xd_1)^2+(y_base-yd_1)^2+(x_base+length)-xd_2)^2+(y_base-yd_2)^2
sqrt error= =SQRT(error)
    
```

Example results from the spreadsheet above.

length=	10
height=	6.6667
x_base=	0
y_base=	0
x_1=	2.4
y_1=	-0.08
x_2=	-0.05
y_2=	3.1
x_3=	7.5
y_3=	7

**VECTOR LOOPS**

k_1=	2.4
k_2=	-0.08
k_3=	-0.05
k_4=	3.1
k_5=	6.6667
k_6=	7.5
k_7=	7
u_1=	2.198116698
u_2=	3.359044951
u_3=	7.827638091
phi=	4.411873198
hx1=	8.44935E-13
hy1=	-7.66942E-13
htheta1=	360
hx2=	5.00534E-13
hy2=	-1.08002E-12
htheta2=	360

**Error**

xd1=	0.20839669
yd1=	-0.249091538
xd2=	10.17876502
yd2=	0.520164883
error =	0.408004211
sqrt error=	0.638752073

## E.2 BLOCK WITH FOUR CONSTRAINTS (SECTION 6.3.3)

The C program shown below allows for all four constraints to be varied in both directions. The starting points are changed for each set-up presented.

```

#include <math.h>
#include <stdio.h>
#include "LUD.h"

#define EPS 0.000001
#define PERTURB 0.000001
#define PI 3.14159265

void func1(double d[], double f[]);
void func2(double d[], double f[]);
void func3(double d[], double f[]);
double resid1(double f[], int nVar);
double resid2(double f[], int nVar);
double resid3(double f[], int nVar);
double rad(double deg);
double nr1(double angle);
double nr2(double angle);
double nr3(double angle);

double x1, z1, x2, y2, x3, y3, x4,
y4;
double xp, yp, h, w;
int nfail1, nfail2, nfail3;
int constraintfail1, constraintfail2,
constraintfail3;
int nochance1, nochance2, nochance3;
int fail1, fail2, fail3, never, one,
two, three, assembled;

float ran1(int *idum);
float gasdev(int *idum);

/*-----
      MONTE CARLO
-----*/
main ()
{
    double errorcase1, errorcase2,
errorcase3, error1, error2, error3;
    double maxerror, totmaxerror,
error, erroroverall;
    double aerrorone, averrortwo,
averrorthree, averagetwo,
averagethree;
    int iSeed, i, nloop, icnt, ierror;

    iSeed = -5;
    nfail1 = 0;
    nfail2 = 0;
    nfail3 = 0;
    constraintfail1 = 0;
    constraintfail2 = 0;
    constraintfail3 = 0;
    nochance1 = 0;
    nochance2 = 0;
    nochance3 = 0;
    error1 = 0.;
    error2 = 0.;
    error3 = 0.;
    never = 0;
    one = 0;
    two = 0;
    three = 0;
    aerrorone = 0.;
    averrortwo = 0.;
    averrorthree = 0.;

    printf ("Enter number of
loops\n");
    scanf ("%d", &nloop);

    for (i=1; i<=nloop; i++)
    {
        /* generate random deviates */

        x1 = gasdev(&iSeed) * 0.2 + 2.0;
        z1 = gasdev(&iSeed) * 0.2 +
0.00;
        x2 = gasdev(&iSeed) * 0.2 +
0.00;
        y2 = gasdev(&iSeed) * 0.2 + 2.5;
        x3 = gasdev(&iSeed) * 0.2 + 4.0;
        y3 = gasdev(&iSeed) * 0.2 +
6.667;
        x4 = gasdev(&iSeed) * 0.2 + 5.5;
        y4 = gasdev(&iSeed) * 0.2 +
0.00;

        error1 = nr1(0.);
        error2 = nr2(0.);
        error3 = nr3(0.);

        if ((fail1 == 1) && (fail2 ==
1) && (fail3 == 1))
        {
            never += 1;
        }

        if (((fail1 == 1) && (fail2 ==
1) && (fail3 == 0)) || ((fail2 == 1)
&& (fail3 == 1) && (fail1 == 0)) ||
((fail1 == 1) && (fail3 == 1) &&
(fail2 == 0)))
        {
            one += 1;
            aerrorone +=
(error1+error2+error3);
        }
    }
}

```

```

        if (((fail1 == 1) && (fail2 ==
0) && (fail3 == 0)) || ((fail2 == 1)
&& (fail3 == 0) && (fail1 == 0)) ||
((fail1 == 0) && (fail3 == 1) &&
(fail2 == 0)))
        {
            two += 1;
            averagetwo =
(error1+error2+error3)/2;
            averrortwo += averagetwo;
        }

        if (((fail1 == 0) && (fail2 ==
0) && (fail3 == 0)) || ((fail2 == 0)
&& (fail3 == 0) && (fail1 == 0)) ||
((fail1 == 0) && (fail3 == 0) &&
(fail2 == 0)))
        {
            three += 1;
            averagethree =
(error1+error2+error3)/3;
            averrorthree += averagethree;
        }

        maxerror = 0.;
        erroroverall = 0.;
        icnt = 0;

        if ((fail1 == 0) || (fail2 == 0)
|| (fail3 == 0))
        {
            assembled += 1;

            if(fail1 == 0)
            {
                if( fabs(error1) >
maxerror) maxerror = fabs(error1);
                erroroverall +=
fabs(error1);
                icnt += 1;
            }

            if (fail2 == 0)
            {
                if( fabs(error2) >
maxerror) maxerror = fabs(error2);
                erroroverall +=
fabs(error2);
                icnt += 1;
            }

            if (fail3 == 0)
            {
                if(fabs(error3) >
maxerror) maxerror = fabs(error3);
                erroroverall +=
fabs(error3);
                icnt += 1;
            }

            if (erroroverall != 0.)
            {
                ierror += 1;
                error +=
(erroroverall/icnt);
                totmaxerror += (maxerror);
            }

            errorcase1 += fabs(error1);
            errorcase2 += fabs(error2);
            errorcase3 += fabs(error3);
        }

        /* output scalar AF values */

        /*printf ("nfail1 =
%d\n", nfail1);
        printf ("constraintfail1 = %d\n",
constraintfail1);
        printf ("nochance1 = %d\n",
nochance1);
        printf ("nfail2 = %d\n",
nfail2);
        printf ("constraintfail2 = %d\n",
constraintfail2);
        printf ("nochance2 = %d\n",
nochance2);
        printf ("nfail3 = %d\n",
nfail3);
        printf ("constraintfail3 = %d\n",
constraintfail3);
        printf ("nochance3 = %d\n",
nochance3);*/
        printf ("error average1 = %lf\n",
errorcase1/(nloop-
(constraintfail1+nfail1-nochance1)));
        printf ("error average2 = %lf\n",
errorcase2/(nloop-
(constraintfail2+nfail2-nochance2)));
        printf ("error average3 = %lf\n",
errorcase3/(nloop-
(constraintfail3+nfail3-nochance3)));
        printf ("never assembles = %d\n",
never);
        printf ("one assembles = %d\n",
one);
        printf ("error for one = %lf\n",
averrorone/(one));
        printf ("two assemble = %d\n",
two);
        printf ("error for two = %lf\n",
averrortwo/(two));
        printf ("three assemble = %d\n",
three);
        printf ("error for three = %lf\n",
averrorthree/(three));
        printf ("Total assembled = %d\n",
assembled);
        printf ("average error = %lf\n",
error/(ierror));
        printf ("Max ave error = %lf\n",
totmaxerror/(ierror));
    }

```

```

/*-----
NR routine-Case 1
-----*/
double nr1(double angle)
{
    int i, j, nRow, nCol, nB, nVar,
    count1, pvt[MAX_ROWS];
    double a[MAX_ROWS][MAX_COLS],
    b[MAX_ROWS], det;
    double d[MAX_ROWS], f1[MAX_ROWS],
    f[MAX_ROWS], totalResid1;
    double error1, xd1, xd2, yd1, yd2,
    yc4;

    /* initialize x */
    nVar = 4;
    d[0] = angle;
    d[1] = y2;
    d[2] = x1;
    d[3] = x3;

    /* call functions */
    func1(d,f);

    /* compute residuals */
    totalResid1 = resid1(f,nVar);

    /* enter main loop */
    count1 = 0;
    fail1 = 0;

    while (fabs(totalResid1) > EPS)
    {
        count1++;
        /*printf ("Residuals %lf\n",
totalResid);*/

        /* Evaluate the Jacobian */
        for (i=0; i<nVar; i++)
        {
            /* perturb x */
            d[i] = d[i] + PERTURB;
            func1(d, f1);

            for (j = 0; j<nVar; j++)
            {
                a[j][i] = (f1[j]-f[j]) /
PERTURB;
            }
            d[i] = d[i] - PERTURB;
        }

