Development of Two-Dimensional Tolerance Modeling Methods for CAD Systems

ADCATS Report No. 88-7

Ki S. Chun, graduate student
Department of Mechanical Engineering
Brigham Young University

M.S. Thesis sponsored by ADCATS

December 1988

ABSTRACT

Research in the development of a two-dimensional geometric tolerance analysis modeler for CAD systems is described. The objective was to determine the procedures and data requirements for an interactive graphical assembly modeling tool which would permit the designer to graphically define 2-D vector assembly loops and create an assembly model database for quantitative tolerance analysis.

Various CAD system limitations were encountered which required compromises in the preferred geometric modeling capability and user interface design.

Principal contributions of this research:
1. The development of a generalized system for modeling 2-D mechanical assemblies for tolerance analysis, based on a kinematic representation of assemblies.
2. Successful incorporation of ANSI-Y14.5 feature control tolerances in the 2-D assembly model.
3. The coding of a prototype program as a conceptual design tool on a commercial CAD system.
4. The augmentation of the CAD data structure to include tolerance analysis data.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>viii</td>
</tr>
<tr>
<td>1.0 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation and Problem Statement</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Objective</td>
<td>6</td>
</tr>
<tr>
<td>1.3 Research Issues</td>
<td>7</td>
</tr>
<tr>
<td>1.4 Thesis Overview</td>
<td>8</td>
</tr>
<tr>
<td>2.0 Assembly Tolerance Model Theory</td>
<td>10</td>
</tr>
<tr>
<td>2.1 Vector Assembly Models</td>
<td>10</td>
</tr>
<tr>
<td>2.1.1 Open Loop</td>
<td>10</td>
</tr>
<tr>
<td>2.1.2 Closed Loop</td>
<td>11</td>
</tr>
<tr>
<td>2.2 Multi-loop Assembly Models</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Matrix Representation of Dimensions and Tolerances</td>
<td>14</td>
</tr>
<tr>
<td>2.3.1 Translational and Rotational Matrices</td>
<td>15</td>
</tr>
<tr>
<td>2.3.2 Sensitivities</td>
<td>16</td>
</tr>
<tr>
<td>2.3.3 Change propagation</td>
<td>17</td>
</tr>
<tr>
<td>2.3.4 ANSI-Y14 Feature Control Tolerances</td>
<td>18</td>
</tr>
<tr>
<td>2.4 Degrees of Freedom</td>
<td>18</td>
</tr>
<tr>
<td>2.5 Dependent Variables</td>
<td>18</td>
</tr>
<tr>
<td>3.0 A Kinematic Model for Assemblies</td>
<td>22</td>
</tr>
<tr>
<td>3.1 Mating Joint Types and Feature Interfaces</td>
<td>23</td>
</tr>
<tr>
<td>3.2 Joint Types and Corresponding Dependent Variables</td>
<td>24</td>
</tr>
<tr>
<td>3.3 Joint Sequence Method to Determine Dependent Variables</td>
<td>27</td>
</tr>
<tr>
<td>3.4 Model Verification and Lost Degrees of Freedom (DOF)</td>
<td>28</td>
</tr>
<tr>
<td>3.5 Summary of Loop Modeling Rules</td>
<td>34</td>
</tr>
<tr>
<td>3.6 Discussion</td>
<td>36</td>
</tr>
<tr>
<td>4.0 ANSI-Y14 feature controls</td>
<td>37</td>
</tr>
<tr>
<td>4.1 ANSI-Y14 Feature Control Accumulation</td>
<td>37</td>
</tr>
<tr>
<td>4.2 Including Feature Control Tolerances in Statistical Analysis</td>
<td>38</td>
</tr>
<tr>
<td>4.3 Matrix Representation of ANSI-Y14 Feature</td>
<td>39</td>
</tr>
<tr>
<td>4.3.1 Rotational Representation of ANSI-Y14 Feature</td>
<td>40</td>
</tr>
<tr>
<td>4.3.2 Translational Representation of ANSI-Y14 Feature</td>
<td>41</td>
</tr>
<tr>
<td>4.4 Matrix Sequence and Change Propagation of ANSI-Y14 Feature</td>
<td>42</td>
</tr>
<tr>
<td>4.5 Modeling Pin Joints</td>
<td>45</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Bike Crank Assembly with Tolerance Loop</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Vector Loop of the Bike Crank</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Interactive computer system for tolerance analysis</td>
<td>7</td>
</tr>
<tr>
<td>2.1</td>
<td>Dimensioned Block and Arc Assembly</td>
<td>12</td>
</tr>
<tr>
<td>2.2</td>
<td>A Simple Vector Assembly Model</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>An Assembly of Stacked Blocks</td>
<td>14</td>
</tr>
<tr>
<td>2.4</td>
<td>Example of Rigid Body Movement</td>
<td>17</td>
</tr>
<tr>
<td>2.5</td>
<td>Vector Loop 1 of Stacked Block Assembly</td>
<td>19</td>
</tr>
<tr>
<td>2.6</td>
<td>Vector Loop 2 of Stacked Block Assembly</td>
<td>20</td>
</tr>
<tr>
<td>2.7</td>
<td>Vector Loop 3 of Stacked Block Assembly</td>
<td>20</td>
</tr>
<tr>
<td>3.1.1</td>
<td>The Slider Joint</td>
<td>22</td>
</tr>
<tr>
<td>3.1.2</td>
<td>The Cam Joint</td>
<td>23</td>
</tr>
<tr>
<td>3.1.3</td>
<td>The Pin Joint</td>
<td>23</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Two continuation joints</td>
<td>24</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Dependent Variable Summary Chart</td>
<td>25</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Dependent Variable Summary Chart 'Cont.'</td>
<td>27</td>
</tr>
<tr>
<td>3.3</td>
<td>Table to Determine Dependent Vectors</td>
<td>28</td>
</tr>
<tr>
<td>3.4</td>
<td>The Three Reference Planes</td>
<td>29</td>
</tr>
<tr>
<td>3.5</td>
<td>2-D Axis of Freedom</td>
<td>30</td>
</tr>
<tr>
<td>3.6.1</td>
<td>DOF Removal Table</td>
<td>31</td>
</tr>
<tr>
<td>3.6.2</td>
<td>DOF Removal Table 'Cont.'</td>
<td>32</td>
</tr>
<tr>
<td>3.7</td>
<td>Joint Order Assembly</td>
<td>34</td>
</tr>
<tr>
<td>3.8</td>
<td>Vectors Continuing Through a Higher Sliding Joint</td>
<td>35</td>
</tr>
<tr>
<td>3.9</td>
<td>Block Sliding in Constrained Model</td>
<td>35</td>
</tr>
<tr>
<td>4.1</td>
<td>Compilation of ANSI-Y14 Feature Control Symbols</td>
<td>37</td>
</tr>
</tbody>
</table>
4.2 Feature Representation Chart ............................................. 39
4.3 The Rotational Variation Due to Tolerance Band ...................... 40
4.4 The Translational Variation Due to Tolerance ......................... 41
4.5 Sample Cam Assembly ..................................................... 42
4.6 Edge Joint Contact ......................................................... 44
4.7 Diagram of Pin Joint Model ............................................... 46
5.1 Sample of Pseudo Array ................................................. 51
5.2 Sample Geometric Error .................................................. 52
5.3 Location of all information on loop block ............................ 53
5.4 Sample CATS.BYU Neutral File ....................................... 72
6.1.1 The Over-Running Clutch Assembly ............................... 75
6.2.1 Sample Three Bar Truss Problem ................................... 81
6.2.2 Boxed end and rounded end pin joint ............................. 82
6.3.1 Sample bicycle crank assembly .................................... 86
6.3.2 The vector loop from the bicycle crank assembly ............... 87
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1.1</td>
<td>List of joints occurring in the over-running clutch</td>
<td>76</td>
</tr>
<tr>
<td>6.1.2</td>
<td>List of vectors occurring in the over-running clutch</td>
<td>78</td>
</tr>
<tr>
<td>6.1.3</td>
<td>DOF results for the over-running clutch assembly</td>
<td>78</td>
</tr>
<tr>
<td>6.1.4</td>
<td>ANSI-Y14 features in the over-running clutch problem</td>
<td>80</td>
</tr>
<tr>
<td>6.2.1</td>
<td>List of joints occurring in the three bar truss</td>
<td>83</td>
</tr>
<tr>
<td>6.2.2</td>
<td>List of vectors occurring in the three bar truss</td>
<td>83</td>
</tr>
<tr>
<td>6.2.3</td>
<td>DOF results for the three bar truss</td>
<td>84</td>
</tr>
<tr>
<td>6.3.1</td>
<td>List of joints occurring in the over-running clutch</td>
<td>88</td>
</tr>
<tr>
<td>6.3.2</td>
<td>List of vectors occurring in the over-running clutch</td>
<td>90</td>
</tr>
<tr>
<td>6.3.3</td>
<td>DOF results for the over-running clutch assembly</td>
<td>90</td>
</tr>
<tr>
<td>6.3.4</td>
<td>ANSI-Y14 features in the bicycle crank problem</td>
<td>91</td>
</tr>
</tbody>
</table>
1.0 Introduction

1.1 Motivation and Problem Statement

In mechanical assemblies optimizing the allocation of tolerances among individual parts of an assembly is an important design task. The designer must be aware that tight tolerances mean higher manufacturing costs and reduced production rates. In order to satisfy both the design constraint and the manufacturing constraint, the designer must perform a thorough tolerance analysis of the proposed assembly. Tolerance analysis is a tedious but necessary operation which assists in producing an optimal design for both functionality and producibility.

*CATS.BYU*

Brigham Young University has been the site for the development of the Computer Aided Tolerance Selection software named CATS.BYU. This on-going program under the direction of Dr. Kenneth W. Chase began in 1984. New algorithms, improved tolerance accumulation models, application of powerful optimization tools, and the extension of tolerance analysis to include process selection are some of the results of this research.

In addition to new tolerance analysis software, there has been a parallel development of one-dimensional tolerance modeling interfaces to CATS.BYU. A model interface is custom interactive graphic modeling software on a CAD system for creating vector models for tolerance analysis. As a result of these parallel efforts, interfaces have been designed for three major CAD systems:

1) Hewlett-Packard ME-10
2) GE Calma/DDM
3) IBM/CADAM
The HP interface, HP-CATS, was written using the ME-10 Macro language. HP-CATS allows the user to digitize points directly from an assembly drawing to create a tolerance loop. All selected data are then stored in a data file which is converted to a neutral file format and sent to CATS.BYU for analysis. HP-CATS is easy to use and is very effective in performing 1-D tolerance analysis. [2, 3]

A similar tool for modeling 2-D tolerance loops is needed. 2-D analysis is much more difficult, primarily because of the difficulty in creating the models and determining the tolerance sensitivities. An interactive, graphic modeling interface would encourage designers to do a more thorough job of tolerance analysis of complex geometries, and permit the application of powerful statistical optimization and other tools available in CATS.BYU.

Many tolerance analysis problems encountered in industry are one-dimensional by nature which means that the tolerance accumulation model consists of colinear vectors (e.g. the shaft end-play in a gearbox assembly). Many other problems require two-dimensional tolerance analysis. The two-dimensional stacks are comprised of vectors which are not necessarily co-linear.
Figures 1.1 and 1.2 illustrate a simple 2-D assembly and corresponding vector assembly model for a bicycle crank and shaft assembly. The resultant dimension, R, is determined by assembling the component parts. It is not a machined dimension. The vector loop represents those component dimensions which govern the magnitude of R. The vectors must be controlled part dimensions, i.e., they must appear as dimensioned quantities on the engineering or CAD drawing.
The tolerance or variation in dimension $R$ is critical to the performance of the assembly. If, due to variations in the magnitudes of the component dimensions, the resultant length of $R$ should be zero or negative, the wedge pin could not be pulled up snugly and the crank would fall off the shaft. By applying a mathematical model for tolerance accumulation, the contribution of each component can be determined. The component tolerances can then be assigned values which will assure that the value of $R$ stays between prescribed design limits. Such tolerance accumulation or tolerance "stackup" analysis is a principal tolerance analysis function, to be carried out thoroughly and carefully by the designer.

Two common models used by designers to estimate tolerance accumulation are:

\[
\text{Worst Case} \quad dR = \sum_{i=1}^{n} \left| \frac{\partial f}{\partial x_i} \right| dx_i \quad \text{Eq. 1.1}
\]

\[
\text{Statistical Case} \quad dR = \sqrt{\sum_{i=1}^{n} \left( \frac{\partial f}{\partial x_i} \right)^2 dx_i} \quad \text{Eq. 1.2}
\]

where $x_i$ are the component dimensions which assemble to produce the resultant $R$.

$dx_i$ are the maximum allowed variations in the $x_i$ (tolerances).

$f$ is the assembly function for computing $R$.

$\frac{\partial f}{\partial x_i}$ is the "tolerance sensitivity" or the weight factor determining the contribution by each $x_i$.

A comprehensive discussion of tolerance analysis using Eq. 1.1 and 1.2 may be found in References 1, 5, 6, and 7. However, the focus of this thesis is on the vector assembly models used in 2-D tolerance analysis.

There is very little information on 2-D tolerance analysis in the literature. Fortini [7] has a chapter on 2-D tolerance analysis, but he only treats single loop problems. Eaton
[6] describes a detailed study of a camera mechanism. A computer program was written and an attempt was made to generalize the method, but it is incomplete. Bjorke [1] addresses the multiple looped tolerance analysis problem, but he does not discuss in detail the modeling techniques required to generate assembly loops for all types of assemblies.

This research incorporates the contributions of the previously mentioned authors with newly developed modeling techniques and rules to create a complete 2-D assembly modeler. Emphasis has been placed on geometric modeling techniques which are suitable for adaptation to CAD systems.

A CAD Tolerance Modeling System

The advantage of tolerance modeling on a CAD system is that all the part drawings may be combined into one drawing and overlaid to create an assembly drawing. Vector loops may be created interactively and overlaid on the assembly drawing. The equations for tolerance analysis may then be generated directly from the vector assembly models.

Performing 2-D tolerance analysis on a CAD system has several innate advantages:

1. The designer does not have to generate the algebraic equations to model the 2-D assembly.

2. Model data may be selected graphically from the CAD system. Numerical data may then be extracted directly from the CAD database using a macro application software.

3. Considerably less numerical keyboard entry is required, which would lessen the chance of human error.

4. The graphical models of the assembly loops permit the designer to visualize the entire assembly more easily.
5. A graphical modeler would allow the designer to store the 2-D tolerance analysis models as part of the CAD database.

6. Since the interface is written using the CAD system's programming language, the designer may access most of the CAD system's functions, such as zoom, pan, new window, etc., while running the interface.

7. There would be less training required, since the designer would already be familiar with the CAD system.

1.2 Objective

The objective of this thesis is to investigate methods of automating a major portion of the tolerance analysis and specification tasks of the designer. This will be done by integrating tolerance analysis software into a commercial CAD system. The main intent is to determine the procedures and data requirements for an interactive graphical assembly modeling tool which will permit the designer to graphically define 2-D assembly loops and create an assembly model database for quantitative tolerance analysis.

A prototype modeling system will be developed on an HP ME-10 CAD system to demonstrate the modeling principles and techniques resulting from this research. Various user interfaces will be experimented with to determine the best interactive design interface. Actual tolerance analysis will be performed by CATS.BYU. Model data will be passed to CATS.BYU by means of a neutral file as shown in Figure 1.3. Following the analysis by CATS.BYU, revised data may be passed back to the CAD system via the same neutral file.
1.3 Research Issues

The following are the issues that needed to be investigated and resolved.

Theory:

1. Create a general system for representing an assembly, both geometrically and mathematically, using kinematic models.

2. Develop a set of modeling rules for the creation of systems of multiple loops and constraints for complex assemblies.

3. Determine a method for verifying that assembly models are well posed and complete.

4. Develop models and procedures for incorporating ANSI-Y14 feature control tolerances into the assembly tolerance analysis.

CAD system:

1. Determine what CAD functions are needed to create the 2-D graphical modeler.
2. Define a data structure for storing tolerance loop data in the CAD geometric database.

3. Establish a procedure for making tolerance analysis models and the resulting analysis an integral part of the engineering drawing documentation.

4. Define a neutral data base format to pass tolerance loop data to CATS.BYU once a valid assembly model has been constructed.

1.4 Thesis Overview

An overview of the research results follows:

Chapter 2. Assembly Tolerance Model Theory

Description of the basic modeling theories of 2-D tolerance analysis. Open and closed vector loops and a matrix method of representing them. Methods of treating complex assemblies requiring systems of loop equations, including constraints and dependent variables.

Chapter 3. A Kinematic Model for Assemblies

Kinematic joint representation of assembly pairs and assembly constraints. Rules for creating assembly model will be described in this chapter. This includes Degree of Freedom (DOF) analysis and design rules to help the designer setup valid assembly models.

Chapter 4. ANSI-Y14 Feature Control

Description of ANSI-Y14 feature controls and techniques for including them in assembly models. Explanation of translational or rotational matrix representations of ANSI-Y14 feature variations and the effects of joint type.
Chapter 5. ME-10 Macro Implementation

A prototype CAD program for creating vector assembly models was developed to test the theories and concepts. Features of ME-10 the Macro language and tools required to implement the theories into code are discussed in this chapter. Macro and database limitations are also discussed.

Chapter 6. Case Studies

Three 2-D tolerance models of mechanical assemblies are reviewed. A detailed explanation of modeling techniques employed is presented.

Chapter 7. Conclusions and Recommendations

Summary of the research and recommendations for future work are presented.

Appendix

The User's Manual and Programmer's Manual to HP-CATS 2.0 will be included in this section.

This research was carried out under the direction of Dr. Kenneth W. Chase with funding from ADCATS (Association for the Development of Computer-Aided Tolerancing Software).
2.0 Assembly Tolerance Model Theory

2.1 Vector Assembly Models

Two-dimensional tolerance analysis may be performed using vectors to represent an assembly. Each vector represents one component dimension. Vectors are added tip-to-tail to simulate assembling parts together. The magnitude of each vector is equal to the nominal length of a dimension. The direction of each vector is determined by its relative angular position to the previous vector. This relative referencing is essential to the ability of the model to describe real effects in assemblies. In the case of the first vector of the loop, the reference angle is taken from the absolute reference frame at the tail of the vector. When the vectors are joined head-to-tail around an assembly, this is termed a vector loop. This method of modeling assemblies for tolerance analysis will be termed Vector Assembly Models (VAM).

The resultant sum of a vector loop is generally a critical clearance or critical dimension of the assembly it represents. Summing the nominal values of the vectors results in the nominal or average clearance. Summing the tolerances of the vectors by worst case or statistically (Eq. 1.1 and 1.2) determines the variation in the clearance.

There are two types of vector loops that arise in tolerance analysis: 1) open loops and 2) closed loops.

2.1.1 Open Loop

The open loop is primarily used to find the clearance variation of a given assembly. To begin the open loop, the designer digitizes a point on one side of the clearance, continues with each vector around the assembly and ends the vector loop on the opposite side of the clearance. The resultant vector, which connects the first vector and the last vector in the loop, is the nominal clearance dimension.
Each open loop introduces three relationships: 1) the sum of all the x-components of the vectors must equal the x-component of the clearance, 2) the sum of all the y-components of the vectors must equal the y-components of the clearance, 3) the sum of the relative angles must equal zero. These three relations may be solved for three unknowns.

2.1.2 Closed Loop

The closed loop arises when there is an adjustable element in an assembly that always reduces the clearance to zero. In a closed loop, the vector loop begins and ends at the same point. As explained in the earlier part of this chapter, each vector is oriented relative to the vector. This relative positioning is especially important in the closed loop. By taking all measurements relative to the previous vector, a change in one vector will not alter the relationship between vectors. It does, however, allow the effects of changes to propagate through an assembly to alter the final tolerance.

Each closed loop introduces three relationships: 1) the sum of all the x-components of the vectors must equal zero, 2) the sum of all the y-components of the vectors must equal zero. 3) the sum of all the relative angles must equal zero. These three relationships may be solved for three unknowns.

