FEATURE BASED AUTOMATED PART INSPECTION

by

Michael T. M. G. van Thiel

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

The University of Utah

August 1993

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a thesis submitted by

Michael T. M. G. van Thiel

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

Chair: Richard F. Riesenfeld

Samuel H. Drake

Thomas C. Henderson

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of The University of Utah:

I have read the thesis of Michael T. M. G. van Thiel in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to the Graduate School.

Date

Richard F. Riesenfeld
Chair, Supervisory Committee

Approved for the Major Department

Thomas C. Henderson
Chair/Dean

Approved for the Graduate Council

B. Gale Dick
Dean of The Graduate School

ABSTRACT

The significance of quality assurance in manufacturing has grown steadily in recent years. The use of Coordinate Measuring Machines (CMM) has become an affordable method to inspect manufactured parts. However, the software controlling these machines is often not very user friendly, creating inspection code is a difficult and error prone task and results of the inspection process can be hard to understand.

An experimental system to automate the inspection process of feature based manufactured parts is being developed. The system provides ICAM standard Dimensional Measuring Interface Specification (DMIS) code generation for a set of mechanical features. It includes an animated simulation of the inspection by parsing the generated code and also provides a intuitive graphical representation of measurement results by visualizing the nonconformities of the part. A separate server handles the execution of DMIS commands by establishing a connection with a Coordinate Measuring Machine.

CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

First I would like to thank Richard Riesenfeld for his support in getting me accepted into graduate school. Without his help this work would not have been possible. I am also very grateful to the rest of my committee, Samuel Drake and Thomas Henderson, for supporting my ideas.

Although the entire Alpha_1 staff and many of the Alpha_1 group provided much advise and resources, I would like to mention several separate. In always making time to answer any question I had, Russ Fish proved to be an invaluable resource. Mike Blum seemed to solve problems a couple of days before I encountered them, and, of course, was always eager so share the solutions with me. I also want to give him special thanks for being my number one thesis editor. Tim Mueller was always ready to provide the details of the display code. I would also like to thank David Johnson for reading and correcting early drafts of this thesis and Beth Cobb for always being extremely supportive of my work.

Finally I would like to thank Jim Rose for editing the video and teaching me how to operate the equipment and Jamie Painter for his help in getting the slides done in time for my defense.

This work was supported in part by DARPA N00014-91-J-123. All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

CHAPTER 1

INTRODUCTION

Inspection can be described as the checking or testing of products against established standards. The goal of the inspection process is to detect and eliminate the nonconforming, bad or defective items in a product and assure its quality. By using inspection devices like coordinate measuring machines (CMMs), vision systems or simple gauges, important information on the conformation of manufactured parts to design specifications can be collected.

Due to increased competition in the manufacturing industry, the significance of quality assurance has grown steadily in recent years and inspection of manufactured parts is therefore increasing in importance. Different manufacturing environments require different inspection systems to meet specific needs. Inspection can be incorporated as an inherent part of manufacturing, so that measurements of an item are made in different manufacturing stages and corrective actions can be taken immediately after processing[23, 26]. In the car industry, for instance, it is important to inspect items directly after each important manufacturing operation and so special devices are developed to inspect certain products fast and efficiently. For example, General Motors developed a machine-vision system to inspect holes in a truck's front end to verify correct punching[28]. This system is fast and works well for the job, but in an environment where different parts are designed and manufactured on a regular basis, there is a need for a more general inspection system.

The two basic forms of inspecting a part are *screening* and *sampling*. Screening can be viewed as global inspection. With advances in lasers, pattern recognition and other vision techniques, it is becoming an attractive practice. Screening, however, is still an expensive and complex process and is mostly used in special purpose areas, or as a support for the sampling method. Sampling gathers a set of point measurements of a part's surface and uses them to estimate the part's geometry. This derived geometry is then compared to and evaluated against a nominal part model to determine the conformance of the part. Inspection using CMMs is an example of the sampling method.

Influenced by availability, affordability and friendliness of smaller machines, the role of CMMs is growing rapidly in today's manufacturing community. The machines are very accurate and can be used for a broad spectrum of inspection problems. However, the software controlling these machines is often not very user friendly and results of the inspection process can be difficult to understand. Also, because an operator usually has no tools other than the physical part and its specifications, inspection code has been hand crafted. This is a difficult and error prone task. Ongoing research in developing CAD/CAM systems for design and manufacturing could help in creating a system for automating the inspection process and eliminating the problems associated with the manual operation and programming of CMMs.