        /* Make sure the functions are
current */
        func1(d, f);

        /* load b vector */
        for (i=0; i<nVar; i++)
        {
            b[i] = -f[i];
        }

        /* print out vector b */
        /*printf ("vector b:\n");
for (i=0; i<nVar; i++)
        {
            printf("%lf ", b[i]);
        }
        printf("\n"); */

        /* call LU DECOMPOSITION
routine */
        det = LUdecomp(a, nVar, pvt);
        LUSolve(a, nVar, b, pvt);

        /* print out matrix a */
        /* printf ("matrix a:\n");
for (i=0; i<nVar; i++)
        {
            for (j=0; j<nVar; j++)
            {
                printf("%lf ", a[i][j]);
            }
            printf("\n");
        } */

        /* print out vector b */
        /*printf ("delta x:\n");
for (i=0; i<nVar; i++)
        {
            printf("%lf ", b[i]);
        }
        printf("\n"); */

        /* compute new value for x */
        for (i=0; i<nVar; i++)
        {
            d[i] = d[i] + b[i];
        }

        /* print new d */
        /*printf ("new d:\n");
for (i=0; i<nVar; i++)
        {
            printf("%lf ", d[i]);
        }
        printf("\n"); */

        /* evaluate the function's
residuals */
        func1(d, f);
        totalResid1 = resid1(f, nVar);
        if (count1 > 15)
        {
            nfail1 +=1;
            /*printf("Failure in
NR1\n");*/
            break;
        }
    }

}

/*-----
CHECK THE 4TH CONSTRAINT
-----*/

```

```

-----*/
    xd1=x2 + sin(rad(d[0]))*d[1];
    yd1=y2 - cos(rad(d[0]))*d[1];
    xd2=xd1 + w*cos(rad(d[0]));
    yd2=yd1 + w*sin(rad(d[0]));

    yc4=((yd2-yd1)/(xd2-
xd1))*x4+((xd1-
x2)*tan(rad(d[0]))+yd1);

    if ((count1 > 15) && (y4 >
yc4))
    {
        nochance1 +=1;
    }

    if((count1 > 15) || (y4 > yc4))
    {
        fail1 = 1;
    }

    if (y4 > yc4)
    {
        constraintfail1 +=1;
        /*printf("Constraint4
crashed\n");*/
        error1=0.0;
        return(error1);
    }

/*-----
        EVALUATE ERROR
-----*/

    if ((y4 < yc4) && (count1 <=
15))
    {
        error1 = pow((xp-
xd1),2.)+pow((yp-
yd1),2.)+pow(((xp+w)-
xd2),2.)+pow((yp-yd2),2.);
        error1 = sqrt(error1);
        /*printf("u1: %lf\nu2:
%lf\nu3: %lf\nphi: %lf\n",
d[2],d[1],d[3],d[0]);*/
        /*printf("xd1, yd1, xd2,
yd2, angle: %lf %lf %lf %lf
%lf\n",xd1, yd1, xd2, yd2, d[0]);
        /*printf("Count: %d,
Error: %lf\n", count, error);*/
        return(error1);
    }
}

/*-----DLM
        Equations
-----*/
void func1(double d[], double f[])
{
    h=6.667;

```

```

    w=10.0;
    xp=0.0;
    yp=0.0;

    f[0] = x1*cos(rad(0.)) +
z1*cos(rad(90.)) +
d[2]*cos(rad(180.+d[0])) +
d[1]*cos(rad(90.+d[0])) +
x2*cos(rad(180.)) +
y2*cos(rad(270.));
    f[1] = x1*sin(rad(0.)) +
z1*sin(rad(90.)) +
d[2]*sin(rad(180.+d[0])) +
d[1]*sin(rad(90.+d[0])) +
x2*sin(rad(180.)) +
y2*sin(rad(270.));
    f[2] = x1*cos(rad(0.)) +
z1*cos(rad(90.)) +
d[2]*cos(rad(180.+d[0])) +
h*cos(rad(90.+d[0])) +
d[3]*cos(rad(d[0])) +
x3*cos(rad(180.)) +
y3*cos(rad(270.));
    f[3] = x1*sin(rad(0.)) +
z1*sin(rad(90.)) +
d[2]*sin(rad(180.+d[0])) +
h*sin(rad(90.+d[0])) +
d[3]*sin(rad(d[0])) +
x3*sin(rad(180.)) +
y3*sin(rad(270.));
}

/*-----
        RESIDUALS
-----*/
double resid1(double f[], int nVar)
{
    int i;
    double tot;

    tot = 0;
    for (i=0; i<nVar; i++)
    {
        tot = tot + f[i]*f[i];
    }
    return (sqrt(tot));
}

/*-----
        NR routine-Case 2
-----*/
double nr2(double angle)
{
    int i, j, nRow, nCol, nB, nVar,
count2, pvt[MAX_ROWS];
    double a[MAX_ROWS][MAX_COLS],
b[MAX_ROWS], det;
    double d[MAX_ROWS], f1[MAX_ROWS],
f[MAX_ROWS], totalResid2;
    double error2, xd1, xd2, yd1, yd2,
yc1;

    /* initialize x */

```

```

nVar = 4;
d[0] = angle;
d[1] = y2;
d[2] = x4;
d[3] = x3;

/* call functions */
func2(d,f);

/* compute residuals */
totalResid2 = resid2(f,nVar);

/* enter main loop */
count2 = 0;
fail2 = 0;

while (fabs(totalResid2) > EPS)
{
    count2++;
    /*printf ("Residuals %lf\n",
totalResid);*/

    /* Evaluate the Jacobian */
    for (i=0; i<nVar; i++)
    {
        /* perturb x */
        d[i] = d[i] + PERTURB;
        func2(d, f1);

        for (j = 0; j<nVar; j++)
        {
            a[j][i] = (f1[j]-f[j]) /
PERTURB;
        }
        d[i] = d[i] - PERTURB;
    }

    /* printf ("matrix a:\n");
for (i=0; i<nVar; i++)
    {
        for (j=0; j<nVar; j++)
            printf("%lf ", a[i][j]);
        printf("\n");
    }
printf("\n"); */

    /* Make sure the functions are
current */
    func2(d, f);

    /* load b vector */
    for (i=0; i<nVar; i++)
    {
        b[i] = -f[i];
    }

    /* print out vector b */
    /*printf ("vector b:\n");
for (i=0; i<nVar; i++)
    {
        printf("%lf ", b[i]);
    }

}

printf("\n"); */

/* call LU DECOMPOSITION
routine */
det = LUDecomp(a, nVar, pvt);
LUSolve(a, nVar, b, pvt);

/* print out matrix a */
/* printf ("matrix a:\n");
for (i=0; i<nVar; i++)
    {
        for (j=0; j<nVar; j++)
            printf("%lf ", a[i][j]);
        printf("\n");
    }
*/

/*printf ("delta x:\n");
for (i=0; i<nVar; i++)
    {
        printf("%lf ", b[i]);
    }
printf("\n"); */

/* compute new value for x */
for (i=0; i<nVar; i++)
    {
        d[i] = d[i] + b[i];
    }

/* print new d */
/*printf ("new d:\n");
for (i=0; i<nVar; i++)
    {
        printf("%lf ", d[i]);
    }
printf("\n"); */

/* evaluate the function's
residuals */
func2(d, f);
totalResid2 = resid2(f, nVar);
if (count2 > 15)
    {
        nfail2 +=1;
        /*printf("Failure in
NR2\n");*/
        break;
    }
}

/*-----
CHECK THE 4TH CONSTRAINT
-----*/
xd1=x2 + sin(rad(d[0]))*d[1];
yd1=y2 - cos(rad(d[0]))*d[1];
xd2=xd1 + w*cos(rad(d[0]));
yd2=yd1 + w*sin(rad(d[0]));

```

```

        yc1=((yd2-yd1)/(xd2-
xd1))*x1+((xd1-
x2)*tan(rad(d[0]))+yd1);

        if ((count2 > 15) && (z1 >
yc1))
        {
            nochance2 +=1;
        }

        if ((count2 > 15) || (z1 >
yc1))
        {
            fail2 = 1;
        }

        if (z1 > yc1)
        {
            constraintfail2 +=1;
            /*printf("Constraint1
crashed\n");*/
            error2=0.0;
            return(error2);
        }