2.2 Multi-loop Assembly Models

Frequently, 2-D assemblies require a combination of open and closed loops to model the assembly relationships. The two part assembly in Figure 2.1 is a simple illustration of a non-linear tolerance model. It represents a block resting in a circular trough.
Figure 2.1 Dimensioned block and arc assembly.

The basic dimensions of the block and arc are given in Figure 2.1. The height H of the assembly is wanted but is not a machined dimension. The average value and the tolerance depend on the dimensions of the block and arc. Thus, H is the "dependent variable," while B, C, D and E are the "independent variables."

Figure 2.2 shows two vector loops which describe the assembly. The first vector loop is an open loop consisting of vectors 1, 2, and 3. This vector loop defines the relationship between the datum A and the top of the block. The second vector loop is a closed loop consisting of vectors 2, 4, and 5. This is a constraint loop. It assures closure of the assembly regardless of variations in the size of the block or the arc.
Each vector in the two vector loops is equivalent to a dimension of a component part of the assembly, $1 = C$, $2 = D/2$, $3 = B$, $4 = E$, $5 = D/2$. The component tolerances accumulate to determine the variation in the height $H$. Note that Vector 2 is shared by both loops. This is what ties the two loops together, forming a system of equations which must be solved simultaneously. The two vector loops contain six scalar unknowns: $H_x$, $H_y$, $\theta_2$, $\theta_3$, $\theta_4$, $\theta_5$. These are determined from an accurate CAD layout. The variations in these quantities, which are required for tolerance analysis, may be determined by the Direct Linearization Method. Methods for solving this system of
equations are beyond the scope of this thesis. A thorough treatment of this subject is presented in another thesis [8].

2.3 Matrix Representation of Dimensions and Tolerances

In two dimensions a vector is described in terms of magnitude of its x and y components and rotation about the z-axis. This eliminates several factors encountered in dealing with 3-D vectors.

The convention that will be employed in the analysis is that each vector will be defined in a local coordinate system oriented with respect to the vector. In other words, the local coordinates will be rotated so that the x-axis points in the direction of the vector (see Figure 2.3).

![Diagram of stacked blocks]

Figure 2.3 An assembly of stacked blocks.
This relative reference system possesses some innate advantages which will be explained later in this chapter.

2.3.1 Translational and Rotational Matrices

Each vector in the vector loop is modeled as a product of a rotational and a translational matrix, as shown in Eq. 2.1. This represents the coordinate transformation from one relative coordinate frame to the next. This augmented 3x3 matrix form is commonly used in CAD systems, since it allows translation to be represented by matrix multiplication. It also requires an augmented 2-D vector \((x, y, 1)^T\) for the transformation matrix to operate upon.

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
\delta_x & \delta_y & 1
\end{bmatrix}
\begin{bmatrix}
\cos \theta & \sin \theta & 0 \\
-\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
= Eq. 2.1
\]

The stacked block problem shown in Figure 2.3, will be used as an example to illustrate the matrix representation of a vector model. As shown in Figure 2.3, the vector loop begins at the top right corner of Part 1. The coordinate axis is first rotated \([R_1]\) and then traverses along Part 1 \([T_{01}]\). The rotation matrix \([R_1]\) and translation matrix \([T_{01}]\) represent Vector 1. The coordinate axis is once again rotated \([R_2]\) and translated to the end of Part 2 \([T_{12}]\). The rotation matrix \([R_2]\) and the translation matrix \([T_{12}]\) represent Vector 2. Vectors 3, 4, and 5 are created in like manner. A final rotation \([R_6]\) is required to bring the final coordinates parallel to the original orientation at position 1.

The Clearance vector of this assembly is \(\vec{u} = [u_x, 0, 1]^T\). It is the resultant vector of the vector loop equation. The first vector \(\mathbf{1}\) is identified by rotating the global coordinate by 90° and translating the reference point along Part 1. The final equation of this vector loop is:
The product of the chain of matrix operators represents the vector loop equation. It is equivalent to globally summing the x and y components of the vectors and summing the relative rotation angles. Note that the sum of the $\theta_i$ is zero.

2.3.2  Sensitivities

An advantage of the matrix representation of vector assembly models is that the sensitivities may be obtained numerically by matrix multiplication. Suppose, for example, the sensitivity of the clearance $\bar{\Delta}$ to changes in Vector 3 is wanted in the stacked block example. By making a small change in $\delta x_3$ and performing the matrix multiplication, the resulting changes in the components of $\bar{\Delta}$ may be calculated. The ratios $\Delta u_x/\Delta x_3$ and $\Delta u_y/\Delta x_3$ approximate the sensitivities.
2.3.3 Change propagation

![Diagram of rigid body movement](image)

Figure 2.4 Example of rigid body movement.

Another advantage of the matrix model is that the matrices are able to correctly model the effects of variation or change occurring anywhere in the assembly. This phenomenon has been termed "Change Propagation." If the contact surface between Part 1 and Part 2 experiences an angular variation due to normal production variations, there would be a rigid body movement starting at the contact surface and propagating through the entire assembly (see Figure 2.4). The permissible rotational variation is controlled by specifying the ANSI-Y14 feature tolerance for perpendicularity of surfaces involved.

This feature is very desirable since the change which occurs at the beginning of the matrix sequence will affect the final clearance of the assembly. In Figure 2.4, the rotational variation which occurs at the joint between Parts 1 and 2 is highly visible in the final clearance. Change propagation also takes into account the translational and rotational variations due to other ANSI-Y14 feature controls.
2.3.4 ANSI-Y14 Feature Control Tolerances

Another advantage of the matrix representation is that ANSI-Y14 features may be added to the assembly as was illustrated in the previous example. The modeling and implementing of ANSI-Y14 features will be discussed in Chapter 4.0.

2.4 Degrees of Freedom

In the two-dimensional plane a body has three degrees of freedom: the translation along the x-axis, y-axis, and rotation along the z-axis. Using these relationships each vector loop may solved for three unknowns. In order to determine if the vector model is valid or not, the degrees of freedom in the assembly are determined. Zero degrees of freedom must be obtained before analysis can be performed. In order to achieve zero degree of freedom it is necessary to constrain each part sufficiently. A complete set of joint types and degrees of freedom removed by each has been charted and discussed in the following chapter.

2.5 Dependent Variables

The determination of dependent vectors or nominal dimensions with no associated tolerance are also an important facet of 2-D tolerance analysis. Each vector loop can only solve for three unknown variables; therefore, there must be enough loop equations to solve for all the dependent variables.

Figure 2.5 illustrates a stacked block assembly with the first vector loop of the assembly overlaid on the assembly drawing. Each vector bears a double subscript notation. The first number in the notation X_{11} denotes the loop number, and the second number denotes the vector number, so the notation X_{11} is read "Vector 1 of Loop 1." The Vectors X_{11} and X_{12} are independent vectors since they correspond to dimensioned lengths of Part 1. But Vectors X_{13}, X_{14}, and X_{15} are dependent vectors. The nominal dimension
of these vectors can be extracted from the CAD system, but there is no tolerance which is associated with their dimensions. The nominal length of Vector of $X_{13}$ is dependent on the point of contact between Part 1 and Part 2. Vector $X_{14}$ is dependent on the contact point of Part 2 and Part 3. Vector $X_{15}$ depends on the contact point of Part 3 and Part 1.

![Diagram](image)

Figure 2.5 Vector loop 1 of stacked block assembly.

There are two dependent vectors in the second vector loop shown in Figure 2.6. Vectors $X_{22}$ and $X_{25}$ are dependent upon the location of two contact points of Part 2 on Part 3. The nominal dimensions of these two vectors is determined numerically from the CAD system, but there are no specified tolerances for these two vectors. In order for CATS.BYU to allocate tolerances to a clearance, each vector tolerance must be known. By defining the dependent variables before analysis by CATS.BYU, CATS.BYU will know how to allocate tolerances to the clearance.
In Figure 2.7 vectors $X_{31}$ and $X_{35}$ are dependent vectors. Note that Vectors $X_{24}$ and $X_{25}$ are the same dependent dimension, as are $X_{22}$ and $X_{35}$. These are the shared vectors that tie the three loops into a single system that must be solved simultaneously. The nominal values of the four dependent lengths may be found directly from the CAD layout, but the variations in their lengths are unknown.
There are also five dependent angles indicated the three loops: $\theta_{12}$, $\theta_{34}$, $\theta_{51}$, $\theta_{23}$, $\theta_{45}$ and $\phi_{12}$. However, since the variation in $\theta_{34}$ is the same as $\theta_{51}$, there are only four dependent angular variations. Adding these to the five dependent length variations, brings the total unknowns to nine. All nine are functions of the independent variables as expressed in the three loop equations. Since each loop equation can solve for three unknowns, the system is sufficient for the solution of all nine values.

A systematic procedure for setting up loops, identifying dependent variables and assuring a sufficient system of loop equations is described in the next chapter.
3.0 A Kinematic Model for Assemblies

This chapter discusses the systematic method for creating vector assembly models, based on representing an assembly as a kinematic model. This process models mating part contacts between assembly pairs as kinematic joints and automatically identifies the dependent variables resulting from the assembly. Degree of freedom analysis is performed to determine when a system is sufficiently constrained. A set of modeling rules for setting up vector loops is also formalized.

3.1 Mating Joint Types and Feature Interfaces

In order to automate the model verification process and to find dependent vectors a complete set of 2-D joint definitions is required. All 2-D mating part joints can be generalized into three primary joint types, as shown in Figures 3.1.1, 3.1.2, and 3.1.3. Several variations of the three primary joints will be discussed later in this chapter. In 2-D, the three primary joint types are: 1) the Slider joint, 2) the Cam joint, and 3) the Pin joint.

![Figure 3.1.1 The slider joint.](image)

Figure 3.1.1 shows a slider or a planar joint. This joint makes surface contact with a reference plane, which appears as a line contact in a 2-D drawing.
Figure 3.1.2 The Cam joint.

Figure 3.1.2 is a cam joint which consists of a ball or a rod making contact with a reference plane. In two dimensions the contact made between the two parts is a point. In three dimensions the contact might be a point or a line. In kinematics this is called "higher sliding" contact because the part can rotate about the point of contact as it slides. A variation of this joint type, having the same geometry, is pure rolling contact or rolling without slipping.

Figure 3.1.3 The Pin joint.

Figure 3.1.3 shows a pin joint consisting of two parts held by a pin, fastener, or a shaft and bearing assembly. This is treated as rotation about a single point. However, a detailed analysis of a pin in a hole with clearance reveals that the internal contact is also higher sliding.
Two additional joint types are created for convenience in modeling assemblies. These are non-moving joints called "continuation joints." A continuation joint does not have a mating part, but is a joint which provides a reference point within a part. They permit all the vectors in a vector loop to be controlled part dimensions. There are only two continuation joint types: the "center" joint and the "rigid" joint. The center joint allows the vector to continue from a radial point of a circle to the center and back out to another point on the surface. The rigid joint type allows the vector loop to turn a corner of a polygon (see Figure 3.1.4).

3.2 Joint Types and Corresponding Dependent Variables

As the vector loop is created through the assembly, dependent vectors and angles will be created due to assembly. Many of these vectors will have a nominal dimension which can be determined from the CAD system, but will not have a specified tolerance. This non-dimensioned quantity may be a length (dependent vector) or an angle (dependent angle). Its tolerance must be solved using the vector loop equations. Therefore, it is important to identify the dependent vectors prior to analysis by CATS.BYU. Figures 3.2.1 and 3.2.2 are tables summarizing the different joint types with their associated dependent variables.
### Joint Types and Corresponding Dependent Variables

<table>
<thead>
<tr>
<th>Joint Type</th>
<th>U (Vector)</th>
<th>φ (Angle)</th>
<th>θ (Prompted Info.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planar</td>
<td>✓</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>Constrained Planar</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Higher Sliding</td>
<td>—</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>Center</td>
<td>—</td>
<td>✓</td>
<td>—</td>
</tr>
</tbody>
</table>

**Figure 3.2.1** Dependent variable summary chart.

A dependent vector is a nominal dimension appearing on the assembly drawing without an associated tolerance. Figures 3.2.1 and 3.2.2 illustrate those joint types which require a dependent vector, and are represented by "U" in the figures. A dependent angle is
a non-specified angle formed by two vectors and is indicated by a "φ" in the figures. The third variable is not a dependent variable, but is input by the designer. This is an orientation angle to which the dependent angle or vector will be referenced and is identified by "θ" in the figures.

In Figure 3.2.1, the Planar joint must have a reference angle θ identified. This reference angle is the direction which the part will slide if the part is not constrained and is identified by selecting the two end points of the Planar joint. The vector locates the distance between two datum references on the two sliding planes.

A higher sliding joint requires a U-vector to locate the contact point and an orientation angle θ. Due to the curvature at the contact surface, the Higher Sliding joint always has a vector normal to the contact plane passing through the center of curvature. Due to this characteristic, the Higher Sliding joint does not have an angular dependent variable.

The Edge joint in Figure 3.2.2 is a variation of the Higher Sliding joint in Figure 3.2.1. The Edge joint is a Higher Sliding joint with a zero radius of curvature. The center of the Edge joint rests on the contact point. In the Edge joint, the wedge may make contact with the plane at any angle. The angle between the contact plane and one face of the wedge is an angular dependent variable φ. The prompted angle θ is used as a reference indicating the sliding direction.
### Joint Types and Corresponding Dependent Variables

<table>
<thead>
<tr>
<th>Joint Type</th>
<th>U (Vector)</th>
<th>( \phi ) (Angle)</th>
<th>( \theta ) (Prompted Info.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Rolling W/O Sliding</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Pin (Free)</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Crank (Fixed)</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Rigid</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.2.2 Dependent variable summary chart continued.

#### 3.3 Joint Sequence Method to Determine Dependent Variables

An automated method to determine dependent vectors is termed the "Joint Sequence Method" (JSM). This method takes 729 possible joint combinations and determines the
dependent vectors by the order in which the joints occur in the vector loop. Three joints are sequentially examined and compared to the dependent vector table (shown in Figure 3.3). This table forms the basis of the Joint Sequence Method.

In the Joint Sequence Method, three variables must be determined for each sequence of joints: dependent vector $U$, dependent angle $\phi$, and prompted reference angle $\theta$. These are the identical variables described in the previous section. The JSM rule chart in Figure 3.3 shows three joints in sequence with the corresponding rule for determining dependent variables.

<table>
<thead>
<tr>
<th>JOINT SEQUENCE COMBINATION</th>
<th>RULE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Center - Slider - Not Center</td>
<td>2nd vector is $U$, global direction is $\theta$ and need to specify reference plane ($\theta$)</td>
</tr>
<tr>
<td>Center - Slider - Center</td>
<td>None</td>
</tr>
<tr>
<td>Not Center - Edge - Not Center</td>
<td>Need to specify $\theta$, $U$ is a vector in $\theta$ direction, $\phi$ is in the direction of the 2nd vector relative to the 1st.</td>
</tr>
<tr>
<td>Not Center - Edge - Center</td>
<td>$\phi$ is the relative angle.</td>
</tr>
<tr>
<td>Anything - Pin - Anything</td>
<td>$\phi$ is the relative angle.</td>
</tr>
<tr>
<td>Anything - Planar - Not Center</td>
<td>Need to specify $\theta$, $U$ is a vector in $\theta$ direction, and $\phi$ is the relative angle.</td>
</tr>
<tr>
<td>Anything - Center - Anything</td>
<td>$\phi$ is the relative angle.</td>
</tr>
<tr>
<td>Anything - Rigid - Anything</td>
<td>None</td>
</tr>
</tbody>
</table>

Figure 3.3 Table to determine dependent vectors.

### 3.4 Model Verification and Lost Degrees of Freedom (DOF)

The traditional method of determining degrees of freedom in kinematic mechanisms analysis is to count all parts in the assembly, count all joints in the assembly, and subtract the joint degrees of freedom from the part degrees of freedom [10]. This DOF model
works well for most cases, but cannot model an over-constrained assembly. An over-constrained assembly may show negative degrees of freedom in the assembly due to redundant members. Because of this limitation, an alternate method has been developed to remove assembly DOF and verify the assembly model.

![Diagram of three reference planes](image)

*Figure 3.4 The three reference planes.*

Consider the 3-D case of positioning a block by means of three datum planes as shown in Figure 3.4. In 3-D space a part has six degrees of freedom; $\delta_x$, $\delta_y$, $\delta_z$, $\theta_x$, $\theta_y$, and $\theta_z$. As the block is placed in contact with each plane in sequence, degrees of freedom (DOF) are removed. Three DOF are removed by the Primary reference plane, two DOF are removed by the Secondary reference plane and one DOF is removed by the Tertiary reference plane. Note that all three joints are planar. So, the number of degrees of freedom removed depends on whether the joint is a primary, secondary or tertiary joint. This sequential reference contact is termed "Contact Sequence" and will be explained in greater detail.
Figure 3.5 2-D axis of freedom.

In the 2-D plane there are only two axis in which an object may translate and one axis in which it can rotate (see Figure 3.5). This allows three degrees of freedom \( \delta_x, \delta_y, \) and \( \theta_z \) in 2-D. Therefore, the primary planar joint removes two DOF; the secondary removes one. This is sufficient to uniquely define the position of the block. Each of the 2-D joint types removes a characteristic number of degrees of freedom, depending on whether it is the primary, secondary, or tertiary joint on a part. After three degrees of freedom have been removed from a part, any additional joint is termed a "redundant joint" and removes no degree of freedom.

Since this method of removing degrees of freedom examines individual parts using Contact Sequence rules, it proves to be robust and is not bothered by redundant joints.

A special case of redundant joint is a "parallel joint," such as two parallel sliding planes acting on the same part or two higher sliding joints which act on the same cylindrical surface. The designer must identify such cases and may have to make adjustments in the model. Examples will be pointed out in Chapter 6.0.
The following Figures shows the DOF removed by each joint with respect to its occurrence in the Contact Sequence.

<table>
<thead>
<tr>
<th>Joint Type</th>
<th>Primary</th>
<th>Secondary</th>
<th>Tertiary</th>
<th>Parallel (redundent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>0</td>
</tr>
<tr>
<td>Planar</td>
<td>-2</td>
<td>-1 or -2</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>Constrained Planar</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>Higher Sliding</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>Edge</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 3.6.1 DOF removal table.
### Joint Order

<table>
<thead>
<tr>
<th>Joint Type</th>
<th>Primary</th>
<th>Secondary</th>
<th>Tertiary</th>
<th>Parallel (redundent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rolling W/O Sliding</td>
<td>-2</td>
<td>-1 or -2</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>Pin (Free)</td>
<td>-2</td>
<td>-1 or -2</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>Crank (Fixed)</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>0</td>
</tr>
<tr>
<td>Rigid</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>0</td>
</tr>
<tr>
<td>Center</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 3.6.2 DOF removal table continued.

The Planar joint in Figure 3.6.1, may occur on a plane or an arc. When a Planar joint occurs on an arc, it is treated as a slider joint. When a block is placed on the primary reference plane, the block immediately loses two degrees of freedom. It no longer can translate on an axis or rotate in the z-axis. The secondary contact removes the remaining DOF. There will be other joints in which the secondary planar contact joint will remove two DOF. This difference in DOF removed is dependent on the direction in which the vector is being created, or rather the order in which the joints are applied to a given part.
The Rolling without Sliding joint in Figure 3.6.2, is analogous to a ball bearing or a rod rolling on a rubber mat. The part is only allowed to roll without slipping, and may be used where the surface friction is assumed sufficient to prevent the roller from sliding. And like the Planar joint, the Rolling without Sliding joint may remove either one or two DOF depending on the contact order.

A simple example of Contact Sequence analysis is shown in Figure 3.7. The assembly consists of a ball bearing sitting on a bearing surface with a retainer making contact on one side of the ball bearing. Vector 1 extends from the contact point on Part 1 to the center of Part 2. Vector 2 extends from the center of Part 2 to the contact point between Part 2 and Part 3.