A system that will automatically generate inspection code from a geometric model of a part could ensure the correctness of the code, improve efficiency and save time. A simulator that would accurately simulate the inspection process would provide a testing platform for experimental code and eliminate the risk of unnecessary programming errors. A method to perform the actual inspection without having to deal with an awkward user-interface to the CMM would make the operating task simpler. Also, reporting inspection results in a more intuitive form would make it easier to visualize the nonconformities of the inspected part. Results could be in the form of written reports or graphical representations of the results. If these ideas are integrated into a user-interface built on top of an already existing modeling and manufacturing environment, we would have a very useful tool for easy, accurate, fast and efficient inspection of different parts. Also, we would have a complete system encompasing design, manufacture, simulation, inspection and evaluation of inspection results. We present an experimental version of such a system next.

CHAPTER 2

INSPECTION SYSTEMS

The implementation of a general purpose system that will inspect and evaluate any manufactured part would be an almost impossible task. Therefore we start with a standardized manufacturing environment, the University of Utah's Alpha_1 [9] modeling system for Computer Aided Design and Computer Aided Engineering. This system has a feature based manufacturing environment upon which the inspection system is built. The inspection system implemented includes automated code generation in the DMIS[6] interface language (see section 3.2), an animated DMIS simulator (see section 6.2), a user friendly mechanism for inspecting parts (see section 5), and an evaluation portion to present measurement results (see section 6.4). These four tasks are controlled by a Motif based interface developed in the UNIX operating system.

2.1 Previous work

Until recently, CMMs were too large and expensive to find a home outside large manufacturing complexes. Operators would develop their own custom software and little research was done on developing systems that fully automated the manufacturing, inspecting and evaluating phases of production. In recent years, CMMs have become more affordable and available to small and medium sized job shops. However, until the introduction of the Dimensional Measuring Interface Specification (DMIS) [6], developing inspection systems was still a complex process due to the widely different types of CMM programming languages. DMIS provides a standard programming and interface language for CMMs and other dimensional measuring devices and makes the development of inspection systems less machine dependent.

Many systems are developed to assist users in measuring different products[12, 15, 16], and some of these could be called "complete" systems[21, 25]. Most of them, however, are too specialized to be used as a base for developing the inspection system described in this thesis. In the next two sections, some of the systems will be mentioned.

2.2 Specialized inspection tools

Integrated Surface Inspection Software[15] (ISIS) is designed to assist in the inspection and analysis of freeform surfaces and other nonprismatic geometric features. ISIS allows a user to generate surface information quickly from a minimal set of points. It is not a CAD/CAM package but an inspection data generation and analysis tool that provides the user with a graphical representation of information.

Another tool is *Tubular Shape Analyses Software* [16] (Tubic). Tubic is designed to ease the inspection and analysis of tubular shapes on a CMM. It offers a simple way to define a tube from a blueprint and it can generate a part program in DMIS to begin inspection.

Finally there is the *Airfoil Inspection Software*[12] (AFIS). AFIS is a package that will inspect any foil shape including turbine blades, stators and hydrofoils. Standard airfoil dimensions are completely implemented by AFIS, and it does not require a nominal definition to define an airfoil section. AFIS can be used to reverse engineer airfoils where nominal definitions are not available.

Because of the commercial value of these tools, little literature exists detailing their specific implementations.

2.3 Complete inspection systems

There are two commercial systems that could be looked upon as *complete* systems. The first one is *CimStation*^{TM}[25]—a graphic workstation for the interactive design, programming and simulation of automated manufacturing systems. CimStation provides users with a system for Computer Integrated Manufacturing applications, and the use of computer simulation to verify the manufacture, assembly and inspection of parts and products. CimStation can be used to simulate entire manufacturing processes.

The second one is *Valisys* [21]—a software package that integrates the design, manufacturing and inspection processes using a CAD database. Valisys translates specifications into part programs for inspection operations and executes inspection on CMMs.

No papers were found that discuss the specific implementations of these two systems in more detail.

2.4 Other work

M.J. Corrigal and R. Bell [10] describe a system that implements the inspection process planning and inspection machine planning. Inspection process planning analyzes a part drawing to determine what to measure and on what type of inspection device to do it. The machine planning performs the actual inspection on the chosen device. Although the idea behind this system is sound, it will be an immense task to make it work for all modeling environments and all available inspection devices.

Another system has been developed by Merat and Radack [22]. It has several similarities with the system described in this thesis and is called "Automatic Inspection Planning Within a Feature-Based CAD System". It is implemented in a variation of the Common LISP Object system and the concept Modeler [29]. Each mechanical feature (see section 4.2) used has a set of geometric dimensioning and tolerancing (GD&T) specifications from which inspection code is generated. The user specifies which tolerances apply to the design and a code generator will then create a code segment for every GD&T. Together these segments form the inspection program. Some of the difficulties mentioned by the authors are interacting features, probe orientations, part orientations, merging of code segments and path planning.

Using queries to the solid modeler, interacting features (see section 4.2.5) are identified and planning is done on a surface-by-surface basis. This means that when a code segment has to be generated for a set of interacting features, surface information will be used to determine how to measure the features.