        /*-----
            EVALUATE ERROR
        -----*/

        if ((z1 < yc1) && (count2 <=
15))
        {
            error2 = pow((xp-
xd1),2.)+pow((yp-
yd1),2.)+pow(((xp+w)-
xd2),2.)+pow((yp-yd2),2.);
            error2 = sqrt(error2);
            /*printf("u1: %lf\nu2:
%lf\nu3: %lf\nphi: %lf\n",
            d[2],d[1],d[3],d[0]);*/
            /*printf("xd1, yd1, xd2,
yd2, angle: %lf %lf %lf %lf
%lf\n",xd1, yd1, xd2, yd2, d[0]);
            printf("Count: %d, Error:
%lf\n", count, error);*/
            return(error2);
        }
    }

    /*-----DLM
        Equations
    -----*/
    void func2(double d[], double f[])
    {
        h= 6.667;
        w= 10.0;
        xp=0.;
        yp=0.;

        f[0] = x4*cos(rad(0.)) +
y4*cos(rad(90.)) +
d[2]*cos(rad(180.+d[0])) +
d[1]*cos(rad(90.+d[0])) +
x2*cos(rad(180.)) +
y2*cos(rad(270.));
        f[1] = x4*sin(rad(0.)) +
y4*sin(rad(90.)) +
d[2]*sin(rad(180.+d[0])) +
d[1]*sin(rad(90.+d[0])) +
x2*sin(rad(180.)) +
y2*sin(rad(270.));
        f[2] = x4*cos(rad(0.)) +
y4*cos(rad(90.)) +
d[2]*cos(rad(180.+d[0])) +
h*cos(rad(90.+d[0])) +
d[3]*cos(rad(d[0])) +
x3*cos(rad(180.)) +
y3*cos(rad(270.));
        f[3] = x4*sin(rad(0.)) +
y4*sin(rad(90.)) +
d[2]*sin(rad(180.+d[0])) +
h*sin(rad(90.+d[0])) +
d[3]*sin(rad(d[0])) +
x3*sin(rad(180.)) +
y3*sin(rad(270.));
    }

    /*-----
        RESIDUALS
    -----*/
    double resid2(double f[], int nVar)
    {
        int i;
        double tot;

        tot = 0;
        for (i=0; i<nVar; i++)
        {
            tot = tot + f[i]*f[i];
        }
        return (sqrt(tot));
    }

    /*-----
        NR routine-Case 3
    -----*/
    double nr3(double angle)
    {
        int i, j, nRow, nCol, nB, nVar,
count3, pvt[MAX_ROWS];
        double a[MAX_ROWS][MAX_COLS],
b[MAX_ROWS], det;
        double d[MAX_ROWS], f1[MAX_ROWS],
f[MAX_ROWS], totalResid3;
        double error3, xd1, xd2, yd1, yd2,
yc3;

        /* initialize x */
        nVar = 4;
        d[0] = angle;

```



```

d[1] = y2;
d[2] = x1;
d[3] = x4;

/* call functions */
func3(d,f);

/* compute residuals */
totalResid3 = resid3(f,nVar);

/* enter main loop */
count3 = 0;
fail3 = 0;

while (fabs(totalResid3) > EPS)
{
    count3++;
    /*printf ("Residuals %lf\n",
totalResid);*/

    /* Evaluate the Jacobian */
    for (i=0; i<nVar; i++)
    {
        /* perturb x */
        d[i] = d[i] + PERTURB;
        func3(d, f1);

        for (j = 0; j<nVar; j++)
        {
            a[j][i] = (f1[j]-f[j]) /
PERTURB;
        }
        d[i] = d[i] - PERTURB;
    }

    /* printf ("matrix a:\n");
for (i=0; i<nVar; i++)
    {
        for (j=0; j<nVar; j++)
        {
            printf("%lf ", a[i][j]);
        }
        printf("\n");
    }
printf("\n"); */

    /* Make sure the functions are
current */
    func3(d, f);

    /* load b vector */
    for (i=0; i<nVar; i++)
    {
        b[i] = -f[i];
    }

    /*printf ("vector b:\n");
for (i=0; i<nVar; i++)
    {
        printf("%lf ", b[i]);
    }
printf("\n"); */

        /* call LU DECOMPOSITION
routine */
        det = LUDecomp(a, nVar, pvt);
        LUSolve(a, nVar, b, pvt);

        /* printf ("matrix a:\n");
for (i=0; i<nVar; i++)
    {
        for (j=0; j<nVar; j++)
        {
            printf("%lf ", a[i][j]);
        }
        printf("\n");
    } */

        /*printf ("delta x:\n");
for (i=0; i<nVar; i++)
    {
        printf("%lf ", b[i]);
    }
printf("\n"); */

        /* compute new value for x */
        for (i=0; i<nVar; i++)
        {
            d[i] = d[i] + b[i];
        }

        /* print new d */
        /*printf ("new d:\n");
for (i=0; i<nVar; i++)
    {
        printf("%lf ", d[i]);
    }
printf("\n"); */

        /* evaluate the function's
residuals */
        func3(d, f);
        totalResid3 = resid3(f, nVar);
        if (count3 > 15)
        {
            nfail3 +=1;
            printf("Failure in NR3\n");
            break;
        }
    }

    /*-----
        CHECK THE 4TH CONSTRAINT
    -----*/
    xd1=x2 + sin(rad(d[0]))*d[1];
    yd1=y2 - cos(rad(d[0]))*d[1];
    xd2=xd1 + w*cos(rad(d[0]));
    yd2=yd1 + w*sin(rad(d[0]));

    yc3=((yd2-yd1)/(xd2-
xd1))*x3+(((xd1-x2)*tan(rad(d[0]))+
yd1)+ h);

    /*printf("count3 = %d\n",
count3);

```

```

        printf("y3 = %lf\n", y3);
        printf("yc3 = %lf\n", yc3);*/

        if ((count3 > 15) && (y3 <
yc3))
        {
            nochance3 +=1;
        }

        /*printf("fail3beforeif =
%d\n", fail3);*/

        if((count3 > 15) || (y3 < yc3))
        {
            fail3 = 1;
        }

        /*printf("fail3afterif = %d\n",
fail3);*/

        if (y3 < yc3)
        {
            constraintfail3 +=1;
            /*printf("Constraint3
crashed\n");*/
            error3 = 0.0;
            return(error3);
        }

        /*-----
                EVALUATE ERROR
        -----*/

        if ((y3 > yc3) && (count3 <=
15))
        {
            error3 = pow((xp-
xd1),2.)+pow((yp-
yd1),2.)+pow(((xp+w)-
xd2),2.)+pow((yp-yd2),2.);
            error3 = sqrt(error3);
            /*printf("u1: %lf\nu2:
%lf\nu3: %lf\nphi: %lf\n",
d[2],d[1],d[3],d[0]);*/
            /*printf("xd1, yd1, xd2,
yd2, angle: %lf %lf %lf %lf
%lf\n",xd1, yd1, xd2, yd2, d[0]);
            printf("Count: %d, Error:
%lf\n", count, error);*/
            return(error3);
        }
    }

    /*-----DLM
                Equations
    -----*/
    void func3(double d[], double f[])
    {
        h=6.667;
        w=10.0;
        xp=0.;

        yp=0.;

        f[0] = x1*cos(rad(0.)) +
z1*cos(rad(90.)) +
d[2]*cos(rad(180.+d[0])) +
d[1]*cos(rad(90.+d[0])) +
x2*cos(rad(180.)) +
y2*cos(rad(270.));
        f[1] = x1*sin(rad(0.)) +
z1*sin(rad(90.)) +
d[2]*sin(rad(180.+d[0])) +
d[1]*sin(rad(90.+d[0])) +
x2*sin(rad(180.)) +
y2*sin(rad(270.));
        f[2] = x4*cos(rad(0.)) +
y4*cos(rad(90.)) +
d[3]*cos(rad(180.+d[0])) +
d[1]*cos(rad(90.+d[0])) +
x2*cos(rad(180.)) +
y2*cos(rad(270.));
        f[3] = x4*sin(rad(0.)) +
y4*sin(rad(90.)) +
d[3]*sin(rad(180.+d[0])) +
d[1]*sin(rad(90.+d[0])) +
x2*sin(rad(180.)) +
y2*sin(rad(270.));
    }

    /*-----
                RESIDUALS
    -----*/
    double resid3(double f[], int nVar)
    {
        int i;
        double tot;

        tot = 0;
        for (i=0; i<nVar; i++)
        {
            tot = tot + f[i]*f[i];
        }
        return (sqrt(tot));
    }

    /*-----
                RADIANS/DEGREES CONVERSION
    -----*/
    double rad(double deg)
    {
        return(deg*PI/180.);
    }