Part 1 is considered the frame and therefore has no degrees of freedom. In Figure 3.7 the first joint to make contact with Part 1 in Figure 3.7 is the Roller joint. Therefore, the primary contact on Part 2 is the roller joint, which removes two degrees of freedom. The Center joint removes no degrees of freedom, but continues the vector loop through Part 2. The second joint on Part 2 is the Edge joint. The Edge joint becomes the secondary contact to Part 2 and the primary contact to Part 3. It removes one degree of freedom from each part. Part 2 is therefore fully constrained. Any further joint contacts on Part 2 would be redundant and remove no more DOF. Part 3 still has two degrees of freedom remaining. Additional constraints would be needed to define its position.
3.5 **Summary of Loop Modeling Rules**

This section deals with the rules which will guide the designer in modeling assemblies properly. These key rules will help the developer create valid tolerance assembly models:

1. **Vectors must cross the boundary between mating parts.** Each time a vector enters a mating part joint the vector cannot proceed along the same part. It must cross the boundary and include the mating part in the vector loop.

2. **For joints involving a sliding plane, one of the connecting vectors must be along the plane.** In part (A) of Figure 3.8, a ball lies on a surface. The vertical vector makes contact with the plane to a datum reference. The next vector must continue along the plane. In part (B) of Figure 3.8, the ball is resting on a curved surface. In this case, a vector comes from the center of the ball to the contact point and the next vector returns to the center of the bowl.
3. Controlled or tolerated dimensions should be used as vectors, if at all possible.

4. Every part in the assembly must have at least one vector through it. If there are five parts, each part must have at least one vector cross the boundary into it.

5. There cannot be any isolated loops. In multiple loop assemblies, each vector loop must share at least one common vector with another loop. Isolated loops may be analyzed as a separate assembly.

6. Zero Degree of Freedom must be achieved in the assembly model before analysis is performed.

7. Parallel joints remove no degree of freedom and the designer must be aware of such joints. The Sliding Block assembly shown in Figure 3.9, is an example of a parallel joint. If the contact joints on Part 2 are not sufficiently constrained, Part 2 may slide.

Figure 3.9 Block sliding in constrained model.
8. The vector loops should be completed before the ANSI-Y14 feature controls are added.

Examples of the application of these rules to the analysis of assemblies are found in Chapter 6.0.

3.6 Discussion

Strength

All of the modeling tools, including joint types, Degree of Freedom analysis to verify valid assembly models, determining dependent vectors, and modeling rules have been generalized. This means that by implementing these tools any assembly may be modeled properly. The real strength of this modeling method is that the procedure is general enough to model almost any assembly.

Weakness

Although the Contact Sequence Method of removing DOF can be conducted in parallel with the creation of the vector loops, a complete check of degrees of freedom cannot be implemented until the Vector Assembly Model is created. There is no way to assure that an individual vector will be part of a valid Vector Assembly Model until the Vector Assembly Model is complete. A check for conformance to modeling rules can be made, but it is still possible to create invalid systems of loops. This inability to perform interactive vector verification may result in an iterative approach to Vector Assembly Model creation.
4.0 ANSI-Y14 feature controls

4.1 ANSI-Y14 Feature Control Accumulation

The ANSI-Y14 standard is a set of feature control tolerances which may be added to an engineering drawing to control form and feature relationships. These are shown in Figure 4.1.

Figure 4.1 Compilation of ANSI-Y14 feature control symbols.

It was determined that in 2-D, the profile of a line and a surface was similar enough to combine. The two features were combined and called "profile." The same was decided about circular and total runout feature controls and straightness. This reduced the number of feature controls to be implemented in the 2-D interface to 11. These 11 ANSI-Y14 features were then analyzed to determine the method by which they would be represented in the VAM. The representation was found to depend on joint type. For some joints a particular feature control produces an independent size variation. For other joints it produces an angular variation. A table summarizing these results is shown in Figure 4.2. In order to implement ANSI-Y14 features in a tolerance loop analysis each feature may be represented either as a rotation or translation matrix.
4.2 Including Feature Control Tolerances in Statistical Analysis

\[ T_i = \sqrt{\sum_{i=1}^{n} \left( \frac{\partial f}{\partial x_i} dx_i \right)^2 + \sum_{i=1}^{n} \left( \frac{\partial f}{\partial \alpha_i} d\alpha_i \right)^2} \]  
Eq. 4.1

The equation used in allocation of tolerances is shown in Eq. 4.1. The partial derivative \( \frac{\partial f}{\partial x_i} \) in the dimensional tolerance equation is the sensitivity of the nominal dimension vector and the partial derivative in the ANSI-Y14 tolerance equation \( \frac{\partial f}{\partial \alpha_i} \) is the sensitivity of the feature control.
### 4.3 Matrix Representation of ANSI-Y14 Feature

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Joint Type Feature Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>[T]</td>
<td>[R]</td>
<td>[T]</td>
<td>[R]</td>
<td>[T]</td>
<td>[R]</td>
<td></td>
</tr>
<tr>
<td>—</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Parallelism</td>
</tr>
<tr>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Perpendicularity</td>
</tr>
<tr>
<td>—</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Angularity</td>
</tr>
<tr>
<td>—</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Runout</td>
</tr>
<tr>
<td>—</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Cylindricity</td>
</tr>
<tr>
<td>—</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Roundness</td>
</tr>
<tr>
<td>—</td>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Profile</td>
</tr>
<tr>
<td>—</td>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Concentricity</td>
</tr>
<tr>
<td>—</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Straightness</td>
</tr>
<tr>
<td>—</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Flatness</td>
</tr>
<tr>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>True Position</td>
</tr>
</tbody>
</table>

Figure 4.2 Feature Representation Chart.

As a general rule of thumb, all surface contacts will be modeled as a rotational matrix and all line contacts will be modeled as a translational matrix. The only feature which does not fall into the category of surface or line contact is the pin joint. The pin joint is a little more difficult to model and will be discussed in more detail later in the chapter.
4.3.1 Rotational Representation of ANSI-Y14 Feature

Joints which meet in surface contact are modeled as rotational matrices. Figure 4.3 shows two blocks in contact. The area of contact is displayed in exploded view showing the mating surfaces.

![Diagram of rotational variation and tolerance zone]

Figure 4.3 The rotational variation due to tolerance band.

The tolerance zone or tolerance band is represented by plus and minus tolerances on either side of the nominal mating surface. This tolerance zone is similar to the standard interpretation for ANSI-Y14 features. Since the surface may take any form as long as it stays within the tolerance zone, an approximation of the possible angular variation is taken by considering a diagonal line through the tolerance zone, as shown in Figure 4.3. The equivalent angular variation is computed as the arc-tangent of the width of the tolerance zone divided by the length of the surface in contact. And since the diagonal could be drawn from the opposite corners, it is a +/- variation.
The sensitivity of this ANSI-Y14 feature will be added to the vector lying parallel on the joint. This allows the angular variation due to the feature to carry through the vector loop.

4.3.2 Translational Representation of ANSI-Y14 Feature

The translational representation of an ANSI-Y14 feature occurs for all joints making line or point contact with its mating part. Figure 4.4 shows an example of such a joint and its corresponding translational variable.

![Figure 4.4 The translational variation due to tolerance.](image)

The circular part shown has a size tolerance and a circularity tolerance. Both result in a variation in the radius dimension. The size tolerance represents a variation in the average radius. The circularity tolerance is an independent size variation due to its roundness variation.

Another way of looking at it is to sketch the circularity tolerance zone as a concentric band as shown in Figure 4.4. The surface of the circle can take any form as long as it stays within the bounds of the tolerance zone. If the circle were elliptical, it might be assembled with the major axis of the ellipse as the diameter or with the minor axis as the
diameter. For a given circle size, rotating to a new orientation could produce an additional size variation in the assembly which is independent of the size tolerance.

Since there is no nominal dimension, the ANSI-Y14 translation matrix is modeled as a zero nominal vector with an associated tolerance. The tolerance sensitivity is in the direction of the surface normal and therefore acts in the same direction as the nominal radius vector at the surface. The sensitivity of the nominal radius vector is then computed and may also be applied to the circularity tolerance.

4.4 Matrix Sequence and Change Propagation of ANSI-Y14 Feature

A major advantage of representing the ANSI-Y14 features in matrix form is that the matrices allow for the effects of change propagation to continue through the vector loop. The figure below will be used to demonstrate the matrix sequence of ANSI-Y14 features and the Change Propagation which accompany this series.

Figure 4.5 Sample Cam assembly.
Figure 4.5 is a cam assembly to demonstrate the rotation of reference coordinates at a joint and the corresponding matrix operators. V1 is the vector radial distance from the edge. V2 is another radial vector from the center of the arc to the tangent point. V3 is the third vector from the tangent point to the end of the valve stem cap. The mating joint occurs between vectors 2 and 3. The mating joint at this point is a higher sliding joint. The sliding plane lies along the valve stem cap which differs by angle $\theta$ from the absolute reference plane.

The matrix representation for the roundness feature control may be inserted into the matrix sequence at the appropriate point and the analysis maybe carried along with no problems. However, for proper analysis, it is necessary first to rotate to the reference plane normal to the surface, apply a translation operation and then to rotate back to the global origin so that the rest of the analysis will not be thrown off.

$$\begin{bmatrix} RV2 & TV2 & RJ2 & TJ2 & -RJ2 & RV3 & TV3 \end{bmatrix} \begin{bmatrix} V2 & \text{ANSI-Y14 Feature} & V3 \end{bmatrix} \text{ Eq. 4.2}$$

Eq. 4.2 shows the matrix sequence of a portion of the Cam assembly. $R_{v2}$ is the initial coordinate rotation into the direction of V2 and $T_{v2}$ is the distance along the direction of V2. $R_{v2}$ is the relative rotation from the V2 direction to the V3 direction. $T_{v3}$ is the translation along the sliding plane. Therefore $R_{v2}$ and $T_{v2}$ represent the matrix model of V2; $R_{v3}$ and $T_{v3}$ represent V3.

At this point in the matrix sequence, we can insert the matrix equivalent of the feature control. In general it will be a three matrix operation as shown. $R_{j2}$ rotates the coordinate system normal to the surface and $T_{j2}$ translates in a direction normal to the surface. Then $-R_{j1}$ is the matrix which rotates the reference back to the original reference. This is done so that $R_{v3}$ will not be effected by the ANSI-Y14 coordinate change of $R_{j2}$. Then $T_{v3}$ is the dimension along Vector 3.
In the case of the Higher Slider shown in Figure 4.6, the sensitivity in the ANSI-Y14 equation is identical to the sensitivity of the nominal dimension. This is due to the colinear nature of the dimension vector and the zero dimension vector variation of the ANSI-Y14 feature.

In the case of the Higher sliding joint, the three ANSI-Y14 matrices are trivial. There is no rotation in the $R_{J2}$ and -$R_{J2}$ since the ANSI-Y14 feature will be acting in the direction of the radial vector which is normal to the reference plane. In a Higher sliding joint the radial vector will always be normal to the reference plane. The translation vector $T_{J2}$ will also be zero, since the ANSI-Y14 features are modeled as a zero dimension with an associated tolerance. When these factors are considered all three ANSI-Y14 matrices are identity matrices. However, they are still necessary so that the sensitivity may be computed from a small change in $T_{J2}$.

![Figure 4.6 Edge joint contact.](image)

If the joint in Figure 4.5 were an Edge joint like Figure 4.6, then the ANSI-Y14 matrices would not be trivial. The incoming vector on the face of the Edge joint could make contact with the surface at some angle other than $90^\circ$. Then the Vector V2 would make contact with the reference plane at a non-normal angle. This would mean that the tolerance variation due to ANSI-Y14 feature would not act along any nominal vector, but
would act normal to the contact surface. In this case the reference axis would have to be rotated \( \theta + 90^\circ \) \( ([R_{J2}]) \) normal to the reference plane and the ANSI-Y14 feature tolerance variation \( ([T_{J2}]) \) would then act normal to the reference plane. Once this has been accomplished the reference coordinate must be rotated back to the original position so that the next vector, \( V3 \), may continue unimpeded.

By treating the ANSI-Y14 joints as a sequence of three matrices, the joint matrices may be inserted in the proper VAM matrix sequence. Since the reference of the ANSI-Y14 matrix is rotated back, the joint may be modeled without disturbing the change propagation throughout the assembly.

4.5 Modeling Pin Joints

The modeling of a pin joint is a difficult problem which still requires more research. As a first approximation, the pin joint maybe modeled as a pin with a position tolerance attached to it. The tolerance band is applied equally to the incoming vector and the outgoing vector (see Figure 4.7). This method of modeling the pin joint gives a conservative estimate of the effects of position tolerance. Each vector associated with the pin joint has at least two tolerances: 1) the nominal dimension tolerance, 2) tolerance due to the pin joint.
If the pin joint is examined closer there are several other factors which must be resolved before the pin joint can be modeled accurately. One of the points is that there are true position tolerances of the hole, in both of the parts to be joined. Second, there could be a circularity error on the holes themselves. Third, the pin could have a roundness error. All of these factors must be taken into account in order to model the pin joint accurately. This would require more research using statistical methods to determine the binding tolerance for each component.
5.0 ME-10 Macro Implementation

The HP ME-10 CAD system was chosen as the experimental development system for the tolerance modeling interface. This system was available in the ME Dept. and featured a menu-driven user interface and a Macro language for user application development. Prototype application software was created to try out modeling concepts, interface designs, and data structures. The objectives of this effort were the investigation of the Macro capabilities and the identification of the CAD functions and database requirements for tolerance modeling.

5.1 ME-10 Macro Language

The Macro Language is the interface language of ME-10, Hewlett-Packard's 2-D CAD system. This section overviews the Macro language and its capabilities.

5.1.1 General Evaluation

The ME-10 macro language is the macro programming language used to extract necessary geometric data from the on-screen tolerance loop. The macro language is primarily designed to manipulate geometric entities and is not suited for extensive application development. It has poor character string manipulation and has no array or subroutine parameter capability. There are three primary types of commands.

The first type is "COMMAND." The commands which fall under this category are all those statements which directly control the geometric data within ME-10. These statements include: LINE, POINT, CHAMFER, and ADD_ELEM_INFO.

With the COMMAND syntax the developer is able to create macros to draw standard bolts and modify existing drawings, and create almost any geometry necessary.
The second type of command is "FUNCTION." Functions handle all of the non-geometric chores of ME-10. The FUNCTION statements include: OPEN_FILE, CLOSE_FILE, EDIT_MACRO, and EDIT_FILE.

With the FUNCTION syntax the developer is able to open a file, write to a file, and read from a file. This set of macro commands helps the developer manipulate the information extracted from the on-screen geometry.

The third and final type of command is "EXPRESSIONS." The expressions include all of the arithmetic symbols (+, -, *, /), logicals (If...then, >, <, =), and character manipulation. Character manipulation is an important aspect of data management since all text and numerics are handled as ASCII. The ME-10 Macro language has a very limited character array handling capability. For example, if the location of a blank (' ') is to be found, then the string must broken down to a substring and the string location searched character by character until the blank is found - a tedious process.

The ME-10 Macro language is a sequentially executed and interpretive programming language. This Macro language lacks two very important tools for the software developer: 1) an array capability, and 2) subroutine parameter passing capability. Each of these shortcomings has been overcome by alternate programming methods.

In order to make up for the lack of an array system, a counter is kept and incremented each time a new vector is chosen. The pseudo array is created by updating all values as the counter is incremented. This method is used to keep all of the variables, such as the plus and minus tolerance, tolerance type, etc. in local memory so that editing of vectors in the loop can be done locally in memory. This turns out to be very complex and inefficient due to the large amount of information to be handled: 24 information entities per vector, multiplied by vectors-per-loop, plus updates for vector insertion and deletion. This pseudo array method of keeping track of vector information added in excess of 5000 lines
of code to the 3000 existing lines of the actual modeling interface. This method was abandoned after 8000 lines of code was not adequate to handle all of the looping capabilities needed.

The lack of subroutine parameter passing capability was originally overcome by setting all variables as global. This allowed the variables to be read by any and all other macros. While setting all variables as global solved the subroutine problem, it became difficult to keep all of the variable names in sequence in order to update them.

Although these two methods of handling variables and arrays did work, it proved to be difficult to document and follow in future modifications. Therefore, rather than keeping all variables in local memory, all necessary information is written to the ME-10 geometric data base (MI format) and retrieved by a Fortran 77 program.

5.1.2 Tools Utilized

In the 1-D interface, all vectors are co-linear and thus, contact lines are used as references for vector heads and tails. In the 2-D interface, contact points are used as references. This could result in non-linear vectors and the location of the vector along a contact surface could change the sensitivity of that vector. The interface must therefore be able to determine if the point chosen on the assembly drawing is at a mating part joint or a continuation joint. Since it is impossible to error check all points selected from the assembly drawing, it is assumed that most points selected from the assembly drawing will be at an intersection and therefore the CATCH option has been set as a default on VERTEX. If the designer encounters an instance where another CATCH option is desired, the designer may simply select from any one of nine CATCH options prior to selecting the desired point. These CATCH options are:

1. ALL (anywhere on the screen)
2. VERTEX (end point of arcs, splines, or straight elements.)
3. ELEM (anywhere on the selected element)
4. INTERSECTION (intersecting point between two elem.)
5. CENTER (center of a circle, line, etc)
6. OFF (turns off the catch mode entirely)
7. GRID (select off grid lines)
8. RANGE (catches given point size radially from the cursor)
9. NO_VIEWPORT_RANGE (catch to point from keyboard)

Once the CATCH option has been selected, the desired point is read into a variable name. This is done by a READ function. Before ME-10 can read a variable, the designer must declare the type specifier. The following declarations identify variable types:

1. LITERAL (assigned to macro without evaluation)
2. NUMBER (real or integer)
3. STRING (character string)
4. PNT (x, y-coordinates of a point)
5. PNT3 (x, y, and z-coordinates of a point)

The LITERAL type specifier reads the variable literally without checking to see if the input matches the type specifier. If the token or input is a command, then the macro function is aborted and the command executed. If the token is a function, then the function is executed and READ request resumed. If the type specifier is a NUMBER, STRING, PNT, or PNT3, the token is checked and if the token does not fit the type specifier an error message is given and the READ request is resumed.

The input to satisfy the READ prompt may either be input from the keyboard or selected from the screen. For example, when HP-CATS 2.0 prompts the designer for a decision in ending a tolerance loop, the designer may either select the appropriate location
on the screen menu, or on the tablet, or type "CONFIRM" from the keyboard. This capability aids greatly in the designer interface because it eases the data entry.

Once the token is assigned to a variable name, it may then be re-assigned to a different variable name with the LET function. The LET function defines or redefines a macro name. The syntax of the LET function is: LET (macro name) (token or expression) e.g. LET Part_name Part1. Using the LET function a pseudo array is created with the help of a counter:

IF (COUNTER=1)
    LET LOOP_NAME_1 LOOP_NAME
ELSE_IF (COUNTER=2)
    LET LOOP_NAME_2 LOOP_NAME
ELSE_IF (COUNTER=3)
    LET LOOP_NAME_3 LOOP_NAME
END_IF

Figure 5.1 Sample of pseudo array.

5.2 Designer Interface

This section is a description of the designer interface and the reasoning behind the interface.

5.2.1 Tolerance Loop Creation and Verification

One of the first rules for creating a tolerance loop is that the designer must begin the loop at a point on one side of the clearance and end at a point on the other side of the clearance (open loop). For a closed loop the designer must start at one point and finish at the identical point. It is imperative that the designer create the tolerance loop on an
assembly drawing that has been drawn exactly to scale. If this is not done then there will be errors in reading the data that will ripple throughout the analysis.

![Diagram showing geometric error](image)

**Figure 5.2** Sample geometric error.

In the Figure 5.2 an error occurred while the designer constructed the contacting geometry. If the tolerance loop is performed on this model, the geometric error will ripple through the matrix equations. The designer must make sure that the geometric model of the assembly is accurate.