    /*-----
                GASDEV routine (for monte carlo)
    -----*/
    float gasdev(int *idum)
    {
        static int iset=0;
        static float gset;
        float fac, r, v1, v2;
        float ran1();

        if (iset == 0)

```

```

    {
        do
        {
            v1 = 2.0 * ran1(idum) - 1.0;
            v2 = 2.0 * ran1(idum) - 1.0;
            r = v1 * v1 + v2 * v2;
        }
        while(r >= 1.0 || r == 0);
        fac = sqrt(-2.0*log(r)/r);
        gset = v1 * fac;
        iset = 1;
        return v2 * fac;
    }
    else
    {
        iset = 0;
        return gset;
    }
}

/*-----
          RAN1 routine
-----*/
#define M1 259200
#define IA1 7141
#define IC1 54773
#define RM1 (1.0/M1)
#define M2 134456
#define IA2 8121
#define IC2 28411
#define RM2 (1.0/M2)
#define M3 24300
#define IA3 4561
#define IC3 51349

float ran1(int *idum)
{
    static long ix1, ix2, ix3;
    static double r[98];

    double temp;
    static int iff=0;
    int j;

    if (*idum < 0 || iff == 0)
    {
        iff = 1;
        ix1 = (IC1-(*idum)) % M1;
        ix1 = (IA1*ix1+IC1) % M1;
        ix2 = ix1 % M2;
        ix1 = (IA1*ix1+IC1) % M1;
        ix3 = ix1 % M3;
        for (j=1; j<=97; j++)
        {
            ix1 = (IA1*ix1+IC1) % M1;
            ix2 = (IA2*ix2+IC2) % M2;
            r[j] = (ix1+ix2+RM2)* RM1;
        }
        *idum=1;
    }
    ix1 = (IA1*ix1+IC1) %M1;
    ix2 = (IA2*ix2+IC2) %M2;
    ix3 = (IA3*ix3+IC3) %M3;
    j = 1+((97*ix3)/M3);
    if (j>97 || j<1) printf("RAN1:
This cannot happen\n");
    r[j] = (ix1+ix2*RM2)*RM1;
    temp = r[j];

    /*printf("the ran1: %d %d\n", ix1,
ix2);*/

    return temp;
}

```

### ***E.3 SLOTTED BLOCK ASSEMBLY (SECTION 6.3.4)***

The C program shown below allows the right peg to be varied in the x-direction. The starting slot angle is input for each set-up presented in the thesis.

```

#include <math.h>
#include <stdio.h>
#include "LUD.h"

#define EPS 0.000001
#define PERTURB 0.000001
#define PI 3.14159265

void func(double d[], double f[]);
double resid(double f[], int nVar);
double rad(double deg);

double nr(double angle);

double x1, y1, x2, y2, x3, y3, x4,
y4, r, phi;
double xp, yp, delx, dely, theta;
int nfail;

float ran1(int *idum);
float gasdev(int *idum);

```

```

/*-----
          MONTE CARLO
-----*/
main ()
{
    double error1, error, theta, d[1],
    y, x;
    int iSeed, i, nloop;

    FILE *data;

    /* Information for the slot:
       distance from center of slot to
       bottom = 0.5
       distance from center of slot to
       left sd= 2 */

    y = 0.5;
    x = 2.0;

    printf ("Angle of the slot
    (CCW+):\n");
    scanf ("%lf", &phi);

    /* to help find x2 */
    delx = cos(rad(((180.- phi)/2)-
    (atan(y/x)*180./PI)))*2*sin(rad(phi/2
    )) *
    (y/sin((atan(y/x))));

    /* to help find y3 */
    dely = sin(rad(((180.- phi)/2.)-
    (atan(y/x)*180./PI)))*2.*sin(rad(phi/
    2.)) *
    (y/sin((atan(y/x))));

    /*printf ("delx: %lf\n dely:
    %lf\n", delx, dely);*/

    x1 = 2.5;
    x2 = 4 + delx;
    x3 = 2;
    y1 = 3;
    y2 = 2;
    y3 = 2 - dely;
    y4 = 3;
    r = 0.5;

    iSeed = -5;
    nfail = 0;
    error = 0.;

    strcat("data", "1.txt");
    data = fopen("data", "w");

    printf ("Enter number of
    loops\n");
    scanf ("%d", &nloop);

    for (i=1; i<=nloop; i++)
    {
        /* generate random deviates */

        x4 = gasdev(&iSeed) * 0.3 +
        6.5;

        printf ("x4:%lf\n", x4);

        error1 = nr(0.);
        error += fabs(error1);
    }

    /* output scalar AF values */

    printf ("nfail = %d\n", nfail);
    fprintf (data,"nfail = %d\n",
    nfail);
    printf ("error average =
    %lf\n", error/(nloop-nfail));
    fprintf (data,"error average =
    %lf\n", error/(nloop-nfail));

    fclose(data);
}

/*-----
          NR routine
-----
-*/
double nr(double angle)
{
    int i, j, nRow, nCol, nB, nVar,
    count, pvt[MAX_ROWS];
    double a[MAX_ROWS][MAX_COLS],
    b[MAX_ROWS], det;
    double d[MAX_ROWS],
    f1[MAX_ROWS], f[MAX_ROWS],
    totalResid;
    double error, xd1, xd2, yd1,
    yd2, u1, base, x_left, x_right;
    double y_left, y_right, theta,
    wanted_angle, angle1, angle2;
    double dxold, dyold, xp, yp,
    length_side, length_bottom;

    /* initialize d */
    nVar = 2;
    d[0] = angle;
    d[1] = 2.;

    /* call functions */
    func(d,f);

    /* compute residuals */
    totalResid = resid(f,nVar);

    /* enter main loop */
    count = 0;

    while (fabs(totalResid) > EPS)
    {
        count++;
        /*printf ("Residuals %lf\n",
        totalResid);*/
    }
}

```

```

/* Evaluate the Jacobian */
for (i=0; i<nVar; i++)
{
    /* perturb x */
    d[i] = d[i] + PERTURB;
    func(d, f1);

    for (j = 0; j<nVar; j++)
    {
        a[j][i] = (f1[j]-f[j]) /
PERTURB;
    }
    d[i] = d[i] - PERTURB;
}

/* printf ("matrix a:\n");
for (i=0; i<nVar; i++)
{
    for (j=0; j<nVar; j++)
    {
        printf("%lf ", a[i][j]);
    }
    printf("\n");
}
printf("\n"); */

/* Make sure the functions are
current */
func(d, f);

/* load b vector */
for (i=0; i<nVar; i++)
{
    b[i] = -f[i];
}

/*printf ("vector b:\n");
for (i=0; i<nVar; i++)
{
    printf("%lf ", b[i]);
}
printf("\n"); */

/* call LU DECOMPOSITION
routine */
det = LUDecomp(a, nVar, pvt);
LUSolve(a, nVar, b, pvt);

/* print out matrix a */
/* printf ("matrix a:\n");
for (i=0; i<nVar; i++)
{
    for (j=0; j<nVar; j++)
    {
        printf("%lf ", a[i][j]);
    }
    printf("\n");
} */

/* print out vector b */
/*printf ("delta x:\n");
for (i=0; i<nVar; i++)
{
    printf("%lf ", b[i]);
}
printf("\n"); */

/* compute new value for x
*/
for (i=0; i<nVar; i++)
{
    d[i] = d[i] + b[i];
}

/* print new d */
/*printf ("new d:\n");
for (i=0; i<nVar; i++)
{
    printf("%lf ", d[i]);
}
printf("\n"); */

/* evaluate the function's
residuals */
func(d, f);
totalResid = resid(f, nVar);
if (count > 15)
{
    nfail +=1;
    printf("Failure in NR\n");
    break;
}
}

while(d[0] > 360 || d[0] < -
180)
{
    if(d[0] >= 0)
    {
        d[0] = 360 - d[0];
    }
    if (d[0] < 0)
    {
        d[0] = 360 + d[0];
    }
}

/*-----
EVALUATE ERROR
-----
*/

if (count <= 15)
{
    dxold=2;
    dyold=2.5;
    base = 9.0;
    xp = 0.5;
    yp = 0.5;
    theta = d[0];
}

```

```

        length_side =
sqrt(pow(dxold,2)+pow(dyold,2));
        length_bottom =
tan(rad(theta))*length_side;
angle1=atan(dyold/dxold)*180/PI;
        angle2=(180-theta)/2;
        wanted_angle = 180-
(theta+angle1+angle2);

        x_left = xp +
cos(rad(wanted_angle)) *
length_bottom;
        y_left = yp -
sin(rad(wanted_angle)) *
length_bottom;

        x_right = x_left + base *
cos(rad(theta));
        y_right = y_left + base *
sin(rad(theta));

        error = pow((xp-
x_left),2)+pow((yp-
y_left),2)+pow(((xp+base)-
x_right),2)
                +pow((yp-
y_right),2);
        error = sqrt(error);

        /*printf("x_left: %lf
y_left: %lf\n x_right: %lf y_right:
%lf\n angle:
%lf\n", x_left, y_left,
x_right, y_right, theta);*/

        printf("u1: %lf angle:
%lf\n", d[1], theta);

        /*printf("Simulation number
%d, Error: %lf\n", count, error);*/

        return(error);
    }
}

/*-----DLM
Equations
-----*/

void func(double d[], double f[])
{
    f[0] = x1*cos(rad(0.)) +
y1*cos(rad(90.))+r*cos(rad(270.+d[0]))
        + y2*cos(rad(270.+d[0])) +
x3*cos(rad(180.+d[0]))
        + x2*cos(rad(d[0])) +
y3*cos(rad(90.+d[0]))
        + d[1]*cos(rad(d[0]+phi)) +
r*cos(rad(90.+d[0]+phi))
        + y4*cos(rad(270.)) +
x4*cos(rad(180.));

    f[1] = x1*sin(rad(0.)) +
y1*sin(rad(90.))+r*sin(rad(270.+d[0]))
        + y2*sin(rad(270.+d[0]))
+ x3*sin(rad(180.+d[0]))
        + x2*sin(rad(d[0])) +
y3*sin(rad(90.+d[0]))
        +
d[1]*sin(rad(d[0]+phi)) +
r*sin(rad(90.+d[0]+phi))
        + y4*sin(rad(270.)) +
x4*sin(rad(180.));
}

/*-----
RESIDUALS
-----
-*/

double resid(double f[], int nVar)
{
    int i;
    double tot;

    tot = 0;
    for (i=0; i<nVar; i++)
    {
        tot = tot + f[i]*f[i];
    }
    return (sqrt(tot));
}

/*-----
RADIANS/DEGREES CONVERSION
-----
-*/

double rad(double deg)
{
    return(deg*PI/180.);
}

/*-----
GASDEV routine (for monte carlo)
-----
-*/

float gasdev(int *idum)
{
    static int iset=0;
    static float gset;
    float fac, r, v1, v2;
    float ran1();

    if (iset == 0)
    {
        do
        {

```

```

        v1 = 2.0 * ran1(idum) - 1.0;
        v2 = 2.0 * ran1(idum) - 1.0;
        r = v1 * v1 + v2 * v2;
    }
    while(r >= 1.0 || r == 0);
    fac = sqrt(-2.0*log(r)/r);
    gset = v1 * fac;
    iset = 1;
    return v2 * fac;
}
else
{
    iset = 0;
    return gset;
}
}

/*-----
          RAN1 routine
-----*/

#define M1 259200
#define IA1 7141
#define IC1 54773
#define RM1 (1.0/M1)
#define M2 134456
#define IA2 8121
#define IC2 28411
#define RM2 (1.0/M2)
#define M3 24300
#define IA3 4561
#define IC3 51349

float ran1(int *idum)
{
    static long ix1, ix2, ix3;
    static double r[98];
    double temp;
    static int iff=0;
    int j;

    if (*idum < 0 || iff == 0)
    {
        iff = 1;
        ix1 = (IC1-(*idum)) % M1;
        ix1 = (IA1*ix1+IC1) % M1;
        ix2 = ix1 % M2;
        ix1 = (IA1*ix1+IC1) % M1;
        ix3 = ix1 % M3;
        for (j=1; j<=97; j++)
        {
            ix1 = (IA1*ix1+IC1) % M1;
            ix2 = (IA2*ix2+IC2) % M2;
            r[j] = (ix1+ix2+RM2)*
RM1;
        }
        *idum=1;
    }
    ix1 = (IA1*ix1+IC1) %M1;
    ix2 = (IA2*ix2+IC2) %M2;
    ix3 = (IA3*ix3+IC3) %M3;
    j = 1+((97*ix3)/M3);
    if (j>97 || j<1) printf("RAN1:
This cannot happen\n");
    r[j] = (ix1+ix2*RM2)*RM1;
    temp = r[j];

    /*printf("the ran1: %d %d\n",
ix1, ix2);*/

    return temp;
}

```

### The formulas used in the spreadsheet

```

x_1= 2.5
x_2= =E23
x_3= 2
x_4= 6.5
y_1= 3
y_2= 2
y_3= =E24
y_4= 3
r_1= 0.5
phi= =E12
theta= -9.0174693536082E-11
u_1= 2

```

To find the x\_2 and y\_3 information

```
xold= 4
yold= 2

x1= 2
y1= 0.5
theta1= =ATAN(Y1/X1)*180/PI()

h= =Y1/SIN(theta1*PI()/180)

rotated slot= 45
half rot slot= =0.5*rotated slot
1/2 l= =SIN(half rotated slot*PI()/180)*h
l= =2*1/2*l

theta2= =(180-rotated slot)/2
thetaneu= =theta2-theta1

del x= =COS(thetaneu*PI()/180)*l
del y= =SIN(thetaneu*PI()/180)*l

x_2= =xold+delx
y_3= =yold-dely
```

To find the x\_left and y\_left

```
dxold= 2
dyold= 2.5

length r-0= =SQRT(dxold^2+dyold^2)
length bot= =TAN(theta*PI()/180)*(lengthr-0)
angle 1= =ATAN(dyold/dxold)*180/PI()
angle 2= =(180-theta)/2

angle want= =180-(theta+angle1+angle2)

x_left= =0.5+COS(wanted_angle*PI()/180)*bottom_length
y_left= =0.5-SIN(wanted_angle*PI()/180)*bottom_length
```

To find x\_right and y\_right