The ME-10 Macro language has been useful in extracting numerical data from the geometry through the use of its READ statements. The Macro language, like Fortran 77, allows formatted reads. If the designer must select a point from the screen, the format calls for a point to be READ and the CATCH option calls for a vertex. Thus, if the designer selects a number from his numeric keyboard, or a character string from the alphanumeric keys, or digitizes anywhere on the screen except for a vertex, ME-10 will not accept the input and will prompt the designer to try another input. This is the first step in error trapping for proper data input.
5.2.2 Practical Implementation

Loop Block

The ME-10's macro language was used to extract all of the numerical data from the CAD system. Since the MI (Mechanical Interface) file only saves the geometric entities and its associated information, all of the assembly, loop, and drawing information is added to a geometric entity. This information is added to the "loop block" using the ADD_ELEM_INFO command. See Figure 5.3 for a sample loop block. This command allows multiple character strings to be appended to a geometric entity as invisible data. When the designer selects the point for the loop block, that location is used as a reference. Then, by using the drawing scale, HP-CATS 2.0 finds the proper locations of the end points of the loop block. All assembly, loop, and drawing information is then attached to the top line of the loop block.

![Location of assembly, loop, and drawing info.](Loop Number 1)

Figure 5.3 Location of all information on loop block.

_descriptions_

HP-CATS 2.0 reads all of the assembly descriptions, loop descriptions, and drawing numbers by means of a logical argument. A counter increments each time a new line is input. When a blank line is read, the counter will stop and all lines of description will be added to the loop block. Due to the lack of an array system in ME-10's macro language each line possesses a unique variable name. The naming scheme of all descriptive lines consists of descriptive names followed by the sequence number. For example, if the designer has input three assembly description lines, the three variable names would be: assem_descript_1, assem_descript_2, and assem_descript_3.
Layering

Each vector loop is put on a separate drawing layer. By doing so, the designer is able to turn off any layer and view a specific part or vector loop any time during the execution of HP-CATS 2.0. HP-CATS 2.0 performs this layering by adding a value of 100 to the loop number designated by the designer. The ME-10 will accept up to 999 layers. The layering capability proves to be a useful feature especially when there is more than one vector loop and multiple part drawings overlayed in the assembly drawing.

Invisible Points

The ME-10 dimensioning function is used to add dimension lines and text to the vector loops on the drawings. However, ME-10 does not allow overlayed parts to be dimensioned unless all of the part drawings are crushed into one drawing. Crushing part drawings is a process by which all of the individual part drawings are fused together into one layer. Consequently, there is a problem in catching a vertex in an assembly drawing that has several layers. This problem is remedied by simply placing a point with the same color as the background at the beginning and ending point of the vector. This invisible point provides a geometric entity which the dimension line can catch. This gives HP-CATS 2.0 the ability to work with multiple layer drawings.

Macros

In order to modularize HP-CATS 2.0, each function of HP-CATS 2.0 is written as a separate macro. A macro is similar to that of a subroutine. ME-10 does not have true subroutine capabilities and due to this lack of subroutine capability, all variables used by more than one macro are set as global variables. This allows the contents of a variable to be accessed by several different macros. It is crucial for the program developer to remember how and at what point each variable is updated as HP-CATS 2.0 progresses through the modeling process.
5.2.3 Linking with CATS.BYU

The linking of HP-CATS 2.0 and CATS.BYU 2.0 is done through the neutral file. The neutral file format and characteristics are discussed later in greater detail. This entails the formulation of the neutral file by formatting the information in the "Mechanical Interface" file into the neutral file format. Presently, CATS.BYU 1.0 converts the neutral file output by HP-CATS 1.0 and formats it into a CATS.BYU database. The data base is a direct access file which has the capability to add and delete information directly. The database also has rigorous internal error checking to check for common loop names, parts, etc. The neutral file created by HP-CATS 2.0 will require a similar capability in CATS.BYU 2.0.

5.2.4 Programming Methods

To avoid future errors, it is worth noting the different programming methods which have been implemented. The three methods of programming HP-CATS 2.0 have been an attempt to overcome the lack of an array capability in ME-10's macro language. The first attempt was to allow the designer to have on-hand editing capability. This meant that the designer may add and delete vectors at any point in the interface, as well as add and delete ANSI-Y14 features at any time. Without the array capability a pseudo array is created with a counter to increment the variables. Each time a new vector is added HP-CATS 2.0 increments the counter, updates all of the variables, and adds the new vector. A naming scheme has been developed to keep track of all of the vectors and variables. For example, the name of vector 1 of loop 1 is loop_1_vec_1, and vector 2 of loop 1 was loop_1_vec_2, etc. There are approximately 35 variables associated with each vector which must be updated. This method created an enormous amount of code and became unwieldy. It required approximately 500 lines of code to simulate one Fortran 77 "DO LOOP." Just the HP-CATS 2.0 editing capability required 7000 lines of code. Variables are scattered
everywhere and the maintenance of HP-CATS 2.0 after the completion of this thesis would be a major fiasco. The strong point in this method was that the designer did have direct editing capability of any vector or feature at any time. The designer could, at any time, insert vectors, delete vectors, insert ANSI-Y14 features, and delete ANSI-14 features. This method also eliminated the need for data manipulation to be performed exterior to the macro interface.

The second method was to try and store all of the pertinent information into character arrays and extract them as needed. This was simpler in that it reduced the number of lines of code and the number of variables. The problem occurred in the 128 character length which ME-10 would allow. Multiple character strings were required for multiple vectors. Extracting and appending information to a character string required a parser. A parser was used to parse the information from the character strings to perform direct editing of the vectors. Each variable was renamed so that editing could be performed. This proved a little slower than the first method. Due to the poor character handling capability of ME-10's macro language, the code to handle the character array soon grew too large to be managed effectively. Since one of the goals of HP-CATS 2.0 is modularity to simplify maintenance, this second method has been abandoned.

Finally, direct vector editing capability has been eliminated and all vector information is added directly to the geometric vector entity. This requires the capability to go into the MI file format and extract the necessary information directly. The extraction of information from the MI file is done by a Fortran 77 program. The Fortran 77 data extraction program provides the array capability HP-CATS 2.0 lacked. All of the global editing is done once the designer has finished creating the vector loop. Local editing is still available within HP-CATS 2.0 to change entities of a vector, i.e. tolerance type, tolerances, dimension, etc. This method of editing works well and has reduced the macro interface to 3500 lines from the original 10,000 lines. The Fortran portion of the interface adds another
3000 lines of code. The advantage of Fortran 77 in the interface is not only its array capability but almost any mechanical engineering student can read the program top down, and by following the comments, understand the internal structure.

5.3 Data Structures

There are two basic file systems used in HP-CATS 2.0. The two file types are the MI file format and the CATS.BYU neutral file format.

5.3.1 MI File Format

The MI format is one of two geometric data base forms which Hewlett-Packard uses to store all ME-10 drawings. One format is the binary format. The drawing is reduced to a set of binary code which is stored in a file. This file can be viewed but the binary format and direct access structure of the file makes editing the file nearly impossible. The second method of storing geometric data is the MI or Mechanical Interface file format. This format is very similar to IGES, DXF, or other geometric neutral file formats. There is only one entity per line with no preceding or trailing blanks. The MI format carries 13 significant digits for real numbers. MI also has a short real format which carries 4 significant digits. These two real number formats would be equivalent to double precision and real number formats in Fortran 77.

Four main blocks of the MI file have been used to generate the information required by HP-CATS 2.0. Each of the four blocks in the MI files and its significance in HP-CATS 2.0 will now be explained.

The first block of interest is ASSP, which stands for Associated Text Property. ASSP is the block which marks geometric a entity with its characteristic information. ASSP may be used to separate entities of the same type from each other and can also be used as an information grouping mechanism. By adding characteristic information to the
vector geometry, all necessary information is stored within the MI file. The ASSP format is as follows:

**Identification data:**

- **ASSP**
- **Entity Sequence Number**

**Parameter Data**

- **string count**
- **n+ (number of string in associated text)**

- **string 1**
- **STRING (Associated text string 1)**

- **...**

- **string n**
- **STRING (Associated text string n)**

The following examples are actual data taken from an MI file. Seven different examples of how ASSP has been used to store necessary information are explained in the following paragraphs.

**Example 1:**  Associated Text Block

```
ASSP
9
1
LAYER: 101
|--
```

:ASSP Marker.
:Event number.
:Number of info. marker.

This info. is a LAYER, and the layer is 101.
The pipe and tilda is the end of info marker.

Example 1 stores the event number containing the loop number. Each loop is stored in a separate layer, beginning with number 101. 100 is added to the loop number that is input by the designer to make a unique LAYER within the MI file. Therefore "Layer 101" refers to Loop 1. This means that any pointer in the LINE block which points back to Event 9 has stored information belonging to Loop Number 1.

**Example 2:** ASSP assembly information block.

```
ASSP
10
1
&*A*stacked block problem*
|--
```

:ASSP Marker.
:Event number.
:Number of info. marker.

:&*A - Assembly description marker
:* - Marks beginning & end of description.
:The pipe and tilda is the end of info marker.
This block holds the assembly description information. The marker "&A" is a pointer to the neutral file format package and indicates that the line to be read is an assembly description. There is no pointer to the loop number in this descriptive line. This is because the assembly describes the entire file and all the loops within the assembly. The character string residing between the two asterisks (*) is the description line. This line can be 124 characters maximum. Since the maximum character length that the MI file will handle is 128 characters, two are taken by the asterisks and two by the "&A" marker.

Example 3: ASSP loop information block.

```
ASSP  :ASSP Marker.
12    :Event number.
1     :Number of info. marker.
&L1*first loop about the top left of assembly*
      :&L - Loop description marker
      :* - Marks beginning & end of description.
|~     :The pipe and tilde is the end of info marker.
```

This information block is identical to that of the previous assembly description except that the "&L" marker signifies that this block is a loop description. Only 122 characters are allowed in the loop description line. This is because two extra spaces have been allocated for the loop number (I am allowing only 99 vector loops to an assembly) which follows immediately after "&L" marker.

Example 4: ASSP drawing number block.

```
ASSP  :ASSP Marker.
13    :Event number.
1     :Number of info. marker.
&D*123-1* :&D - Drawing numbers marker.
          :* - Marks beginning & end of drawing number.
|~     :The pipe and tilde is the end of info marker.
```

This block stores the ID numbers of the part drawings that were overlaid to create the assembly drawing. Like the assembly description there is no reference to a loop number. Since multiple loops may be created on an assembly drawing, it is assumed that all necessary part drawings have been compiled.
Example 5: ASSP loop header block.

ASSP : ASSP Marker.
16 : Event number.
1 : Number of info. marker.
>1*DEMO*LOOP.ONE*FIRST*
   : - End loop description marker.
   : * - Marks beginning & end of 1) Assembly file name
      2) Assembly name
      3) Loop name
<- : The pipe and tilda is the end of info marker.

This block stores the loop number, assembly file name, assembly name, and the loop name. The marker "->" is used as a pointer to the neutral file formatting application and indicates that this line is the header of loop number one.

Example 6: ASSP vector information block.

ASSP : ASSP Marker.
17 : Event number.
1 : Number of info. marker.
^L1:4*+0.005*-0.005*0*Edge*2*3*0*-9.0741*0*14.2493*0*-0.9588*
   : vector info. marker.
   L1:4 : vector name
   + : tolerance feature (+/- or upper and lower)
   0.005 : plus tolerance
   -0.005 : negative tolerance
   0 : tolerance type (0 = not fixed, 1 = fixed)
   Edge : joint type
   2 : first part to cross the joint boundary
   3 : second part to cross the joint boundary
   -9.0741 : x-coordinate of the first reference point
   0 : x-coordinate of the second reference point
   14.2493 : y-coordinate of the second reference point
   0 : x-coordinate of the tail of the vector
   -0.9588 : y-coordinate of the tail of the vector
<- : The pipe and tilda is the end of info marker.

The block with "^" marking denotes vector information.

Example 7: ASSP end loop information block.

ASSP : ASSP Marker.
40 : Event number.
1 : Number of info. marker.
<3*0.8522*-5.7919*0.005*0.0005*
   : footer info. marker.
   < : loop number
   3 : loop number
   0.8522 : x-coordinate of the last point in the loop.
   -5.7919 : y-coordinate of the last point in the loop.
   0.005 : desired clearance.
   0.0005 : desired clearance tolerance.
<- : The pipe and tilda is the end of info marker.
This final ASSP block stores the footer information. With the x and y-coordinate of the last point in the loop it is possible to find out if the loop is a closed loop or an open loop. This done by measuring the distance between the tail of the first vector and the final point of the loop. If the measured distance between them is smaller than .0001, then the loop is considered a closed loop. The number .0001 was used as a reference point since the numbers written to the ASSP block have only four significant digits after the decimal.

The next information block used is the point block. This block has a P header as the block identifier. The P block is important in the formatting process because the points mark the beginning and the ending of each vector. By checking different event numbers with the x and y-coordinates of those points, equivalent vectors can be identified. The general format is:

Example 8: Point entity information.

<table>
<thead>
<tr>
<th>Identification data:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity Type</td>
<td>P</td>
</tr>
<tr>
<td>Entity Sequence Number</td>
<td>n+ (integer)</td>
</tr>
<tr>
<td>Parameter Data</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>real</td>
</tr>
<tr>
<td>y</td>
<td>real</td>
</tr>
<tr>
<td>z</td>
<td>real (only existent if GLOBAL SPACE DIMENSION equals '3D')</td>
</tr>
</tbody>
</table>

P :P Marker.
57 :Event number.
2.43968254968205L1 :X coordinate of the point
1.42445124731295L1 :Y coordinate of the point
!- :The pipe and tilda is the end of info marker.

The third and most important information block is the line block. This block header is identified by a LIN marker. This block stores the necessary information to identify
vectors, the loop numbers with which those vectors are associated, and the end points of the vectors. The general format is:

Identification data:

```
Entity Type
Entity Sequence Number n+ (integer)
```

Attribute Data

```
Color n+ (integer)
Line Type n+ (integer)
Line Width Weight n+ (integer)
```

Property Data

```
Property Count n+ (integer)
Property Pointer 1 PTR
...
Property Pointer n PTR
```

Parameter Data

```
PTR to Pstart PTR Pointer to Start Point
PTR to Pend PTR Pointer to End Point
```

Example 9: Line entity information.

```
LIN
219 :LIN Marker.
3 :Event number.
0 :Color
1 :Line Type
2 :Line Width Weight
9 :Property Count
17 :PTR to Property 1
128 :PTR to Property 2
90 :PTR Pointer to Start Point
92 :PTR Pointer to End Point
```

After the LIN header and the attribute data, the Property Count tells how many Property Pointers will be listed. All vectors have only two Property Pointers (PTR). Each points to the "LAYER," which corresponds to the loop number. The next two pointers
indicate the starting and ending point coordinate data. This data is needed so the beginning and ending point of the line maybe checked with all the other vectors to see which vectors are shared from the other vector loops. If a vector is not being shared, an error message is sent out to the designer notifying him of the discrepancy.

The fourth and final information block that is read is the **Single Dimension** or **SD** block. This is the ME-10 function used to mark all vectors with dimension lines. When a vector is added or deleted, many of the Single Dimension markers must be updated. In order to do this, all the Single Dimension information is read into a Fortran array. Then the array can be updated as the designer updates the drawing.

A vector loop may be modified on the drawing by: 1) HP-CATS 2.0 reads the vector and loop number of the vector to be added or deleted, 2) HP-CATS 2.0 reads the head of the new vector, 3) all of this information is written to an invisible file, 4) a Fortran 77 program reads the invisible file and deletes the old vector from the MI file, 5) HP-CATS 2.0 redraws the new geometry to the screen and inserts the new vector, 6) HP-CATS 2.0 saves the updated geometry back into MI format. The general format for an SD block is:

**Identification Data:**

<table>
<thead>
<tr>
<th>Entity Type</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity Sequence Number</td>
<td>n+</td>
</tr>
</tbody>
</table>

**Property Data:**

<table>
<thead>
<tr>
<th>Property Count</th>
<th>n+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to Property 1</td>
<td>PTR</td>
</tr>
<tr>
<td>. . .</td>
<td></td>
</tr>
<tr>
<td>Pointer to Property n</td>
<td>PTR</td>
</tr>
</tbody>
</table>

**Parameter Data:**

<p>| Location Point | Real,Real |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculated/Edited flag</td>
<td>0,1</td>
</tr>
<tr>
<td>Balloon Pointer</td>
<td>PTR</td>
</tr>
<tr>
<td>First Pointer to Geometry Point</td>
<td>PTR</td>
</tr>
<tr>
<td>Second Pointer to Geometry Point</td>
<td>PTR</td>
</tr>
<tr>
<td>With/without extension lines</td>
<td>0,1</td>
</tr>
<tr>
<td>Arrow Line Direction</td>
<td>0,...,2</td>
</tr>
<tr>
<td>Extension Line Inclination</td>
<td>30,...,90 (default is 90)</td>
</tr>
<tr>
<td>Connect Flag</td>
<td>0,1</td>
</tr>
<tr>
<td>Character Height</td>
<td>Real &gt; 0</td>
</tr>
<tr>
<td>Character Width</td>
<td>Real &gt; 0</td>
</tr>
<tr>
<td>Color</td>
<td>n+</td>
</tr>
<tr>
<td>Text Color</td>
<td>n+</td>
</tr>
<tr>
<td>Text Lower Left</td>
<td>Real, Real</td>
</tr>
<tr>
<td>Text Upper Right</td>
<td>Real, Real</td>
</tr>
<tr>
<td>Angle</td>
<td>Real</td>
</tr>
<tr>
<td>Delta Angle</td>
<td>Real</td>
</tr>
<tr>
<td>Text Origin</td>
<td>1,...,9</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ARROWS</td>
<td></td>
</tr>
<tr>
<td>Count</td>
<td>n+</td>
</tr>
<tr>
<td>Arrow Type (#1)</td>
<td>0,...,4</td>
</tr>
<tr>
<td>Arrow Size</td>
<td>Short Real</td>
</tr>
</tbody>
</table>

![Diagram of arrow types](image-url)
<table>
<thead>
<tr>
<th>Arrow Location</th>
<th>Short Real, Short Real</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrow Type (#n)</td>
<td>0, ..., 4</td>
</tr>
<tr>
<td>Arrow Size</td>
<td>Short Real</td>
</tr>
<tr>
<td>Arrow Location</td>
<td>Short Real, Short Real</td>
</tr>
<tr>
<td><strong>LINES</strong></td>
<td></td>
</tr>
<tr>
<td>Count</td>
<td>n+ (number of lines)</td>
</tr>
<tr>
<td>Begin Point (#1)</td>
<td>SHORTREAL, SHORTREAL</td>
</tr>
<tr>
<td>End Point (#1)</td>
<td>SHORTREAL, SHORTREAL</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Begin Point (#n)</td>
<td>SHORTREAL, SHORTREAL</td>
</tr>
<tr>
<td>End Point (#n)</td>
<td>SHORTREAL, SHORTREAL</td>
</tr>
<tr>
<td><strong>PREFIX</strong></td>
<td></td>
</tr>
<tr>
<td>Omit/Draw Prefix</td>
<td>0, 1</td>
</tr>
<tr>
<td>Reference Point</td>
<td>SHORTREAL, SHORTREAL</td>
</tr>
<tr>
<td>Prefix Scale</td>
<td>REAL</td>
</tr>
<tr>
<td>Prefix Text</td>
<td>STRING</td>
</tr>
<tr>
<td><strong>POSTFIX</strong></td>
<td></td>
</tr>
<tr>
<td>Omit/Draw Postfix</td>
<td>0, 1</td>
</tr>
<tr>
<td>Reference Point</td>
<td>SHORTREAL, SHORTREAL</td>
</tr>
<tr>
<td>Postfix Scale</td>
<td>REAL</td>
</tr>
<tr>
<td>Postfix Text</td>
<td>STRING</td>
</tr>
<tr>
<td>Dual Dimensioning Flag</td>
<td>0, 1</td>
</tr>
<tr>
<td><strong>Tolerance Type</strong></td>
<td>1, ..., 4</td>
</tr>
<tr>
<td>1</td>
<td>none</td>
</tr>
<tr>
<td>Reference Point</td>
<td>SHORTREAL, SHORTREAL</td>
</tr>
<tr>
<td>Dimensioning Text</td>
<td>STRING</td>
</tr>
<tr>
<td>2</td>
<td>Limit</td>
</tr>
<tr>
<td>Upper Limit</td>
<td>Real</td>
</tr>
<tr>
<td>Lower Limit</td>
<td>Real</td>
</tr>
<tr>
<td>Reference Point</td>
<td>SHORTREAL, SHORTREAL</td>
</tr>
<tr>
<td>Upper Text</td>
<td>STRING</td>
</tr>
<tr>
<td>Reference Point</td>
<td>SHORTREAL, SHORTREAL</td>
</tr>
<tr>
<td>Lower Text</td>
<td>STRING</td>
</tr>
</tbody>
</table>
Example 10: Single Dimension Block.