```
base= 9

x_right =x_left+base*COS(theta*PI()/180)
y_right =y_left+base*SIN(theta*PI()/180)
```



$$\begin{aligned}
 & \text{hx} = \\
 & =x_1*\text{COS}(0*\text{PI}()/180)+y_1*\text{COS}(90*\text{PI}()/180)+r_1*\text{COS}(270*\text{PI}()/180+\text{theta}*\text{PI}()/180)+y_2*\text{COS}(270*\text{PI}()/180+\text{theta}*\text{PI}()/180)+x_3*\text{COS}(180*\text{PI}()/180+\text{theta}*\text{PI}()/180)+x_2*\text{COS} \\
 & (\text{theta}*\text{PI}()/180)+y_3*\text{COS}(90*\text{PI}()/180+\text{theta}*\text{PI}()/180)+u_1*\text{COS}(\text{theta}*\text{PI}()/180+\text{phi}*\text{PI} \\
 & )/180)+r_1*\text{COS}(\text{theta}*\text{PI}()/180+\text{phi}*\text{PI}()/180+90*\text{PI}()/180)+y_4*\text{COS}(270*\text{PI}()/180)+x_4* \\
 & \text{COS}(180*\text{PI}()/180)
 \end{aligned}$$

$$\begin{aligned}
 & \text{hy} = \\
 & =x_1*\text{SIN}(0*\text{PI}()/180)+y_1*\text{SIN}(90*\text{PI}()/180)+r_1*\text{SIN}(270*\text{PI}()/180+\text{theta}*\text{PI}()/180)+y_2* \\
 & \text{SIN}(270*\text{PI}()/180+\text{theta}*\text{PI}()/180)+x_3*\text{SIN}(180*\text{PI}()/180+\text{theta}*\text{PI}()/180)+x_2*\text{SIN}(\text{theta}*\text{PI} \\
 & )/180)+y_3*\text{SIN}(90*\text{PI}()/180+\text{theta}*\text{PI}()/180)+u_1*\text{SIN}(\text{theta}*\text{PI}()/180+\text{phi}*\text{PI}()/180)+r_ \\
 & 1*\text{SIN}(\text{theta}*\text{PI}()/180+\text{phi}*\text{PI}()/180+90*\text{PI}()/180)+y_4*\text{SIN}(270*\text{PI}()/180)+x_4*\text{SIN}(180*\text{PI} \\
 & )/180)
 \end{aligned}$$

$$\text{htheta} = 0+90+180+\text{theta}+0-90-180+90-\text{phi}+90+180-\text{theta}-\text{phi}-90-180$$

$$\text{phi} = (90-\text{htheta})/2$$

Error

$$\text{overall} = (0.5-x_{\text{left}})^2+(0.5-y_{\text{left}})^2+((0.5+9)-x_{\text{right}})^2+(0.5-y_{\text{right}})^2$$

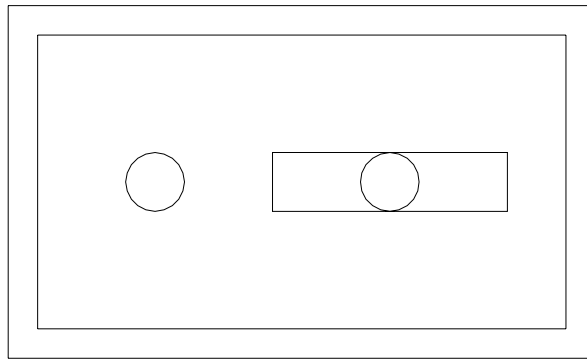
$$\text{sqrt} = \text{SQRT}(\text{overall})$$

Example results from the spreadsheet above

x_1=	2.5	To find the x_2 and y_3 information
x_2=	4.939339828	
x_3=	2	xold=
x_4=	6.5	yold=
y_1=	3	
y_2=	2	x1=
y_3=	0.732233047	y1=
y_4=	3	theta1=
r_1=	0.5	
phi=	45	h=
theta=	-9.01747E-11	
u_1=	2	rotated slot=
		half rot slot=
hx=	0	1/2 l=
hy=	-6.29461E-12	l=
htheta=	0	
phi=	45	theta2=
		thetaneu=
		del x=
		del y=
		x_2=
		y_3=
Error		To find the x_left and y_left
overall=	1.62243E-22	dxold=
sqrt=	1.27375E-11	dyold=
		length r-0=
		length bot=
		angle 1=
		angle 2=
		angle want=
		x_left=
		y_left=
		To find x_right and y_right
		base=
		x_right
		y_right

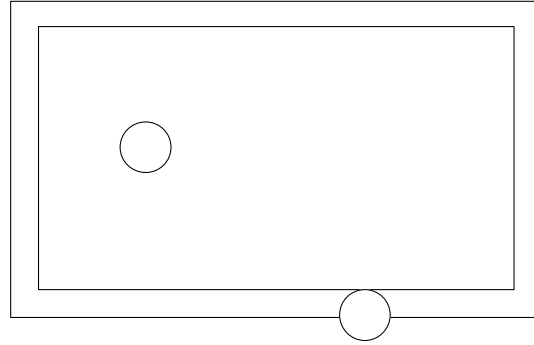
### **Background to the slotted block problem**

In industry, it is common to use slots in order to accommodate variation in a design. Some slots can be considered as passive smart assemblies because they absorb variation in dimensions. However, it is possible that a slot can over-constrain the problem, and perhaps lead to detrimental results. To show the effects of exact constraint vs. over-constraint on an assembly, the following design will be used.

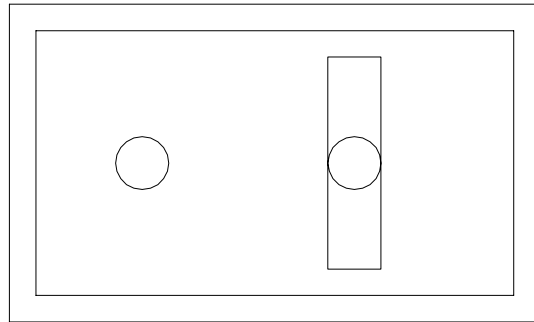


This assembly is composed of a base block with two pins rigidly connected. The top plate has a hole on the left side manufactured so as to always be able to fit over the pin. The tolerance variation of the hole is not under consideration in this problem. On the right of the plate is a slot. Again, the pin can fit perfectly between the edges of the slot if the part assembles. The variation in the size or width of the slot is not under consideration.

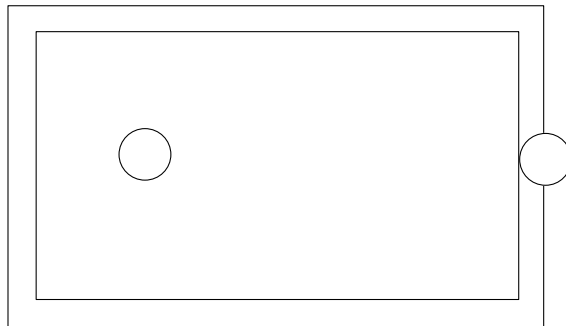
If the slot is at  $0^\circ$  (its nominal position), the assembly is exactly constrained. It is synonymous to the figure below.



However, if the slot is at  $90^\circ$ , as shown in the figure below, the assembly is now over-constrained in the x-direction and under-constrained in rotation. In other words, there may be some play in the block.



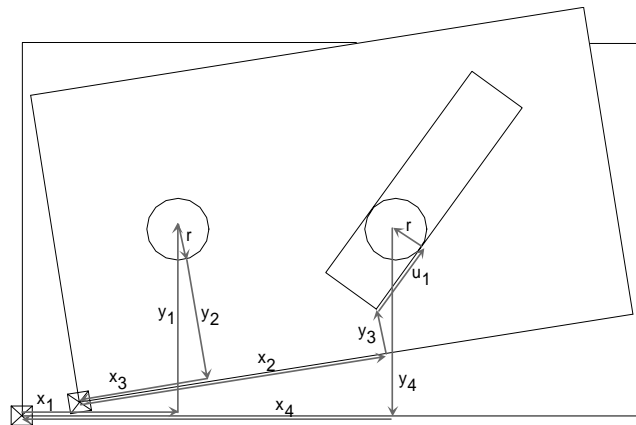
The assembly at  $90^\circ$  is synonymous with the figure below.



In this problem, the right pin in the slotted block is allowed to vary its position in the x-direction. The angle of the slot (user-defined) is allowed to vary between  $0^\circ$  and  $90^\circ$ . A Monte Carlo simulation will be run to determine if the block can assemble given the position of the right pin and the angle of the slot.

### Development of the problem

In order to determine if the block will assemble, the vector loop equations will be used. The assembly with the vector loops is shown below. The slot has been rotated  $45^\circ$ , and the plate is allowed to rotate in order to assemble.



Notice there is only one loop. It is assumed for this problem that there are 2 parts: the block and the plate. There are 2 joints, one revolute joint from the pin fitting in the hole, and one cylindrical joint where the pin touches only one side of the slot (with almost no clearance on the other side). Thus, according to the equation  $L = J - P + 1$ , there should only be one loop.

The vector loop equations that result from this setup are given below.

$$h_x = x_1 \cos(0) + y_1 \cos(90) + r \cos(270 + \theta) + y_2 \cos(270 + \theta) + x_3 \cos(180 + \theta) + x_2 \cos(\theta) + y_3 \cos(90 + \theta) + u_1 \cos(\theta + \phi) + r \cos(\theta + \phi + 90) + y_4 \cos(270) + x_4 \cos(180)$$

$$h_y = x_1 \sin(0) + y_1 \sin(90) + r \sin(270 + \theta) + y_2 \sin(270 + \theta) + x_3 \sin(180 + \theta) + x_2 \sin(\theta) + y_3 \sin(90 + \theta) + u_1 \sin(\theta + \phi) + r \sin(\theta + \phi + 90) + y_4 \sin(270) + x_4 \sin(180)$$

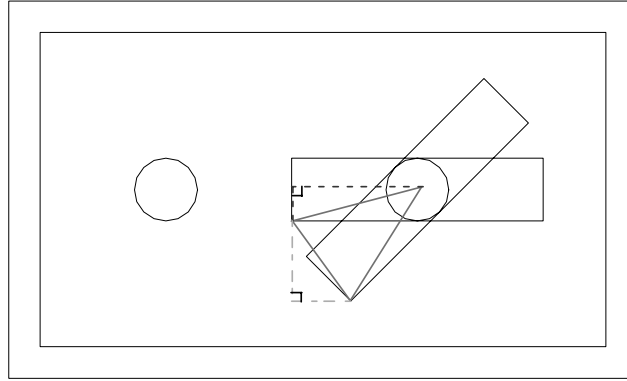
$$h_\theta = 0 + 90 + 180 + \theta + 0 - 90 - 180 + 90 - \phi + 90 + 180 - \theta - \phi - 90 - 180$$