1) SD
2) 293
3) 2
4) 6
5) 5
6) 2.07533488449945L1
7) -8.06385902668717L-1
8) 0
9) 0
10) 92
11) 91
12) 0
13) 2
14) 1.5707963267949
15) 0
16) 6L-1
17) 6L-1
18) 2
19) 2
20) 1.24410491922223L1
21) -1.2951160567755
22) 2.08635491922223L1
23) -9.29511605677546L-1
24) 0
25) 0
26) 5
27) 2
28) 1
29) 2L-1
30) 2.18281L1
31) -6.1687L-1
32) 1
33) 2L-1
34) 5.99731L-1
35) -4.36001245724307
36) 4
37) 2.20111L1
38) -1.65483308086671
39) 2.17586L1

:Single Dimension Entity
:Event Number
:Property Count
:Pointer to Property 1
:Pointer to Property 2
:X coordinate of the location point
:Y coordinate of the location point
:Calc/Edit Flag
:Balloon Pointer
:First Pointer to Geometric Point
:Second Pointer to Geometric Point
:with/without Extension Line
:Arrow line direction
:Extension Line Inclination
:Connect Flag
:Character Height
:Character Width
:Color
:Text Color
:X coordinate of Text Lower Left
:Y coordinate of Text Lower Left
:X coordinate of Text Upper Right
:Y coordinate of Text Upper Right
:Angle
:Delta Angle
:Text Origin
:Arrow Count
:First Arrow Type
:First Arrow Size
:First X coordinate of Arrow Location
:First Y coordinate of Arrow Location
:Second Arrow Type
:Second Arrow Size
:Second X coordinate of Arrow Location
:Second Y coordinate of Arrow Location
:Line Count
:X coordinate of Line 1, Pstart
:Y coordinate of Line 1, Pstart
:X coordinate of Line 1, Pend
Due to the number of entities, the Single Dimension block is the most complex information block to read. Several key variables must be read in order for the formatting program to update the MI file once a vector and dimension have been inserted or deleted from the drawing. Two main variables will affect the updating of the MI file.

The first variable is on line 3), the Property Count, which is equal to two. The two property pointers which follow refer to the Event Numbers of other dimensioning information.

The second variable is on line 60), the Tolerance Type. Depending on the Tolerance Type, the number of reference points and tolerance text will vary. The MI file updating program has the capability of handling all five tolerance types.
5.3.2 CATS.BYU Neutral File

The CATS.BYU neutral file contains information necessary to identify all vectors which are used to model the assembly. The basic format of the neutral file is sequentially accessed and contains 80 characters per line. Each line starts with a three character mnemonic pointer which defines the type of data. The vector names act as pointers to the previous and next vector. This naming scheme makes reading the neutral file simpler to follow. The following figure is a sample CATS.BYU neutral file with detailed internal descriptions.

FILE>> NEU.DAT

ASSEMBLY: GEOMSTACK ()

DESCRIPTION: Sample neutral data file for transmitting tolerance stack data from a CAD system to CATS.BYU. The file is sequential, formatted.

1. The first line is a file header containing file name and creation info.
2. Each block begins with a header line (===) containing the block type (LOOP:) and 12 character code name required by CATS.
3. Every block can have its own description. Any number of continuation lines may be added, each preceded by "$ ".
4. Each record type is preceded by a unique 3 character mnemonic label.
5. Each data item has a bracket behind it ( ) for a "change flag".
6. Any data item which has been changed by CATS is flagged by a (j).
7. Data which may not be changed (vendor-supplied or design requirement) may be flagged by a (*).

6. LABELE RECORD TYPE
==== Data Block Separator.
$ ASS Assembly Name
$ DRA Drawing No.
$ MOD List the related assembly loops defining a single model.
$ DES Description
$ TEX Non-Data Record (ignored)
$ $ Continuation

ASSEMB NAME: STACKED GEOMETRIC SHAPES
DRAWING NO.: 0027-943
MODEL: GLOOP1 () GLOOP2 () GLOOP3 ()

DESCR: Each loop is stored as a table in a block of 80 character records.
$ Each record type is preceded by a unique 3 character mnemonic label.
1. LABELE RECORD TYPE
$ DES Description INI Starting coordinates for loop
$ JOI Assembly Pair(Joint) ORI Joint Orientation Data
$ DIM Component Dimension TAB Stack Table Data
$ FEATURE CONTROLS
$ FLA Flatness ROU Roundness
$ STR Straightness  CYL Cylindricity
$ PAR Parallelism  CON Concentricity
$ PER Perpendicularity  PRO Profile
$ ANG Angularity  RUN Runout
$ TRU True Position

$ Dimension record format specifications:
$23456789/123456789/123456789/123456789/123456789/123456789/123456789/123456789/
$TEX X0 Y0 Z0 XANG0 YANG0 ZANG0
$INI XXX.XXXX(A) XXX.XXXX(A) XXX.XXXX(A) XXX.XXX(A) XXX.XXX(A) XXX.XXX(A)
$TEX PART/DIM NAME AVE.DIM +/-TOL XANGLE YANGLE ZANGLE ORDER
$DIM AAAAAAAAAAA(A) XXX.XXXX(A) XXX.XXX(A) XXX.XXX(A) XXX.XXX(A) XXX.XXX(A)
$TEX JOINT NAME TYPE PART1 DEP PART2 DEP
$JOI AAAA AAAAA(A) AAAAA(A) AAAAA AAA(A) AAA(A)
$TEX JOINT NAME  XANGLE YANGLE ZANGLE
$ORI AAAAAAAAAA(A) XXX.XXX(A) XXX.XXX(A) XXX.XXX(A)

$ Feature control record format specifications:
$TEX PART/DIM NAME AVE.DIM +/-TOL XANGLE TOL YANGLE TOL ZANGLE TOL
$FLA AAAAAAAAAAA(A) XXX.XXXX(A) XXX.XXX(A) XXX.XXX(A) XXX.XXX(A) XXX.XXX(A)
$TEX -------------------------------
$TABLE: 1 TYPE: HOR X: -1.347656 Y: 0.131836 TEXTSIZE: 0.164
$SPECS: DIM: P1/10 AVE.DIM: 19.058 +/-TOL: 0.322
$TEX -------------------------------
$23456789/123456789/123456789/123456789/123456789/123456789/123456789/123456789/
$TEX X0 Y0 Z0 XANG0 YANG0 ZANG0
$INI XXX.XXXX(A) XXX.XXXX(A) XXX.XXXX(A) XXX.XXX(A) XXX.XXX(A) XXX.XXX(A)
$INI 90.0 ( ) 19.05( ) 0.0 ( ) 0.0 ( ) 0.0 ( ) 90.0 ( )
$TEX PART/DIM NAME AVE.DIM +/-TOL XANGLE/TOL YANGLE/TOL ZANGLE/TOL
$DIM P2/01 ( ) 1.750( ) 0.0015( ) -90.0
$TEX JOINT NAME TYPE PART1 DEP PART2 DEP
$JOI AAAAAAAAAAA(A) AAAAA(A) AAAAAAAAAAA(A) AAA(A)
$JOI JOINT1 ( ) CENTER( ) P2/01 ( ) ( ) P2/02 ( ) A( )
$DIM P2/02 ( ) 6.620( ) 0.2( ) -74.7239
$TEX JOINT NAME TYPE PART1 DEP PART2 DEP
$JOI AAAAAAAAAAA(A) AAAAA(A) AAAAAAAAAAA(A) AAA(A)
$JOI JOINT2 ( ) SLIDER( ) P2/02 ( ) ( ) P3/11 ( ) D( )
$ROU P2/02 ( ) 0.0 ( ) 0.05( )
$PAR P3/11 ( ) 0.0 ( ) 0.006( )
$ANG P3/11 ( ) 0.0 ( ) 0.006( )
$DIM P3/11 ( ) 8.6705(*) ? ( ) 270.
$TEX JOINT NAME TYPE PART1 DEP PART2 DEP
JOIAAAAAAAAAAAA(A) AAAAA(A) AAAAAAAAAAAA(A) AAA(A)
JOI JOINT3 ( ) EDGE ( ) P3/11 ( ) ( ) P1/14 ( ) DA ( )

ANG P1/14 ( ) 0.0 ( ) 0.004 ( )

TEX PART/DIM NAME AVE.DIM +/-TOL XANGLE/TOL YANGLE/TOL ZANGLE/TOL

DIM P1/14 ( ) 10.0477( ) ? ( ) 74.7239

TEX JOINT NAME TYPE PART1 DEP PART2 DEP
JOI AAAAAAAAAAAAA(A) AAAAA(A) AAAAAAAAAAAA(A) AAA(A)
JOI JOINT4 ( ) RIGID ( ) P1/14 ( ) D ( ) P1/10 ( ) D ( )

PER P1/14 ( ) 0.0 ( ) ( ) 0.05

DIM P1/10 ( ) 18.7182( ) 0.0005(*) 180.

TEX JOINT NAME TYPE PART1 DEP PART2 DEP
JOI AAAAAAAAAAAAA(A) AAAAA(A) AAAAAAAAAAAA(A) AAA(A)
JOI JOINT5 ( ) SLIDER( ) P1/10 ( ) D ( ) P2/01 ( ) ( )

PER P1/10 ( ) 0.0 ( ) ( ) 0.07

ROU P2/01 ( ) 0.0 ( ) 0.05(*)

LOOP: GLOOP2 ( )

DESCR: Loop specs may be defined in terms of unequal tolerances as shown.
$ Keywords in the record tell CATS which format is being used.
$

---

TEX -----------------------------------------------

TABLE: 2 TYPE: HOR X: 0.550781 Y: 5.077149 TEXTSIZE: 0.164
SPECS:  

TEX -----------------------------------------------

$23456789/123456789/123456789/123456789/123456789/123456789/
TEX X0 Y0 Z0 XANG0 YANG0 ZANG0
INI XXX.XXXXX(A) XXX.XXXXXX(A) XXX.XXXXXX(A) XXX.XXXXXX(A) XXX.XXXXXX(A) XXX.XXXXXX(A)
INI 0.0 ( ) 10.0477( ) 0.0 (A) 0.0 (A) 0.0 (A) 90.0 (A)

TEX DIM NAME AVE.DIM +/-TOL XANGLE/TOL YANGLE/TOL ZANGLE/TOL

DIM P3/12 ( ) 6.805( ) 0.075( ) -164.7239

TEX JOINT NAME TYPE PART1 DEP PART2 DEP
JOI AAAAAAAAAAAAA(A) AAAAA(A) AAAAAAAAAAAA(A) AAA(A)
JOI JOINT6 ( ) RIGID ( ) P3/12 ( ) ( ) P3/13 ( ) ( )

PAR P3/13 ( ) 0.0 ( ) ( ) 0.02( )

PER P3/13 ( ) 0.0 ( ) ( ) 0.02( )

DIM P3/13 ( ) 2.1894( ) ? ( ) 90.

TEX JOINT NAME TYPE PART1 DEP PART2 DEP
JOI AAAAAAAAAAAAA(A) AAAAA(A) AAAAAAAAAAAA(A) AAA(A)
JOI JOINT7  ( ) EDGE ( ) P3/13  ( ) D ( ) P1/08  ( ) A ( )
DIM P1/08  ( ) 4.060(*)  0.15 ( ) -105.2761

TEX JOINT NAME  TYPE  PART1  DEP  PART2  DEP
JOI  AAAAAAAAAAAAA(A)  AAAAAAA(A)  AAAAAAAAAAAAA(A)  AAA(A)
JOI JOINT8  ( ) RIGID ( ) P1/08  ( ) ( ) P1/09  ( ) ( )
PER P1/09  ( ) 0.0 ( ) ( ) 0.01( )
DIM P1/09  ( ) 3.805( ) 0.1250( ) 270.

TEX JOINT NAME  TYPE  PART1  DEP  PART2  DEP
JOI  AAAAAAAAAAAAA(A)  AAAAAAA(A)  AAAAAAAAAAAAA(A)  AAA(A)
JOI JOINT9  ( ) RIGID ( ) P1/09  ( ) ( ) P1/14  ( ) ( )
DIM P1/14  ( ) 10.0477( ) ? ( ) 270.

TEX JOINT NAME  TYPE  PART1  DEP  PART2  DEP
JOI  AAAAAAAAAAAAA(A)  AAAAAAA(A)  AAAAAAAAAAAAA(A)  AAA(A)
JOI JOINT10  ( ) EDGE ( ) P1/14  ( ) D ( ) P3/12  ( ) A ( )
PER P1/09  ( ) 0.0 ( ) ( ) 0.01( )

------------------------ LOOP: GLOOP3 ------------------------

DESCR: Loop specs may also be defined in terms of max/min dimensions as shown.
$Keywords in the record tell CATS which format is being used.
T EX

____________________________________________________________________________
TABLE: 2 TYPE: HOR X: 0.550781 Y: 5.077149 TEXTSIZE: 0.164
SPECS:
T EX

$23456789/123456789/123456789/123456789/123456789/123456789/123456789/123456789/
TEX X0  Y0  Z0  XANG0  YANG0  ZANG0
INI  XXX.XXXXX(A)  XXX.XXXXXX(A)  XXX.XXXXXX(A)  XXX.XXXXXX(A)  XXX.XXXXXX(A)
INI  27.655 ( )  10.675 ( )  0.0 ( )  0.0 ( )  0.0 ( )  15.2761( )

TEX DIM NAME  AVE.DIM  +/-TOL  XANGLE/TOL  YANGLE/TOL  ZANGLE/TOL
DIM P1/05  ( ) 10.675( )  0.125( ) -105.2761

TEX JOINT NAME  TYPE  PART1  DEP  PART2  DEP
JOI  AAAAAAAAAAAAA(A)  AAAAAAA(A)  AAAAAAAAAAAAA(A)  AAA(A)
JOI JOINT11  ( ) RIGID ( ) P1/05  ( ) ( ) P1/06  ( ) ( )
PER P1/09  ( ) 0.0 ( ) ( ) 0.04( )
DIM P1/06  ( ) 4.060( )  0.15( )  180.

TEX JOINT NAME  TYPE  PART1  DEP  PART2  DEP
JOI  AAAAAAAAAAAAA(A)  AAAAAAA(A)  AAAAAAAAAAAAA(A)  AAA(A)
JOI JOINT12  ( ) RIGID ( ) P1/06  ( ) ( ) P1/07  ( ) ( )
DIM P1/07  ( ) 24.22(*)  0.35 ( ) 270.
5.4 Requirements for Future Implementation

ME-10 is a very robust and flexible 2-D CAD system. However, its macro language is lacking in several major points. First, due to the lack of an array capability, reading variables by a loop is nearly impossible. The developer may wish to create a pseudo variable array or character array, but the pseudo arrays require many lines of code and do not portray a refined programming style. Second, it would be helpful to the developer if there were a specific macro system to work with the MI file. Since all geometric information is stored in the MI file, it would be most helpful if a procedure could be implemented to extract necessary data from the MI file. If such a procedural interface could be implemented, it would provide a great tool to interface the ME-10 CAD system to many other analytical packages.

The third useful feature would be the ability within ME-10 to read and execute another programming language. There are other CAD systems which execute LISP programs to manipulate geometry in real time. This would be a great tool to the developer.
to perturb vectors geometrically. Most engineers are visually oriented and if a vector loop could be placed on the screen, and each vector perturbed with immediate visual results the designer may get a better feel for tolerance loop relationships. The possibilities could be great if the commands to manipulate geometry were in the form of a programming language.

Another benefit could be automatic geometric manipulation after nominal dimensions have been altered in the assembly drawing. If the interface language could access local memory before storing the drawing in the MI format, then the option of updating the drawing on the screen would be available to the designer. There are arguments both for and against automatic geometric update. But just to have the option available for those who would want to see the update could be a powerful visualizing tool.
6.0 Case Studies

The following examples of 2-D tolerance analysis problems illustrate the application of the modeling system described in this thesis.

6.1 Over-running Clutch

The over-running clutch shown in Figure 6.1.1 illustrates the use of a closed loop to find the angular tolerance of a critical assembly variable. This assembly is designed to allow rotation in only one direction. The hub (Part 2) is attached to a shaft, which in turn may be attached to some driving mechanism. As long as the hub rotates CW relative to the cage (Part 4), the roller (Part 3) simply drags on the inside of the cage. If the hub rotates in the counter-clockwise direction, then the spring causes the roller to wedge between the hub and the cage, locking the two so they rotate together. [7]

The object of this analysis is to determine the location of the rollers with respect to the cage and the hub. The exact location of the roller must be held between strict limits to ensure proper performance. The location is measured by the angle $\alpha$, which is a dependent variable. The length of Vector $c$ is also a dependent variable. Since there are only two dependent variables, only one vector loop is necessary to model the assembly.

*Drawing Preparation*

All part drawings are overlaid to create an assembly drawing. Care must be taken to assure proper part orientation within the assembly to prevent geometric error as discussed in section 5.2.1.
Prior to creating the assembly vector loop, the designer should add any construction lines or points that will be needed as vector end points. Note that Vector \( \mathbf{c} \) will end at a non-vertex or non-intersection point. Therefore, the designer should create a construction line horizontally through Joints 3 and 4 and vertically through Joints 4 and 5. The intersection between these two construction lines is the point at which the head of Vector \( \mathbf{c} \) will rest. Catching a point intersection of construction lines is done using the "intersection" option. Creation of construction lines is not a ME-10 function, but a command, and is not allowed during the execution of HP-CATS 2.0.

**Modeling Procedure**

Due to symmetry, the two diagonally opposed quadrants include the diametrical length of the cage and its associated tolerance and provides sufficient information to model the entire assembly. First, all joints in the two quadrants should be identified by the
designer by visual inspection of the part interfaces. By generally defining the joints prior to
the creation of the VAM, the designer can get a better idea of the path of the vector loop
through the assembly. Once the designer feels comfortable about the joint definitions, the
VAM is created. It is important to note that all joints are assigned to the tail of the outgoing
vector from that joint. A table of joints for the clutch assembly is shown in Table 6.1.1.

Table 6.1.1 List of joints occurring in the over-running clutch.

<table>
<thead>
<tr>
<th>Joint</th>
<th>Type</th>
<th>Located at:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Roll w/o Sliding</td>
<td>Intersection, Parts 1 &amp; 4</td>
</tr>
<tr>
<td>2</td>
<td>Center</td>
<td>Center, Part 1</td>
</tr>
<tr>
<td>3</td>
<td>Slider/Cylinder</td>
<td>Intersection, Parts 1 &amp; 2</td>
</tr>
<tr>
<td>4</td>
<td>Rigid</td>
<td>Construction point @ Part 3</td>
</tr>
<tr>
<td>5</td>
<td>Roll w/o Sliding</td>
<td>Intersection, Parts 2 &amp; 3</td>
</tr>
<tr>
<td>6</td>
<td>Center</td>
<td>Center, Part 3</td>
</tr>
<tr>
<td>7</td>
<td>Slider/Cylinder</td>
<td>Intersection, Parts 3 &amp; 4</td>
</tr>
<tr>
<td>8</td>
<td>Center</td>
<td>Center, Part 4</td>
</tr>
</tbody>
</table>

Vector Loop

A closed vector loop is created by starting at any joint that interfaces two parts.
Vectors from that point are added tip-to-tail and must pass through the next joint to the
adjacent mating part in the assembly. The loop proceeds from mating part to mating part,
passing through the joint at the point of contact. Each vector is digitized by "catching" a
point at the head of the vector. That point then becomes the tail of the next vector. Each
digitized point represents a joint or continuation point in the assembly. In the example
problem, the joints are numbered in the sequence in which the vectors were created. The
vector loop proceeds from part-to-part until it arrives back at the starting point.
Vector Construction

Joints 1, 3, 4, 5, and 7 are located by catching an intersection between two geometric entities. Each is the tangent point between two circles or a circle and a plane. The "intersection" option is chosen from the ME-10 menu and the two entities are selected. If the drawing is not accurate, there could be two intersection points or none at all. The zoom function may be used to check the accuracy and select each point.