Where  $\theta$  is the angle between the top plate and the base, and  $\phi$  is the angle of the slot.

The unknowns for this problem are  $u_1$  and  $\theta$ . Most of the other dimensions necessary for a solution are easily pulled from a CAD model of the assembly. However, in order to find  $x_2$  and  $y_3$ , some calculations need to be performed based on the angle of the slot.

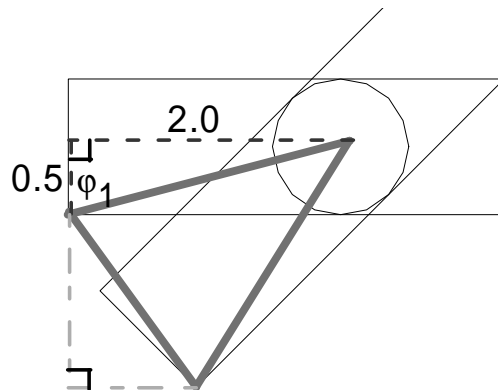
### Finding $x_2$ and $y_3$

In order to proceed with an analysis of this slotted assembly,  $x_2$  and  $y_3$  must be found. This can be done using geometries, and the derivation will be shown with the help of the following figure.



In this figure, the slot is shown in both the  $0^\circ$  position and the  $45^\circ$  position. To find both  $x_2$  and  $y_3$ , the location of the bottom left corner must be found for both slots. The bottom left corner for the slot at  $0^\circ$  is simply pulled from the CAD drawing. It is the distance from the left corner of the plate to the left bottom corner of the slot. Thus, for the slot at  $0^\circ$ ,  $x_2$  is 4, and  $y_3$  is 2.

However, to find these values for the  $45^\circ$  slot requires some calculation. The next figure helps to show that through the rules of a right triangle, the length of the upper sides of the bolded triangle can be found.



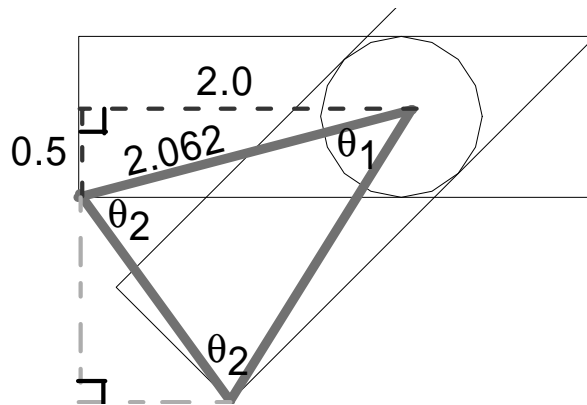
It is known that the distance from the center of the slot to the left edge is 2.0 in the x-direction and 0.5 in the y-direction. Thus, the hypotenuse of this upper triangle, which represents the side length of the bolded triangle, is simply

$$\text{hypotenuse} = \sqrt{2^2 + 0.5^2} = 2.06155$$

This length is also the length of the opposite side of the bolded triangle, relating the center of the slots to the corner on the 45° slot. Now, the angle marked as  $\phi_1$  must be found for future use. Again, from the geometry,

$$\phi_1 = \tan^{-1}\left(\frac{2}{0.5}\right) = 75.96^\circ$$

Now, consider the following figure. Because the slot has been rotated 45°,  $\theta_1$  must be 45°.

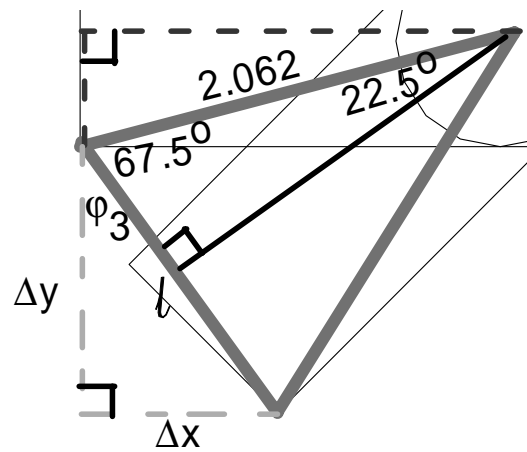




The  $\theta_2$ 's are the same angle, due to the nature of the triangle. They are simply computed, as shown below.

$$\theta_2 = \frac{180 - 45}{2} = 67.5^\circ .$$

To find the third side of the bolded triangle requires the next figure. The bolded triangle has been divided into two. Each of the known dimensions is listed in the figure. To find half the length of the side in question requires a simple calculation, shown below the figure.



$$\frac{1}{2}l = \sin(22.5^\circ) * 2.06155 = 0.7889$$

$$l = 1.5778$$

To find  $\varphi_3$ ,

$$\varphi_3 = 180 - 75.96 - 67.5 = 36.54^\circ$$

The change in the x and y-direction can now be found:

$$\Delta y = \cos(36.54^\circ) * 1.57784 = 1.2677$$

$$\Delta x = \sin(36.54^\circ) * 1.57784 = 0.9393$$

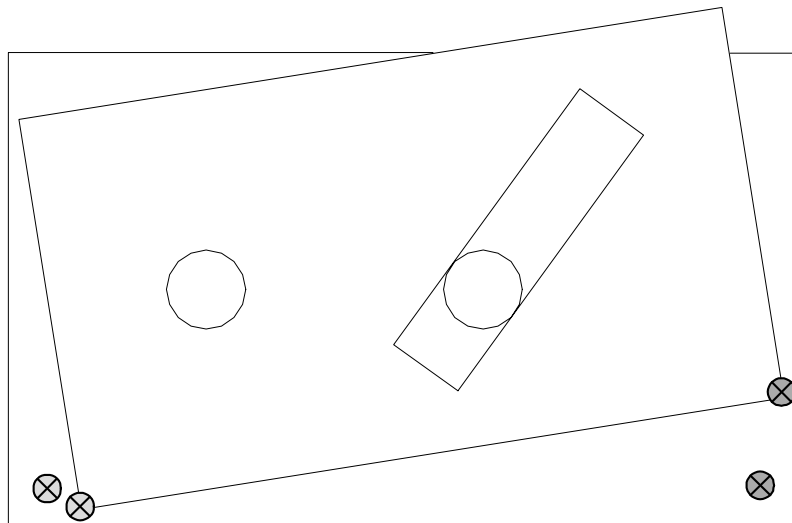
To now find  $x_2$  and  $y_3$ , simply add each change to the nominal positions. The following table shows all values for the independent variables.

Variable	value
$x_1$	2.5
$x_2$	$4 + \Delta x$
$x_3$	2.0
$x_4$	variable
$y_1$	3.0
$y_2$	2.0
$y_3$	$2 - \Delta y$
$y_4$	3.0
r	0.5

### Error

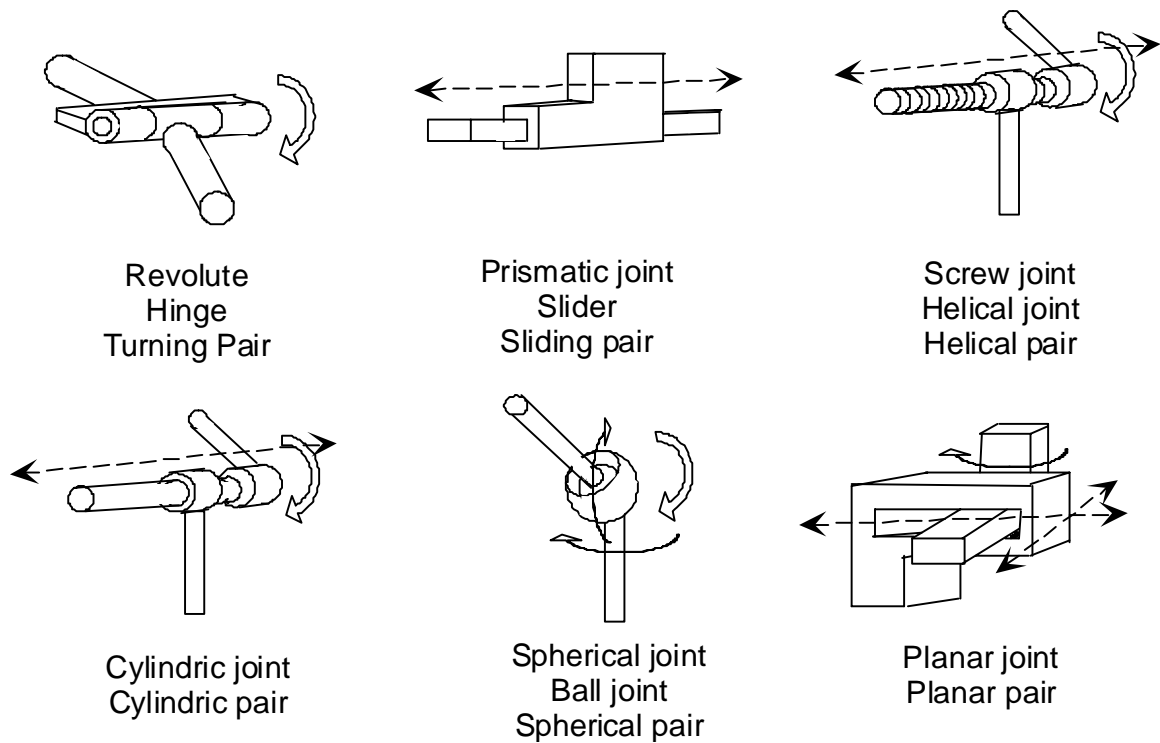
One final consideration before the Monte Carlo simulation is run is the error. The assembly is considered to have no error if the bases of the block and the plate are parallel. If there is any angle between them, the error is calculated as the square root sum of the squares of this difference between the nominal positions of the corners of the plate and the new positions of the corners of the plate.

Consider the following figure. The error is the square root of the sum of the squares for the difference in the x and y-positions of each corner.



## APPENDIX F 3D JOINTS IN ASSEMBLIES

While assemblies in 3D will not be treated in this thesis, it is important to note that Waldron and Kinzel [1999] demonstrate two types of 3D joints: lower pair joints and higher pair joints. Lower pair joints are those joints that keep all points of the mating surfaces of the links in contact. Higher pair joints only keep contact on isolated points or along line segments. They also suggest that there are six distinct lower pair joints, and an infinite number of higher pair joints (Fig. F.1).



**Figure F.1 – Six lower pair joints for 3D assemblies (after Waldron and Kinzel [1999]). The arrows represent the allowable degrees of freedom.**



**APPENDIX G                  RECIPROCAL RELATIONSHIP FOR FIG. 5.12**

**Table G.1 – Relationship between the equations of equilibrium and the DLM for the block with three constraints.**

	<b>C (Equilibrium)</b>	<b>B (DLM)</b>
More unknowns than equations 5 equations 6 unknowns	Over-constrained	Under-constrained
More equations than unknowns 5 unknowns 6 equations	Under-constrained	Over-constrained
Equal number of equations and unknowns 5/6 equations 5/6 unknowns	Exactly constrained OR	Exactly constrained OR
	Over-constrained: columns are linearly dependent OR	Under-constrained: column goes to zero columns are linearly dependent OR
	Under-constrained: row goes to zero column goes to zero	Over-constrained: row goes to zero rows are linearly dependent



## BIBLIOGRAPHY

- Adams, Jeffrey D. *Feature Based Analysis of Selective Limited Motion in Assemblies*. Master of Science Thesis, Massachusetts: Massachusetts Institute of Technology, 1998.
- Adams, Jeffrey D.; Whitney, Daniel E. "Application of Screw Theory to Constraint Analysis of Assemblies of Rigid Parts." *Proceedings of the 1999 IEEE International Symposium on Assembly and Task Planning*, Portugal, July, 1999.