Joints 2 and 6 are located by catching the center of a circle. The "center" option is chosen from the ME-10 CATCH menu, and then a point on the perimeter of the circle is selected.

Vector Modeling Rules

Vector directions are determined by the VAM modeling rules. Joints 1 and 7 describe contact between two cylinders. Both the "incoming" and "outgoing" vectors must pass through the center of curvature and the point of contact. Hence, both vectors act normal to the surface at the contact point. Joints 3 and 5 describe sliding contact between a cylinder and a plane. On the cylindrical part, the vector must pass through the center of curvature and the contact point. On the planar part, the vector must be parallel to the direction of sliding. Joints 2, 6, and 8 are centers. Vectors simply pass through these points. Their directions are determined by adjacent joints. Joint 4 is a rigid joint and is used to turn a corner and continue the vector loop. Here also the vector directions are determined by adjacent joints.

Two special cases are noted here. The sliding vectors at Joints 3 and 5 have been combined into a single vector. This reduces the number of dependent variables. Joint 8 is merely a pass-through point. Since, the relative angle between the incoming and outgoing vectors is zero, this joint could have been omitted.
Table 6.1.2 List of vectors occurring in the over-running clutch.

<table>
<thead>
<tr>
<th>Vector</th>
<th>Nominal Size (mm)</th>
<th>Tolerance</th>
<th>Relative Angle(deg.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>22.86</td>
<td>0.013</td>
<td>180</td>
</tr>
<tr>
<td>b</td>
<td>22.86</td>
<td>0.013</td>
<td>(\alpha_1 = 7)</td>
</tr>
<tr>
<td>c</td>
<td>24.84</td>
<td>(T_c)</td>
<td>-90</td>
</tr>
<tr>
<td>d</td>
<td>32.23</td>
<td>0.014</td>
<td>90</td>
</tr>
<tr>
<td>e</td>
<td>22.86</td>
<td>0.013</td>
<td>0</td>
</tr>
<tr>
<td>f</td>
<td>22.86</td>
<td>0.013</td>
<td>(\alpha_2 = -7)</td>
</tr>
<tr>
<td>g</td>
<td>50.80</td>
<td>0.078</td>
<td>180</td>
</tr>
<tr>
<td>h</td>
<td>50.80</td>
<td>0.078</td>
<td>0</td>
</tr>
</tbody>
</table>

The first angle is absolute to orient the first vector in the global coordinate system.

Degrees of Freedom

A degree of freedom analysis is performed by HP-CATS 2.0. Table 6.1.3 shows the results of that analysis. In the table, the term RDOF stands for "Removed Degrees of Freedom" by each joint. Rigid and Center joints are ignored, since they do not remove any degrees of freedom. Since there are only two joints on each part, there are no tertiary joints to list.

Table 6.1.3 DOF results for the over-running clutch assembly.

<table>
<thead>
<tr>
<th>Primary Part</th>
<th>Joint</th>
<th>RDOF</th>
<th>Secondary Joint</th>
<th>RDOF</th>
<th>Tertiary Joint</th>
<th>RDOF</th>
<th>Other Joint</th>
<th>RDOF</th>
<th>Net DOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

Assembly DOF 0
Of significance is the fact that at least one joint on each part had to be defined as rolling without slipping. In this case, Joints 1 and 5 were so defined, but Joints 3 and 7 would also work. This was necessary to prevent the rollers from rotating unconstrained about their centers and to prevent the hub from sliding freely in the horizontal direction. In other words it was assumed that there was sufficient friction to prevent slipping. This assumption was a modeling necessity, since sliders ordinarily remove only one DOF and the assembly would have had a non-zero resultant DOF.

The two sliding joints on the hub act in parallel and are therefore unable to fully constrain its motion. The two sliding joints on each roller and the outer cage produce circumferential sliding motion, which is another case of "parallel" constraints which are insufficient to fully constrain motion. The designer may have to modify his model when such parallelism occurs, as was done in this example by redefining two joints as rotating without slipping. HP-CATS will alert you by reporting unremoved degrees of freedom.

ANSI-Y14 Features

Once the Vector Assembly Model has been created, the ANSI-Y14 features may be inserted into the vector loop. At each contact joint, there could be the occurrence of two ANSI-Y14 feature tolerances, one tolerance for each mating part. The sensitivities due to the ANSI-Y14 features are associated with a nominal dimension vector. A more complete discussion may be found in Chapter 4.0.

Table 6.1.4 shows the possible ANSI-Y14 features which may occur in the sample problem. The heading, Matrix Rep. is the matrix form in which the ANSI-Y14 feature tolerance would be represented.
Table 6.1.4 ANSI-Y14 features in the over-running clutch problem.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Roundness</td>
<td>0.0005</td>
<td>Translation</td>
</tr>
<tr>
<td>b</td>
<td>Roundness</td>
<td>0.0005</td>
<td>Translation</td>
</tr>
<tr>
<td>e</td>
<td>Roundness</td>
<td>0.005</td>
<td>Translation</td>
</tr>
<tr>
<td>f</td>
<td>Roundness</td>
<td>0.0005</td>
<td>Translation</td>
</tr>
<tr>
<td>g</td>
<td>Roundness</td>
<td>0.001</td>
<td>Translation</td>
</tr>
<tr>
<td>h</td>
<td>Roundness</td>
<td>0.001</td>
<td>Translation</td>
</tr>
</tbody>
</table>

Conclusion

In this assembly, the tolerance of each part is an important factor in the relative position of the rollers in the assembly. If the cage has some concentricity tolerance, then the cage shifts up and down during rotation. This movement essentially reduces the relative size of the hub. The concentricity would then push Part 3 further to the right and thus increase angle α. Once the hub rotates 180⁰, Part 3 will move back to the left beyond its nominal point and thus decrease angle α. If the hub and the cage were to be fixed to a shaft center, then there could be error both in the hub and in the cage due to the concentricity error in their respective shafts. This would cause angle α to vary independently with respect to size variation, but dependently due to concentricity.

This is not the only way to model this device. The two rollers could be assumed to have unequal α by creating an independent loop for each roller and including the drive shaft at Joint 8.
6.2 Three Bar Truss

The three bar truss is another example illustrating positional tolerance. In Figure 6.2.1 shows the three bar truss composed of two bars pinned at both ends with the ground serving as the third link.

![Diagram of three bar truss](image)

Figure 6.2.1 Sample three bar truss problem.

The object of this analysis is to determine the location of Joint 3, given all of the nominal dimensions and tolerances of the three linkages. A combination of a closed loop and an open loop are used to model this assembly, and by perturbing each vector, the sensitivity of each vector is used to determine the variation in the position of Joint 3.

The global location of Joint 3 is given by \( P_x \) and \( P_y \) which are dependent on all three links, Vectors \( a, b, \) and \( c \). If Vector \( a \) is lengthened, Joint 3 will shift up and to the left. If Vector \( b \) is lengthened, then Joint 3 will shift down and to the right. If Vector \( c \) is lengthened, then Joint 3 will move up and to the right.
Preparation

Construction lines should be drawn through the links such that the construction lines intersect at Joints 1, 2, and 3. This must be done so that the center of the joints may be selected accurately. This is important since there is no catch option to select the center of a boxed end. In Part A of Figure 6.2.2, construction lines are placed through the center of the boxed end pin joint. The designer then selects the intersection and defines the vector. In Part B, the pin joint consists of two round end parts. In this case the designer selects the "center" option of the "CATCH" menu and selects a point on the arc. ME-10 will then calculate the center of the arc and catch to that point.

![Figure 6.2.2. Boxed end and rounded end pin joint.](image)

Modeling Procedure

In this example the joint identification is trivial. There are only three joints in the assembly and they are all free pin joints. Table 6.2.1 lists the joints and their locations. Proper identification of pin centers are crucial in this modeling process.
Table 6.2.1 List of joints occurring in the three bar truss.

<table>
<thead>
<tr>
<th>Joint</th>
<th>Type</th>
<th>Located at:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Free Pin</td>
<td>Intersection, Parts 1 &amp; Frame</td>
</tr>
<tr>
<td>2</td>
<td>Free Pin</td>
<td>Intersection, Parts 2 &amp; Frame</td>
</tr>
<tr>
<td>3</td>
<td>Free Pin</td>
<td>Intersection, Parts 1 &amp; 2</td>
</tr>
</tbody>
</table>

*Vector Loop*

The closed vector loop may begin and end at any joint. As long as Joints 1, 2, and 3 are included in the vector loop the VAM will be valid. The open vector loop must start at Joint 1 and end at Joint 3. There are two possible paths: Vector c or Vector a plus Vector b.

*Vector Construction*

The construction lines should intersect at the centers of all three joints. As long as the selection is made on these intersections the vectors will be defined properly.

*Vector Modeling Rules*

Due to the nature of the assembly the modeling rules are trivial and are discussed under the previous headings. Table 6.2.2 shows the vectors occurring in the three bar truss.

Table 6.2.2 List of vectors occurring in the three bar truss.

<table>
<thead>
<tr>
<th>Vector</th>
<th>Nominal Size (mm)</th>
<th>Tolerance</th>
<th>Relative Angle (deg.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Closed Loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>36.0</td>
<td>0.02</td>
<td>-155 (absolute)</td>
</tr>
<tr>
<td>b</td>
<td>70.0</td>
<td>0.02</td>
<td>β</td>
</tr>
<tr>
<td>c</td>
<td>85.0</td>
<td>0.02</td>
<td>γ</td>
</tr>
<tr>
<td></td>
<td>Closed Loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>85.0</td>
<td>0.02</td>
<td>25 (absolute)</td>
</tr>
</tbody>
</table>
The first angle is absolute to orient the first vector in the global coordinate system. There are three dependent angles in this assembly - $\alpha$, $\beta$, and $\gamma$ - and two dependent lengths - $P_x$ and $P_y$.

**Degrees of Freedom**

A degree of freedom analysis is performed by HP-CATS 2.0. Table 6.2.3 shows the results of that analysis. In the table, the term RDOF stands for "Removed Degrees of Freedom" by each joint.

<table>
<thead>
<tr>
<th>Primary</th>
<th>Secondary</th>
<th>Tertiary</th>
<th>Other</th>
<th>Net</th>
<th>DOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 1</td>
<td>Joint 1</td>
<td>RDOF 3</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Part 2</td>
<td>Joint 2</td>
<td>RDOF 3</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Part 3</td>
<td>Joint 2</td>
<td>RDOF 2</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Assembly DOF 0

It is worth while to note that Part 3 is the frame. In the DOF table, the primary contact removes 2 DOF and the secondary contact removes 1 DOF and sufficiently constrains Part 3. Actually, since Part 3 is the frame it has no degrees of freedom and the contacts are considered redundant and remove no degree of freedom.

**ANSI-Y14 Features**

There are no ANSI-Y14 feature controls in this model. The holes at the end of each link could be treated as true position tolerances, but since each link in one-dimensional, position errors can only change the length between centers or the length of the link. Therefore, dimensional variations are sufficient. If the Frame (Vector a) is not perfectly
vertical, an angularity feature tolerance could be added to Vector \( \mathbf{a} \), which would be represented by a translational Joint 2.
6.3 Bicycle Crank

This example is taken from the peddle crank assembly of a bicycle. [9] Three main parts comprise the bicycle crank assembly: 1) the Crank, 2) the Pin, and 3) the Shaft as shown in figure 6.3.2. An open loop is used to determine the clearance tolerance of the bicycle crank assembly.

![Diagram of bicycle crank assembly]

Figure 6.3.1 Sample bicycle crank assembly.

The clearance must be allocated such that when the nut is tightened over the threads extending out of the assembly, the pin will be sinched up snugly. If the angle on the shaft is too small, then the pin will extend out of the assembly and the assembly will be loose and inoperable. Since the tolerance of the clearance depends on the tolerance of several non-linear parts, a vector loop is necessary to represent the path to which the tolerance will be allocated. In Figure 6.3.2 the vector loop of the assembly is shown along with the joints. Each vector is labeled with a letter and the joint is labeled by a number in a circle.
Figure 6.3.2 The vector loop from the bicycle crank assembly.

*Drawing Preparation*

The bicycle crank assembly is comprised of only three parts, but this simple assembly requires careful planning prior to creating the vector loop. Once the part drawings are overlaid onto the assembly drawing, the designer must use construction lines to locate intersections at Joints 2, 3, 5, 7, and 8. According to the constraints in this assembly, Vectors e, f, g, and h must be normal to Vector i and pass through the center of curvature. This constraint affects the location of Joints 5, 7, and 8. These joints must be identified and marked prior to the creation of the VAM. Joints 2, 3, 7, and 8 are associated with a dimension, but do not occur at an intersection or a vertex and must be marked. All other joints are either at an intersection of two geometric entities or at a vertex.
Modeling Procedure

Table 6.3.1 lists the joint types and their occurrence in the vector loop.

Table 6.3.1 List of joints occurring in the bicycle crank assembly.

<table>
<thead>
<tr>
<th>Joint</th>
<th>Type</th>
<th>Located at:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rigid</td>
<td>Rim of Pin hole</td>
</tr>
<tr>
<td>2</td>
<td>Center</td>
<td>Center, Pin hole</td>
</tr>
<tr>
<td>3</td>
<td>Rigid</td>
<td>Center line, Shaft hole</td>
</tr>
<tr>
<td>4</td>
<td>Center</td>
<td>Center, Shaft hole</td>
</tr>
<tr>
<td>5</td>
<td>Roll w/o Sliding</td>
<td>Intersection, Shaft &amp; Crank</td>
</tr>
<tr>
<td>6</td>
<td>Center</td>
<td>Center, Shaft</td>
</tr>
<tr>
<td>7</td>
<td>Rigid</td>
<td>Rim of Shaft</td>
</tr>
<tr>
<td>8</td>
<td>Planar</td>
<td>Intersection, Pin &amp; Shaft</td>
</tr>
<tr>
<td>9</td>
<td>Rigid</td>
<td>Corner, Pin</td>
</tr>
<tr>
<td>10</td>
<td>Planar</td>
<td>Corner, Pin</td>
</tr>
</tbody>
</table>

Vector Loop

Since an open loop will be used to model this assembly, the vector loop must begin at one end of the clearance vector and end at the opposite end. The length of the clearance vector is defined as the distance between the beginning and ending point. The direction of the clearance vector is consistent with the direction of the vector loop.

Vector Construction

Joints 1, 5, 9, and 10 are located by catching an intersection between two geometric entities. Each is a vertex or the tangent point between two circles. The "intersection" option is chosen from the ME-10 menu and the two entities are selected. In the case of Joint 5, if the drawing is not accurate, there could be two intersection points or none at all. The zoom function maybe used to check the accuracy and select each point.
Joints 4 and 6 are located by catching the center of a circle. The "center" option is chosen from the ME-10 menu, and then a point on the perimeter of the circle is selected.

Joints 2, 3, 7, and 8 are selected from the intersecting construction lines at these points. The "intersection" option is selected from the ME-10 menu and the intersection point digitized.

**Vector Modeling Rules**

Joints 2 and 3 occur at a center. Unlike the continuation center joint, these joints occur on an circle in which the radius of curvature is shown in profile. Thus, they do not have an associated dependent angle.

The designer must note that there are two centers that lie in the crank. There is the center of the Shaft, and slightly above that, the center of the Shaft hole. Joint 4 occurs at the center of the Shaft hole within the Crank, while Joint 6 occurs at the center of the Shaft.

Vectors e and f are used to identify Joint 5 as a Higher Sliding joint. Vector e enters Joint 5 from the center of the Shaft Hole, and Vector f leaves Joint 5 to the center of the Shaft. Both of these vectors obey the modeling rule by passing through the center of curvature and the contact point.

Joint 6 is merely a pass-through point between Vectors f and g. Since the relative angle between the incoming and outgoing vectors is zero, this joint could have been omitted. Joint 7 is on the perimeter of a circle and is not a contact joint. This joint is used to continue the vector loop through the assembly and therefore, it is modeled as a Rigid joint.
Table 6.3.2 List of vectors occurring in the over-running clutch.

<table>
<thead>
<tr>
<th>Vector</th>
<th>Nominal Size (mm)</th>
<th>Tolerance</th>
<th>Relative Angle (deg.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>4.760</td>
<td>0.0037</td>
<td>-90</td>
</tr>
<tr>
<td>b</td>
<td>12.610</td>
<td>0.0042</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>13.550</td>
<td>0.1875</td>
<td>-90</td>
</tr>
<tr>
<td>e</td>
<td>7.850</td>
<td>0.0005</td>
<td>$\alpha_1 = 85$</td>
</tr>
<tr>
<td>f</td>
<td>7.830</td>
<td>0.0005</td>
<td>180</td>
</tr>
<tr>
<td>g</td>
<td>7.830</td>
<td>0.0005</td>
<td>0</td>
</tr>
<tr>
<td>h</td>
<td>3.300</td>
<td>0.0075</td>
<td>180</td>
</tr>
<tr>
<td>i</td>
<td>8.430</td>
<td>0.0075</td>
<td>90</td>
</tr>
<tr>
<td>j</td>
<td>7.300</td>
<td>0.0075</td>
<td>-95</td>
</tr>
<tr>
<td>c</td>
<td>5.127</td>
<td>$T_c$</td>
<td>85</td>
</tr>
</tbody>
</table>

Degrees of Freedom

A degree of freedom analysis is performed by HP-CATS 2.0. Table 6.3.3 shows the results of that analysis. Joints 5, 8, and 10 are the only mating contact joints between two parts. There were no tertiary joints.

Table 6.3.3 DOF results for the bicycle crank assembly.

<table>
<thead>
<tr>
<th>Primary</th>
<th>Secondary</th>
<th>Tertiary</th>
<th>Other</th>
<th>Net</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part</td>
<td>Joint</td>
<td>RDOF</td>
<td>Joint</td>
<td>RDOF</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>2</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>1</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

| Assembly DOF | 0 |

It is worth while to note that the Crank (Part 1) is the frame. In the DOF table, the primary contact removes 1 DOF and the secondary contact removes 2 DOF and sufficiently constrains Part 1. Actually, since Part 3 is the frame it has no degrees of freedom and the contacts are considered redundant and remove no degrees of freedom. Also, in the secondary contact, Joint 10 removes two DOF from Part 1, but removes only one DOF.
from Part 2. This is because there was only one DOF remaining in Part 2 after the action of the primary contact. If two more DOF were removed from Part 2, the resulting DOF would be negative. The system does not do this since anything over three DOF removed is redundant.

ANSI-Y14 Features

Table 6.3.4 shows the possible ANSI-Y14 features which may occur in the sample problem.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Cylindricity</td>
<td>0.001</td>
<td>Translation</td>
</tr>
<tr>
<td>d</td>
<td>Concentricity</td>
<td>0.005</td>
<td>Translation</td>
</tr>
<tr>
<td>e,f</td>
<td>Roundness</td>
<td>0.002</td>
<td>Translation</td>
</tr>
<tr>
<td>g,h</td>
<td>Roundness</td>
<td>0.002</td>
<td>Translation</td>
</tr>
<tr>
<td>i</td>
<td>Angularity</td>
<td>0.005</td>
<td>Rotation</td>
</tr>
<tr>
<td>j</td>
<td>Cylindricity</td>
<td>0.001</td>
<td>Translation</td>
</tr>
</tbody>
</table>

The angularity associated with Vector i is the angular tolerance of the Pin (Part 2).