- Adams, Jeffrey D.; Whitney, Daniel E. "Application of Screw Theory to Constraint Analysis of Mechanical Assemblies Joined by Features." *Journal of Mechanical Design: Transactions of the ASME*, Vol. 123, pp. 26-32, March 2001.
- Ball, Robert Stawell. *A Treatise on the Theory of Screws*. Cambridge: Cambridge University Press, 1900.
- Blanding, Douglass L. *Exact Constraint: Machine Design Using Kinematic Principles*, ASME Press, 1999.
- Chase, Kenneth W. "Chap. 13--Multi-Dimensional Tolerance Analysis." *Dimensioning & Tolerancing Handbook*, McGraw-Hill, 1999.
- Chase, Kenneth W.; Gao, Jinsong; Magleby, Spencer; Sorensen, Carl. "Including Geometric Feature Variations in Tolerance Analysis of Mechanical Assemblies." *IIE (Institute of Industrial Engineers) Transactions*, Chapman & Hall Ltd., pp. 795-807, 10 Oct 1996.
- Downey, Kristopher D. *A Formal Methodology for Smart Assembly Design*. Master of Science Thesis, Provo, Utah: Brigham Young University, December 2001.
- Downey Kristopher D.; Parkinson, Alan; Chase, Kenneth. "Smart Assemblies for Robust Design: A Progress Report." *Proceedings of 2002 ASME Design Automation Conference*, Paper DETC2002/DAC-34135, 2002.
- Evans, Chris. *Precision Engineering: An Evolutionary View*, Cranfield Press, 1989.
- Faerber, Paul. *Tolerance Analysis of Assemblies Using Kinematically-Derived Sensitivities*. Master of Science Thesis, Provo, Utah: Brigham Young University, 1999.



- French, Michael. *Form, Structure and Mechanism*, Springer-Verlag, 1992.
- French, Michael. *Conceptual Design for Engineers*, Springer-Verlag, 3<sup>rd</sup> edition, 1998.
- Gao, Jinson; Chase, Kenneth W.; Magleby, Spencer P. "Global Coordinate Method for Determining Sensitivity in Assembly Tolerance Analysis." *Proceedings of the ASME International Mechanical Engineering Conference and Exposition*, Anaheim, CA, Nov. 15-20, 1998.
- Hammond, Alisha M.; Parkinson, Alan R. "On the Robustness of Exactly Constrained Mechanical Assemblies." *Proceedings of 2003 ASME Design Automation Conference*, Paper DETC2003/DAC48840, 2003.
- Kamm, Lawrence J. *Designing Cost-Efficient Mechanisms: Minimum Constraint Design, Design with Commercial Components, and Topics in Design Engineering*, McGraw-Hill, 1990, reprinted by the Society of Automotive Engineers, 1993.
- Kriegel, Jon M. "Exact Constraint Design." *ASME International ME Congress and Exhibition (Winter Annual Meeting)*, Paper 94-WA/DE-18, pp. 1-9, November 1994.
- Kriegel, Jon M. "Exact Constraint Design." *Mechanical Engineering*, Vol. 117, pp. 88-90, May 1995.
- Konkar, Ranjit. *Incremental Kinematic Analysis and Symbolic Synthesis of Mechanisms*. Doctor of Philosophy Dissertation, Palo Alto, California: Stanford University, June 1993.
- Konkar, Ranjit; Cutkosky, M. "Incremental Kinematic Analysis of Mechanisms." *Journal of Mechanical Design*, Vol. 117, pp. 589-596, December 1995.
- Niven, W. D. "General Considerations Concerning Scientific Apparatus." *The Scientific Papers of James Clerk Maxwell: Vol 2*, Dover Publications, 1890.
- Parkinson, Alan. "Robust Mechanical Design Using Engineering Models." *Journal of Mechanical Design*, Vol. 117, pp. 48-54, June 1995.
- Pearce, Eric L. *Designing Active Smart Features to Provide Nesting Forces in Exactly Constrained Assemblies*. Master of Science Thesis, Provo, Utah: Brigham Young University, August 2003.
- Pearce, Eric L.; Parkinson, Alan; Chase, Kenneth. "On the Design of Nesting Forces for Exactly Constrained, Robust Mechanical Assemblies." *Proceedings of 2003 ASME Design Automation Conference*, Paper DETC2003/DAC48839, 2003.

- Phillips, Jack. *Freedom in Machinery. Volume 1, Introducing Screw Theory*, Cambridge: Cambridge University Press, 1984.
- Roth, Bernard. "Screws, Motors, and Wrenches that Cannot be Bought in a Hardware Store." *Robotics Research*, Chapter 8, pp. 679-693, 1984.
- Savoie, Troy. "Micronworm." Photo. Pre-2003. 26 Feb 2003  
<<http://web.mit.edu/savoie/www/portfolio/micronworm/index.html>>
- Skakoon, James G. *Detailed Mechanical Design: A Practical Guide*, ASME Press, 2000.
- Smith, Daniel K. *Constraint Analysis of Assemblies Using Screw Theory and Tolerance Sensitivities*. Master of Science Thesis, Provo, Utah: Brigham Young University, December 2001.
- Waldron, Kenneth J. "The Constraint Analysis of Mechanisms." *The Journal of Mechanisms*, Vol. 1, pp. 101-114. Great Britain: Pergamon Press, 1966.
- Waldron, Kenneth J.; Kinzel, Gary L. *Kinematics, Dynamics, and Design of Machinery*, New York: John Wiley & Sons, Inc., 1999.
- Whitehead, T. N. *The Design and Use of Instruments and Accurate Mechanisms: Underlying Principles*, Dover Publications, 1954.
- Whitney, Daniel E.; Mantripragada, R.; Adams, Jeffrey. "Use of Screw Theory to Detect Multiple Conflicting Key Characteristics in Complex Mechanical Products." *Proceedings of the ASME Design Engineering Technical Conferences*, Las Vegas, Nevada, 12-15 September, 1999.
- Wittwer, Jonathan W. *Predicting the Effects of Dimensional and Material Property Variations in Micro Compliant Mechanisms*. Master of Science Thesis, Provo, Utah: Brigham Young University, December 2001.
- Wittwer, Jonathan W.; Howell, Larry L.; Wait, Sydney M.; Cherry, Michael S. "Predicting the Performance of a Bistable Micro Mechanism Using Design-Stage Uncertainty Analysis." *Proceedings of the ASME International Mechanical Engineering Congress and Exposition*, New Orleans, Louisiana, Nov. 10-12, 2002.
- Willis R. *Principles of Mechanism*, London, 1841. 2<sup>nd</sup> Edition, London, 1870.