Conclusion

At this point, both the shaft and the pin are constrained properly and the VAM is ready for analysis.
7.0 Conclusions and Recommendations

7.1 Research Contributions

The following contributions resulted from this research:

1. A generalized system for modeling 2-D mechanical assemblies for tolerance analysis was developed. The main features are: i) use of vector loops to model mechanical assemblies with each vector to representing a nominal part dimension, ii) matrix representation of the vectors and ANSI-Y14 features by translational and rotational matrices, and iii) reduced modeling time by eliminating the need for deriving algebraic equations.

2. Definition of joint types and a systematic procedure for determining dependent variables. The joint types not only include the mating part joints, but also continuation joints which permit controlled dimensions to be used as vectors. The Joint Sequence Method of determining dependent variables automates the process by which dependent vectors are identified in the VAM.

3. The development of a Degree of Freedom (DOF) analysis method which automates the Vector Assembly Model verifying process. The Contact Sequence Method of removing DOF not only automates the VAM verifying process, but is able to handle redundant joints.

4. The development of 2-D modeling rules for the creation of valid assembly models. A comprehensive set of modeling rules has been determined which assist the designer in modeling complex assemblies.

5. The development of a model for ANSI-Y14 Features and a method to incorporate ANSI-Y14 models in the VAM. As a result, the propagation of form and feature
variations through an assembly may now be predicted by statistical or worst case methods.

6. The specification of the functional requirements and the data requirements for a two-dimensional tolerance analysis modeler on a CAD system.

7. The coding of a prototype program as a conceptual design tool on a commercial CAD system. This functioning modeler has been developed to test modeling procedures, user interface design, data structures and system integration issues.

8. The development of a method to extract tolerance analysis data graphically from a CAD geometric database.

9. The integration of the two-dimensional tolerance analysis models as a part of the CAD geometric database so they can be saved along with the engineering drawing set. This data can be passed to manufacturing to document the engineering analysis used to establish tolerance requirements. As a result, proposed production design changes can be made more knowledgeably.

10. The design of a file specification for transfer of data to CATS.BYU. This file stores all geometric and numerical data of the modeled assembly. This neutral file format is a standard format allowing the file to be read by geometric modelers on other systems.
7.2 Current Status

A functioning 2-D modeling system has been developed on a commercial CAD system featuring:

1. Vector and Matrix representations for mechanical assembly analysis.
2. Kinematic model of assemblies.
3. Joint definitions of both mating part joints and continuation joints.
4. Method of defining dependent variables from joint types using the Joint Sequence Method.
5. Built in degree of freedom analysis to assure model completeness using the Contact Sequence Method.
6. ANSI-Y14 feature control representations using rotational and translational matrices.

The graphic modeling software operates on an HP Series 350 workstation, using the ME-10 CAD system. By means of this prototype system, we have:

1. Investigated user interfaces to determine the most ergonomic and functional geometric modeler for the designer.
2. Investigated CAD macro utilities for creating models.
3. Investigated CAD database requirements to incorporate the VAM model data directly into the CAD geometric database.
4. Augmented the CAD data structure to include tolerance analysis models.
The user interface and set of modeling rules have been implemented in a way which:

1. Assists designers in creating models by providing a tutorial and examples of complex assemblies.

2. Provides less chance for error. Human error may be reduced due to less keystroke entry and modeling error by adherence to the modeling rules.

3. Is reasonably easy for a designer to learn.

A Designer's Manual has been written and is included in reference 11 in the Appendix.

7.3 Remaining Issues

The following issues remain to be resolved:

1. Once CATS.BYU has allocated all tolerances in an assembly, should the interface HP-CATS 2.0 automatically update the drawing? This could be automated by saving the drawing attributes of each dimension and tolerance extracted from the drawing and using these as pointers to the original location in the drawing file. The issue is whether or not designers would allow a software package to modify their drawings.

2. A valid model for true position features as components of a 2-D vector loop must be developed. Pin joints are a special case. A model which includes clearance and position errors in arbitrary directions is needed. The current model is only a conservative approximation.
3. Feature control tolerances must be made compatible with worst case analysis, i.e. form tolerances on a part should be zero at Maximum Material Condition (MMC). Currently, form tolerances are treated independent of size tolerances. Under worst case they can add to exceed the envelope of perfect form at MMC.

4. More complete internal data integrity verification is needed.

7.4 Recommendations

It is recommended that future efforts include the following:

1. Internal package improvements:
   a. More rigorous error trapping.
   b. Enhancements to allow the insertion and deletion of vectors and ANSI-Y14 features in a VAM.

2. A designer interface and a file format should be developed to store mating hole patterns for tolerance analysis.

3. Explore the possibility of a complete turnkey CAD-Tolerance Analysis Modeler. The package would include Dynamic Vector Sensitivity Propagation (DVSP) which is the real time geometric update of allocated tolerances. DVSP would provide real-time graphic updating as CATS.BYU allocates tolerances and would highlight any interferences.
4. Develop a translator to read in popular geometric neutral files such as IGES and DXF with tolerance analysis data. This would provide interchangeable tolerance analysis drawings between systems. This raises questions about standards for assembly tolerance models which would have to be recognized by major CAD vendors. An alternative would be to have IGES and DXF capability in CATS.BYU itself. But this would require that much of the geometric modeling capabilities of HP-CATS would have to be recreated in CATS.BYU.

5. Investigate the possibility of automating the loop definition tasks of the Vector Assembly Modeling process. The designer would then only be required to define all the joints of the assembly and the modeling interface would create a complete and valid system of loops for the VAM.
Appendix A
Programmer's Manual

HP-CATS 2.0 consists of a Macro program to gather data, and three Fortran 77 programs to extract that data from the MI assembly file. The combination of Fortran 77 and Macro language gives HP-CAT 2.0 the needed geometric manipulation and analytical capability.

The Macro language is a sequentially interpreted and executed language. Macros may be written directly from ME-10 or be written from a shell using the VI editor. Once a Macro is written, it is input into ME-10 and executed by typing the Macro name in the ME-10 command line. Refer to Hewlett-Packard's ME-10 user's manual for more information on Macros.

Note: HP's ME-10 user's manual provides only an introductory level of Macro understanding. For a more in-depth review of Macros, it would be beneficial to the designer to review this section closely. ME-10's on-line HELP provides detailed explanation of all Macro commands, functions, and statements.

Within the Macro section of HP-CATS 2.0, Macros are modularized to simplify maintenance. If the designer is familiar with Macro commands it is not difficult to read a Macro top-down to learn its functions. The Macro interface will be explained thus: 1) a short description of each macro will be given followed by a call list, 2) the page number in which the Macro appears in the Appendix, and 3) a variable list with its functions will be listed.
Macro: **Cats_loop_2**

**Description:** This is the main driver program for HP-CATS 2.0. This Macro calls macros to read assembly and loop descriptions, drawing numbers, create vectors, dimension the vectors, store all information out to the MF file, etc.

**Variables:** Vector_type - Flag to determine if the designer is creating vectors or ANSI-Y14 features. Vector_type is set as "Vector."

Called by: Sm_screen_2
Sm_screen_2mains
Another_loop
Another_loops

Calls: Call_initiate
Call_pt2
Call_end_loop
Call_part_name
Call_spaces
Call_edit
Call_arrow
Call_add_info
Call_place_name
Call_increment

Macro: **Feature_control**

**Description:** This is the second main Macro driver. This Macro allows the designer to insert ANSI-Y14 feature controls into the vector loop. The Macro prompts the designer for the loop number and the two vector numbers. With this information the Macro prompts the user for the feature to be associated with the first part and the feature to be associated with the second part. Once all of the information has been input, the Macro formats the information like the vector line information and writes the information line to the loop block for retrieval by the Fortran 77 program.

**Variables:** Vector_type - Flag to determine if the designer is creating vectors or ANSI-Y14 features. Vector_type is set as "Feature."
Fea_loop_no - Number of loop in which ANSI-Y14 feature will be inserted.
Fea_pt_no - Mating pair to which the feature will be assigned.

Called by: Sm_screen_cat2

Calls: Call_initiate
Call_part_menu
Fea_pt_no
Sm_screen_2main
Call_end_loop
Sm_screen_2feature
Call_part_name
Call_spaces
Call_edit
Call_add_info
Call_place_name
Call_increment
Macro: Call_initiate

Description: Since the Macro language is a sequentially interpreted, if a variable is called later in the Macro, that variable must be defined prior the the call. Call_initiate defines all counters and variables that will be used in HP-CATS 2.0. All of the variables that are initialized are listed in the variable section.

Variables:
- Draw_scale - Drawing scale of the assembly drawing.
- C - Original color setting
- DC - Original dimension color setting
- L - Original Line type
- Counter - Incrementing counter initialized to 1
- Input_counter - Input Counter initialized to 1
- Loop_dir - Loop direction initialized to '2D'.
- Loop_name - Name of the loop. Read as character string.
- Loop_number - Number of the loop. Read as number.
- Layer_no - Layer number is loop number plus 100.
- Text_loc - Point location to place loop block.
- Ds - Another name for Draw_scale.
- Xtl - X coordinate of Text_loc.
- Ytl - Y coordinate of Text_loc.

![Diagram of Loop Block and Associated Points]

Figure 8.1. Loop block and associated points.

- Ax - X coordinate of Point A
- Ay - Y coordinate of Point A
- Bx - X coordinate of Point B
- By - Y coordinate of Point B
- Cx - X coordinate of Point C
- Cy - Y coordinate of Point C
- Dx - X coordinate of Point D
- Dy - Y coordinate of Point D
- Ex - X coordinate of Point E
- Ey - Y coordinate of Point E
- Fx - X coordinate of Point F
- Fy - Y coordinate of Point F
- Gx - X coordinate of Point G
- Gy - Y coordinate of Point G

Called by:
- Cats_loop_2
- Feature_control

Calls:
- Color_menu
- Sm_screen_2main
- Call_loop_description
- First_stack
Macro:  First_stack

Description: Prompts the designer to select the starting point of the vector loop. Reads this point, and
assigns the exact coordinate to which the cross hairs catch to X1 and Y1. A point is places
at P1 and this is the start of the vector. This macro then rounds X1 and Y1 to four
significant digits and reassigns the rounded number to Xp1 and Yp1.

Variables:  
P1 - First point of stack
X1 - X coordinate of P1
Y1 - Y coordinate of P1
Xp1 - X1 rounded to four significant digits
Xx1 - X1 rounded to four significant digits
Yp1 - Y1 rounded to four significant digits
Yy1 - Y1 rounded to four significant digits

Called by:  Call_initiate
            Edit_vec

Calls: none

Macro:  Call_pt2

Description: Prompts the designer for the second point (head) of the vector. Locates the exact coordinate of
the second point and places a point at that location.

Variables:  
P2 - Second point of the vector, read as PNT_XY.
X2 - X coordinate of point 2
Y2 - Y coordinate of point 2

Called by:  Cats_loop_2
            Edit_vec

Calls: none

Macro:  Call_end_loop

Description: Tests to see if the designer wished to end the loop. If the token is "y" then the loop is
terminated and Call_exit_loop is called to end the loop. If the token is "N", then loop is
not terminated and main menu is again displayed.

Called by:  Cats_loop_2
            Feature_control
            Add_vector

Calls: Make_sure_menu
       Call_exit_loop
       Sm_screen_2feature
Macro: \texttt{Call\_part\_name}

Description: The counter is read in as a parameter and appended to a character string to create the part name. Then it checks to see if the vector type is a vector or a feature. If the vector type is a feature it calls \texttt{Check\_joint\_type} and \texttt{Call\_feature\_check}. Else if the vector type is a vector then \texttt{Call\_joint\_type} and \texttt{Check\_joint\_type} is called. The main menu is displayed and the designer is prompted to see if the part has a fixed tolerance. The designer is then prompted for the tolerance type. If the tolerance type is "+/-" then the designer is prompted for a single tolerance. If the tolerance type is "upper/lower" then the designer is prompted for an upper and a lower tolerance. The tolerance is formatted to "+/-".

Variables:  
- \texttt{Pt\_name}: Part name that will be displayed on the vector.  
- \texttt{Prefix}: Parenthesis added to the \texttt{Pt\_name}.  
- \texttt{Bought}: Tolerance type, fixed or not fixed.  
- \texttt{Tol\_format}: Tolerance format, +/- or upper/lower.  
- \texttt{Tolerance\_in}: If \texttt{Tol\_format} +/-, Tolerance\_in is the single tolerance read.  
- \texttt{Tol\_plus}: Plus tolerance.  
- \texttt{Tol\_minus}: Minus tolerance.  
- \texttt{Pm\_tol}: +/- format of tolerance/parameters that were read.  
- \texttt{Ast}: - If the tolerance is fixed, Ast = 1. If the tolerance is not fixed, Ast = 0.

Called by:  
- \texttt{Cats\_loop\_2}  
- \texttt{Feature\_control}  
- \texttt{Call\_info\_p2}

Calls:  
- \texttt{Check\_joint\_type}  
- \texttt{Call\_feature\_check}  
- \texttt{Check\_joint\_type}  
- \texttt{Sm\_screen\_2main}  
- \texttt{Tolerance\_table}  
- \texttt{Round\_off}

Macro: \texttt{Call\_spaces}

Description: Formats the Part name to 12 character length.

Variables:  
- \texttt{Pt\_name}: 12 character part name.  
- \texttt{Spaces}: "()"

Called by:  
- \texttt{Cats\_loop\_2}  
- \texttt{Feature\_control}  
- \texttt{Call\_info\_p1}  
- \texttt{Call\_info\_p2}

Calls:  
- \texttt{Call\_fix\_string}
Macro: Call_edit

Description: Prompts the designer to see which vector entity is to be edited. Once the designer selects an entity, the proper Macro is called to make the changes.

Variables:
- Bought1 - If Bought = 1, then Bought1 = "Fixed". If Bought = 0, then Bought1 = "Not Fixed"
- Edit_number - Number of the entity to be edited.

Called by:
- Cats_loop_2
- Feature_control
- Call_info_p2

Calls:
- Edit_menu
- Redo_name
- Redo_equiv_name
- Redo_feature
- Redo_dimension
- Redo_tolerance
- Redo_format
- Sm_screen_2main
- Redo_constraint
- Redo_joint_type
- Edit_menu
- Redo_mate_part

Macro: Call_arrow

Description: Takes the second point of the vector, calculates the positions of the arrow head and draws the arrow head.

Variables:
- X1 - X coordinate of P1
- Y1 - Y coordinate of P1
- X2 - X coordinate of P2
- Y2 - Y coordinate of P2
- Avg_dim - Distance between P1 and P2
- Xnot - X coordinate of arrow reference
- Ynot - Y coordinate of arrow reference
- Ppos_x - X coordinate of the arrow base to the left of P2
- Ppos_y - Y coordinate of the arrow base to the left of P2
- Pneg_x - X coordinate of the arrow base to the right of P2
- Pneg_y - Y coordinate of the arrow base to the right of P2

Called by: Cats_loop_2

Calls: none
Macro: **Call_add_info**

Description: This macro appends all the vector entities into a character string and adds this line to the vector line.

Variables:
- **Pt_name**  - Vector name
- **Line_1**  - (Pt_name+"*"+Tol_format+"*"+STR(Tol_plus)+"*")
- **Line_2**  - (STR(Tol_minus)+"*"+Ast+"*"+Joint_type+"*")
- **Line_3**  - (STR(First_part)+"*"+STR(Second_part)+"*")
- **Line_4**  - (STR(Ref1_x)+"*"+STR(Ref1_y)+"*"+STR(Ref2_x)+"*"+STR(Ref2_y)+"*)
- **Line_5**  - (STR(Xx1)+"*"+STR(Yy1)+"*")
- **Line_all**  - (Line_1+Line_2+Line_3+Line_4+Line_5)
- **X_mid**  - X coordinate of the center of the vector.
- **Y_mid**  - Y coordinate of the center of the vector.
- **Pnt_on_vector**  - Point with reference
- **Ref_pt1**  - first point of the reference point(.123,.123) means that there is no point. A point is defined for future reference.

Called by: **Cats_loop_2**
**Feature_control**

Calls: none

Macro: **Call_place_name**

Description: If the Vector type is feature, then P2 is only .000001 from P1. This is done because ME-10 will not dimension zero dimension. The dimension text is placed between P1 and P2. If the tolerance type is +/- then +/- tolerance is added. If the tolerance type is upper and lower, then upper and lower tolerance is added.

Variables:
- **P3**  - Location for dimension text.

Called by: **Cats_loop_2**
**Feature_control**

Calls: **Sm_screen_2main**

Macro: **Call_increment**

Description: This Macro increments the counter by one and sets X2 to X1, Y2 to Y1, and P2 to P1.

Variables:
- **Counter**  - Vector counter
- **Input_counter**  - Input counter
- **P1**  - Head of the vector P2 is renames to P1, or the tail of the next vector.
- **X1**  - X coordinate of P1
- **Y1**  - Y coordinate of P1
- **Xx1**  - Rounded value of X1
- **Yy1**  - Rounded value of Y1
- **Feature_counter**  - ANSI-Y14 feature counter. Number of features added to the vector loop.

Called by: **Cats_loop_2**
**Feature_control**

Calls: **Sm_screen_2feature**
Macro: Call_exit_loop

Description: If the vector type is a vector then the clearance and the clearance tolerance is prompted and rounded. Then the loop number, Xp1, Yp1, D_c, and Dct are appended to a character line. This character line is added to the loop block to be stored in the MI assembly file. An invisible file "edit.no" is opened and the name of the assembly file is written. Fortran 77 program to determine the DOF is executed. The Fortran 77 program reads "edit.no" and reads the assembly file name to analyze.

Variables:
- D_c - Desired Clearance
- Dct - Desired Clearance Tolerance
- Line3 - (<"+STR(Loop_number)+"*"+STR(Xp1)+"*"+STR(Yp1)+"*"+STR(D_c)+"*"+STR(Dct)+"*)

Called by:
- Call_end_loop

Calls:
- Tolerance_table
- Sm_screen_cat2

Macro: Write_descriptions

Description: This macro checks how many lines of Assembly description, Loop description, and Drawing numbers and adds all description lines to the loop block.

Variables:
- Ad1 to Ad10 - Up to 10 lines of assembly description.
- Ld1 to Ld10 - Up to 10 lines of loop description.
- Dn1 to Dn20 - Up to 20 lines of drawing numbers. The drawing numbers are read as characters.

Called by:
- Call_initiate

Calls:
- none

Macro: Color_menu

Description: Customize the menus to display the different color options.

Variables:
- none

Called by:
- Call_initiate

Calls:
- T_clear_menu

Macro: Sm_screen_cat2

Description: This is the main menu 1.

Variables:
- File_name_assem - Since this variable is to be displayed in the main menu, this variable is defined at this point.

Called by:
- Call_exit_loop
- Mod_vec
- Sm_screen_1

Calls:
- T_clear_menu
Macro: Mod_vec
Description: This menu displays the MAIN MENU 2. This menu primarily displays the vector editing options.
Variables: none
Called by: Sm_screen_cat2
           New_loop
Calls: T_clear_menu

Macro: View_dof
Description: This Macro opens "dof.mi", "dof.mi" is the file which the Fortran 77 DOF analysis programs writes the DOF left in the assembly.
Variables: none
Called by: Mod_vec
Calls: T_clear_menu

Macro: New_loop
Description: This macro just prompts the designer for the MI assembly name and reads the name.
Variables: File_name_assem - Name of assembly file to which the MI file will be written.
Called by: Sm_screen_cat2
           Mod_vec
           Loop_direction
Calls: Mod_vec

Macro: Load_file
Description: Clears the monitor of all drawings and loads the MI assembly file.
Variables: File_name_assem
Called by: Mod_vec
Calls: none
Macro: Edit_loop

Description: Read "edit.no" to find the name of the MI assembly file, then runs a Fortran 77 program which reads the MI assembly file and writes all loop information to be edited to "edit.mi." "edit.mi" is then opened in the ME-10 editor to be viewed by the designer and edited. Another Fortran 77 program is run to reformat the edited information back into the character string and the MI assembly file is updated with the new information.

Variables: none

Called by: Mod_vec

Calls: Fortran Program: edit
        Fortran Program: mi

Macro: Edit_vec

Description: This Macro allows the designer to redefine a vector. This means that the designer inputs the loop and number of the vector and this Macro will prompt the user for all vector information. Once all of the information has been entered, this information is appended into a character string and a Fortran 77 program is called to update the MI assembly file.

Variables: Vec_reside - Loop number of the vector.
           Vec_num - Vector number.
           Pt_name - Vector name.
           Line_1 - (Pt_name+"+Tol_formats+"+STR(Tol_plus)+"")
           Line_2 - (STR(Tol_minus)+"+Ast+"+Joint_type+"")
           Line_3 - (STR(First_part)+"+STR(Second_part)+"")
           Line_4 - (STR(Ref1_x)+"+STR(Ref1_y)+"+STR(Ref2_x)+"+STR(Ref2_y)+"")
           Line_5 - (STR(Xx1)+"+STR(Yy1)+"")
           Line_all - (Line_1+Line_2+Line_3+Line_4+Line_5)

Called by: Mod_vec

Calls: First_stack
       Call_pt2
       Call_joint_type
       Check_joint_type
       Sm_screen2main
       Redo_constraint
       Redo_format
       Sm_screen_cat2
       Fortran Program: mi
Macro: Sm_screen_1

Description: This macro is suppose to execute the 1-D version of HP-CATS. Since HP-CATS 1.0 is not included in this interface, a message is flashed to the designer and the main menu is redisplayed.

Variables: none

Called by: Sm_screen_2mains
           Sm_screen_2main
           Sm_screen_2feature
           Call_part_menu
           Tolerance_table
           Make_sure_menu
           Feature_menu

Calls: Sm_screen_cat2

Macro: Sm_screen_2

Description: This macro is the driver of HP-CATS 2.0. It displays the main menu, prompts the designer for the MI assembly file and executes the vector generation macro.

Called by: Sm_screen_cat2

Calls: Sm_screen_2mains
       Loop_direction
       Cats_loop_2

Macro: Sm_screen_2mains

Description: This macro displays the HP-CATS 2.0 MAIN menu for vector type = 'Vector'.

Variables: none

Called by: Sm_screen_2

Calls: T_clear_menu

Macro: Sm_screen_2main

Description: This Macro is the identical menu to Sm_screen_2mains but without the loop direction.

Variables: none

Called by: Feature_control
           Call_initiate
           Call_part_name
           Call_edit
           Call_place_name
           Edit_vec
           Check_joint_type
           Redo_format

Calls: T_clear_menu
Macro: Sm_screen_2feature

Description: This Macro is the main menu for ANSI-Y14 feature insert option.

Variables: none

Called by: Feature_control
           Call_end_loop
           Call_increment

Calls: T_clear_menu

Macro: Add_vector

Description: This Macro allows the designer to insert a vector in the vector loop. This Macro is not complete.

Variables: Vec_name - Name of the vector to be modified.
           Pp1 - Head location of the new vector.

Called by: Mod_vec

Calls: Make_sure_menu

Macro: Call_Pname

Description: This macro is part of Add_vector and the two vector numbers are passed into it as a parameter and new prefixes are assigned to the modified and newly formed vector.

Variables: Pname1 - Prefix for the modified vector
           Pname2 - Prefix for the newly formed vector

Called by: none (this Macro has not been implemented)

Calls: none

Macro: Call_info_p1

Description: This Macro is also a part of Add_vector and has not been implemented.

Variables: P5 - A point near the origin
           X1 - X coordinate of P1
           Y1 - Y coordinate of P1
           X2 - X coordinate of P2
           Y2 - Y coordinate of P2

Called by: Add_vector

Calls: Call_spaces
Macro: Call_info_p2

Page: 38

Description: This Macro is also a part of Add_vector and has not been implemented.

Variables: P5 - A point near the origin
X1 - X coordinate of P1
Y1 - Y coordinate of P1
X2 - X coordinate of P2
Y2 - Y coordinate of P2
Avg_dims - Distance between P1 and P2
Avg_dim - Avg_dims is renamed to Avg_dim.

Called by: Add_vector

Calls: Call_part_name
Call_spaces
Call_edit

Macro: Place_new_name

Description: This Macro is also a part of Add_vector and has not been implemented. It allows the new prefixes to be placed along with the dimension information on the modified and newly created vectors.

Variables: none

Called by: Add_vector

Calls: none

Macro: Input_error

Description: This Macro just displays an input error message to the designer for 1.5 seconds.

Variables: none

Called by: none

Calls: none
Macro: **Put_name**

Description: This Macro is also a part of Add_vector and is not implemented. It reads in several parameters and places the name of the vector to the location as parameter indicates.

Variables:  
- Infront - Prefix of the vector  
- Pre_no - Vector number  
- Tol_format - Tolerance format, '+/-' or 'upper & lower'  
- Tol_plus - Plus tolerance  
- Tol_minus - Minus tolerance  
- Pt1 - Point 1, tail of the vector  
- Pt2 - Point 2, head of the vector  
- Pt3 - Location of the dimension text  

Called by: none  
Calls: none

Macro: **Ydate**

Description: Finds the date & time from the system. This is inserted into the neutral file to mark the date & time of creation.

Variables:  
- To_day - Date and time.  

Called by: none  
Calls: none

Macro: **Edit_menu**

Description: This menu displays all of the vector attributes for editing.

Variables:  
- Pt_name - Part name  
- Avg_dim - Nominal dimension of the vector  
- Tol_plus - Plus tolerance  
- Tol_minus - Minus tolerance  
- Tol_format - Tolerance format, +/- or Upper & Lower  
- Bought1 - Tolerance type, fixed or not fixed  
- First_part - First part of the mating joint  
- Second_part - second part of the mating joint  

Called by: Call_edit  
Calls: none
Macro: Loop_direction

Description: Prompts the designer for the name of the MI assembly loop. Then call the Macros to input assembly description and drawing numbers.

Variables: File_name_assem - Name of the MI assembly file

Called by: Sm_screen_2

Calls: New_loop
Call_assem_description
Call_drawing

Macro: Call_feature_check

Description: Checks to see what ANSI_Y14 symbol the designer selected and writes the equivalent character string into variables.

Variables: Feature_input - The ANSI-Y14 feature the designer selects from the feature menu.
Screen_output - Prefix for ANSI-Y14 feature to be displayed on the screen along with the tolerances.
Edit_output - Prefix that will be displayed on the Edit_menu.
Feature - Variable to be displayed in the Edit_menu, Feature = Edit_output.

Called by: Call_part_name

Calls: Feature_menu

Macro: Call_joint_type

Description: Menu displaying all of the joint types.

Variables: none

Called by: Call_part_name
Edit_vec
Redo_joint_type

Calls: none

Macro: Type_joint

Description: Prompts the designer to select the type of joint from the joint menu.

Variables: Joint_type - Type of joint the designer selected.

Called by: Call_joint_type

Calls: none
Macro: Check_joint_type

Description: Prompts the user to define the first and second part of the mating surface joint. Checks to see if the joint type requires a reference plane. If a reference plane is required, then the designer is prompted to select the two end points of a reference plane. The two end points are divided into the x and y components for future use.

Variables:
- First_part - First part to cross the mating part joint.
- Second_part - Second part to cross the mating part joint.
- Ref_pt1 - First point of the reference plane.
- Ref_pt2 - Second point of the reference plane.
- Ref1_x - X coordinate of Ref_pt1
- Ref1_y - Y coordinate of Ref_pt1
- Ref2_x - X coordinate of Ref_pt2
- Ref2_y - Y coordinate of Ref_pt2

Called by:
- Call_part_name
- Edit_vec
- Redo_joint_type
- Redo_mate_part

Calls:
- Call_part_menu
- Sm_screen_2main

Macro: Call_part_menu

Description: Menu displaying parts 1 - 10 for designer selection.

Variables: none

Called by:
- Feature_control
- Check_joint_type
- Redo_mate_part

Calls:
- T_clear_menu

Macro: Round_off

Description: Reads in a number as a variable and rounds the number to four significant digits. This is done by multiplying the number by 10,000, rounding it to the nearest integer, then dividing y 10,000.

Variables:
- Round_it - Number to be rounded
- Set_number - Number * 10,000
- Rounded_it - Set_number rounded to the nearest integer
- Rounded - Rounded_it divided by 10,000

Called by:
- First_stack
- Call_part_name
- Call_increment
- Call_exit_loop
- Check_joint_type
- Another_loop

Calls: none
Macro:  

Call_fix_string

Description: A character string is read in as a parameter and if the length of the string is less than 12 characters appropriate number of blanks are inserted to make the length equal 12.

Variables: 
The_string - Character string  
String_length - Length of the character string  
Fixed_string - Character string with the appropriate number of spaces

Called by: 
Call_spaces

Calls: 
none

Macro:  

Redo_name

Description: Prompts the designer for the new loop number and redefines the prefix to be placed with the associated vector.

Variables: 
Loop_number - Number of the loop  
Pt_name - New part name  
Prefix - New prefix

Called by: 
Call_edit

Calls: 
none

Macro:  

Redo_dimension

Description: Allows the designer to select a new vector head and consequently, redefine the vector.

Variables: 
P2 - New head of the vector  
Avg_dim - Nominal dimension between P1 and P2

Called by: 
Call_edit

Calls: 
Redo_dimension

Macro:  

Redo_feature

Description: Allows the designer to redefine the ANSI-Y14 feature from the edit menu.

Variables: 
Feature_input - New feature control  
Screen_output - Prefix that will be displayed by the dimensioning statement  
Feature - Variable to be displayed in the Edit_menu, Feature = Edit_output.

Called by: 
Call_edit

Calls: 
Feature_menu
Macro: Redo_tolerance

Description: Allows the designer to redefine the tolerances from the edit menu.

Variables: Tolerance_in - If the tolerance format is '+/-', only one tolerance is required. This
tolerance is Tolerance_in.
Tol_plus - Plus tolerance
Tol_minus - Minus tolerance
Pm_tol - Plus/Minus tolerance format
Trans_operator - Same as Pm_tol

Called by: Call_edit
Redo_format

Calls: Tolerance_table

Macro: Redo_format

Description: Allows the designer to redefine the tolerance format from the edit menu.

Variables: Tol_format - Tolerance format, '+/-' or 'Upper/Lower'

Called by: Call_edit
Edit_vec

Calls: Sm_screen_2main
Redo_tolerance

Macro: Redo_constraint

Description: Allows the designer to redefine the tolerance constraint from the edit menu. Either fixed or not
fixed.

Variables: Bought - Designer defined tolerance constraint

Called by: Call_edit
Edit_vec

Calls: none

Macro: Redo_joint_type

Description: Allows the designer to redefine the joint type from the edit menu.

Variables: none

Called by: Call_edit

Calls: Call_joint_type
Check_joint_type
Macro: Redo_mate_part
Description: Allows the designer to change the mating parts from the edit menu.

Variables: none
Called by: Call_edit
Calls: Call_part_menu
        Check_joint_type

Macro: Call_drawing
Description: Allows 20 part drawings to be entered.

Variables: Draw_counter - Keeps track of number of drawings entered by the designer.
           Dno - Number of drawings entered
           Drawing_number1-10 - The variable names of the 20 different drawing numbers.

Called by: Loop_direction
Calls: none

Macro: Call_assem_description
Description: Allows 10 lined of assembly description to be entered.

Variables: Assem_counter - Counter to keep track of assembly description lines.
           Adn - Number of assembly description lines entered.
           Assem_descript1-10 - The variable names of the assembly description lines.

Called by: Loop_direction
Calls: none

Macro: Call_loop_description
Description: Allows 10 loop description lines to be entered.

Variables: Loop_counter - Counter to keep track of loop description lines.
           Ldn - Number of loop description lines entered.
           Loop_descript1-10 - The variable names of the loop description lines.

Called by: Call_initiate
Calls: none
Macro: Tolerance_table
Description: Displays a set of tolerance on the menu bar.
Variables: none
Called by: Call_part_name
           Call_exit_loop
           Redo_tolerance
           Another_loop
Calls: none

Macro: Make_sure_menu
Description: Menu displaying the options to confirm the exit request, undo the request, or create another loop.
Variables: none
Called by: Call_end_loop
           Add_vector
Calls: Another_loop
       Sm_screen_1
       Sm_screen_2

Macro: Another_loop
Description: Allows the designer to create another loop without exiting out of HP-CATS 2.0.
Variables: D_c - Desired Clearance
            Dct - Desired Clearance Tolerance
            Line3 - ("<+STR(Loop_number)+"+STR(Xp1)+"+STR(Yp1)+"+STR(D_c)+"+STR(Dct)+")"
Called by: Make_sure_menu
Calls: Tolerance_table
       Cats_loop_2

Macro: Another_loops
Description: Just redisplay Cats_loop_2.
Variables: none
Called by: Sm_screen_cat2
Calls: Cats_loop_2
Macro: Feature_menu

Description: Displays all of the ANSI-Y14 features in the menu.

Variables: none

Called by: Call_feature_check
            Redo_feature

Calls: none

Macro: Cats_table

Description: Reads the file the designer inputs and formats that file to ME-10's text format to be displayed on the screen.

Variables: File_name_assem - Name of the file to be displayed

Called by: Sm_screen_cat2

Calls: none

Macro: Cats_analysis

Description: Opens "edit.no" and writes the name of the assembly. Then calls a Fortran program to read the assembly file name and perform analysis prior to formatting the MI file into neutral file format.

Variables: none

Called by: Mod_vec

Calls: Fortran Program: ana
        Call_confirm

Macro: Call_confirm

Description: Writes the contents of "dof.temp" to the screen. If there is no degree of freedom then it calls a Fortran program to continue with the analysis.

Variables: Vert - Vertical spacing for text locations
           Bean - Line counter
           Cont_mark - Continuation marker

Called by: Cats_analysis

Calls: Fortran Program: ana
Macro: End_loop
Description: Just a point located near the origin which is used as a marker to test if the designer wants to end the loop.
Variables: none
Called by: Cats_loop_2
Feature_control
Call_edit
Sm_screen_2mains
Sm_screen_2main
Sm_screen_2feature
Call_drawing
Call_assem_description
Call_loop_description
Calls: none

[********** SOME PERSONAL MACROS **********]
Macro: Sh
Description: Executes the "c" shell from ME-10.
Variables: none
Called by: none
Calls: none

Macro: remenu
Description: Redraws the creation menu.
Variables: none
Called by: none
Calls: none

Macro: Lmn
Description: Lists the directory content of the path specified in the Macro.
Variables: none
Called by: none
Calls: none
Macro:  Quit
Page:  59
Description: Quits out of ME-10 by executing the exit and confirm statements.
Variables:  none
Called by:  none
Calls:  none

Macro:  Em
Description: Short for Edit_macro. Need to type "em" then the macro name from ME-10 to edit a Macro.
Variables:  none
Called by:  none
Calls:  none

Macro:  Be
Description: This macro begins the echo series so that everything the designer does is echoed.
Variables:  none
Called by:  none
Calls:  none

Macro:  Ee
Description: End the echoing command.
Variables:  none
Called by:  none
Calls:  none

Macro:  Sm
Description: Short for Same_macro. Just type "sm" followed by the Macro name to save the Macro.
Variables:  none
Called by:  none
Calls:  none
Macro: Cls
Description: Clears everything from the screen.
Variables: none
Called by: none
Calls: none

Macro: Tron
Description: This Macro turns on the trace and records everything in a file named "temp." This is a good debugging tool.
Variables: none
Called by: none
Calls: none

Macro: Troff
Description: Turn trace off.
Variables: none
Called by: none
Calls: none
Appendix B

REFERENCES


10. Spotts, M. F., Dimensioning And Tolerancing for Quantity Production, Prentice-Hall, 1983

Thesis Prospectus
By: Ki S. Chun

Proposed Title:

Development of a Two-Dimensional Tolerance Modeling Interface to a CAD System.

Introduction:

In mechanical assemblies optimizing the allocation of tolerances among individual parts of an assembly is an important design task. The designer must be aware that tight tolerances mean higher manufacturing cost and reduced production rate. In order to satisfy both the design constraint and the manufacturing constraint the designer must perform a thorough tolerance analysis of the proposed assembly. Tolerance analysis is a tedious but necessary operation which assists in producing an optimal design for both functionality and producibility.

Most of the tolerance problems encountered in industry are 1-dimensional by nature. This would mean that the tolerance stack consists of linear vectors (e.g. the tolerance loop around a gearbox assembly or the stacking of high pressure valve components). There are also many instances where 2-dimensional tolerance analysis is required. The 2-dimensional stacks are comprised of vectors which are not necessarily co-linear. The sum of all the vectors, both the 1-D and the 2-D, constitute the 2-dimensional tolerance loop.
Figure 1. Bike crank assembly with tolerance loop.

Figure 2. Vector loop of the bike crank.

The advantage of designing parts on a CAD system is that all the part drawings may be called into one drawing and overlaid to create an assembly drawing. Tolerance loops may be taken directly from the assembly drawing and a.tolerance analysis performed. Instead of manually inputting data from a keyboard, the designer digitizes selected points from the assembly drawing. Numerical data is extracted directly from the CAD database and written to a data file. With this information a.tolerance analysis may be performed automatically. This link between design and analysis reduces the time spent by the designer in tedious work and thus frees him to concentrate on other design matters.
Brigham Young University has been the site for the development of the Computer Aided Tolerance Selection software named CATS.BYU. This ongoing program under the direction of Dr. Ken Chase began in 1984. New algorithms, improved tolerance accumulation models, application of powerful optimization tools, and the extension of tolerance analysis to include process selection are some of the results from this research.

A one-dimensional interface to CATS.BYU which uses the CAD system as a tolerance modeling tool has also been designed for three major CAD systems:

1) Hewlett-Packard ME-10
2) GE Calma/DDM
3) IBM/CADAM

The HP interface, HP-CATS, was written using the macro language of the ME10. HP-CATS allows the user to digitize points directly from an assembly drawing to create a tolerance loop. All selected data are then stored in a data file which is converted to a neutral file format and sent to CATS.BYU for analysis. HP-CATS is easy to use and is very effective in performing 1-D tolerance analysis.

A similar tool for modeling 2-D tolerance loops is needed. 2-D analysis is much more difficult, primarily because of the difficulty in creating the models and determining the tolerance sensitivities. An interactive, graphic modeling interface would encourage designers to do a more thorough job of tolerance analysis of complex geometries, and permit the application of powerful statistical optimization and other tools available in CATS.BYU.

Objective:
The object of this research is to 1) develop an interactive, graphic 2-D assembly tolerance model generator on the ME10 CAD system, 2) link it to CATS.BYU, and 3) determine the constraints and equations required for tolerance analysis. 4) Compute the tolerance sensitivities numerically by perturbing the loop parameters one at a time.
Methods to be followed:
1) Identify most common constraints necessary to constrain assembly models such as closure of a structure, perpendicularity, concentricity, tangent to a surface, etc.
2) Use Hewlett-Packard's ME10 procedural interface to generate an interactive graphics interface. The software package will enable the designer to construct a vector loop model of an assembly with the necessary constraints as an overlay on a CAD drawing of a complex assembly. An appropriate data file structure in which to store this information will also be designed.
3) By small perturbations of the graphics model parameters, the tolerance sensitivities will be determined numerically.
4) A formatted file will then be generated and passed to the CATS.BYU program for tolerance analysis and re-allocation.

Delimitation of the problem:
Determining the tolerance sensitivities from general algebraic expressions is being performed by another researcher. The application of the tolerance allocation algorithms to 3-D tolerance problems will be investigated as part of a separate research project.

References:


Chase, K. W., and Nielsen, D., "CATS.BYU USER'S GUIDE", Brigham Young University, 1987