The system currently allows only vertical probe orientations and does not mention the fact that different styli may be needed during an inspection process. During a typical inspection, multiple styli and probe orientations are used to perform the required measurements. Because the system does not utilize all possible probe orientations, multiple constraints on part orientations are made.

The system generates inspection points from the GD&Ts. If two or more GD&Ts are specified for one feature, the system will attempt to merge the code segments so no redundant measurements are made. Finally, the system tries to create an optimal path between all the point measurements to minimize inspection time.

Besides contact based inspection using Coordinate Measuring Machines, vision systems have been studied as a method for inspection. Several CAD-based computer vision systems for object recognition and inspection have been developed [2, 3, 4, 5, 18, 19, 20, 27].

CHAPTER 3

BACKGROUND

To fully understand the inspection approach of this thesis, a description of important background information will be given. We will talk about the modeling environment, the inspection language, the CMM, and the basic scenario for inspecting a part.

3.1 The modeling environment

The environment is the Alpha_1 modeling [9] system for Computer Aided Design and Computer Aided Engineering. It was developed at the University of Utah and runs on UNIX platforms. For mechanical design, a large set of mechanical features, including a number of hole types such as counterbores and tapped holes, several kinds of pockets, and slots are provided. The feature package includes linear and radial patterns which can be formed from other features. There are also procedures associated with each feature to generate Numerically Controlled (NC) code for the manufacturing of the modeled parts. The features describing the part are the main source of information for the generation of inspection code in the described system.

3.2 The inspection language

The Dimensional Measuring Interface Specification (DMIS) [6] was developed by Computer Aided Manufacturing International (CAM-I)[7] as a standard for the communication between inspection systems and design equipment and is used as the language for all the inspection programs generated. DMIS provides geometric calculations of circles, cones, cylinders, lines, planes and spheres from point measurements. These *primitive features* are used to describe the more complex mechanical features used in the Alpha_1 environment. DMIS also implemented the ANSI Y14.5M-1982[1] tolerancing standards for cone angles, angularity, circularity, sphericity, conicity, concentricity, coordinate location, cylindricity, diameter, flatness, parallelism, perpendicularity, position and straightness. Besides geometry-oriented definitions, there are also process-oriented commands in DMIS. They consist of motion, machine parameters and other commands that are used in the inspection process itself. The particular inspection device used in this thesis is a gantry CMM that uses the Coordinate Measuring Inspection Software (CMIS)[13] as its interface language. CMIS is a direct implementation of DMIS and no post- or pre-processing is required.

The following CMIS commands groups are used for the code generation (for specific details see [17]):

- Feature Definitions Commands

  The Feature Definitions Commands are used to provide nominal definitions of the primitive features. For example the FEAT/CIRCLE and FEAT/CONE commands define a circle and a cone.

- Parameter Set Commands

  The Parameter Set Commands are used to provide further definition for features and the actions performed on them. An example is FEDRAT for setting the velocities used in measurements and safe moves.

- Motion Sequence Commands

  The Motion Sequence Commands are used to perform nonmeasurement motion. For example the GOTO statement to move the probe to a specific location.

- Measurement Sequence Commands

  The Measurement Sequence Commands are used to initiate a measurement of a feature. Some examples are the MEAS, PTMEAS and ENDMES commands to start a measurement, measure a point and end the measurement.

- Sensor Commands

  The Sensor Commands are used to define, calibrate (see section 4.3.4) and invoke settings for the *sensor*. The described system uses a sensor consisting of a stylus mounted on a probe. For example, the SNSDEF/PROBE, CALIB, SNSLCT and SNSET commands which are used to respectively define a sensor, calibrate the sensor, select a predefined sensor and invoke settings like approach and retract vectors.

- Coordinate System Commands

  The Coordinate System Commands are used to create a *Part Coordinate System* (PCS). Since part features are defined in reference to datums, a PCS must be established reflecting these datums before features can be measured. Some examples are ROTATE and TRANSLATE to rotate and translate a coordinate system.

- File Commands

  The File Commands are used to define sensor, data and coordinate system files. For example, the FILNAM/DATA is used to define the data file.

- Output Commands

  The Output Commands are used to output results of feature measurements to a file or communication port. For example, the DISPLY and OUTPUT commands are used to set the output device and output actual data.

See Appendix A for an example of DMIS inspection code. DMIS makes certain requirements on the point patterns used for geometry calculations. When measuring a cone, DMIS requires the first three points to lie on a circle in a plane perpendicular to the cone's axis. The next three points have to lie in another perpendicular plane as far as possible from the first plane. The points in each circle must span at least 180°. Additional points can be taken anywhere.

For a cylinder, DMIS requires the first three points to lie on a circle in a plane perpendicular to the axis of the cylinder. DMIS requires a minimum of two additional points that can be taken anywhere.

When measuring a circle, DMIS requires a minimum of three points that lie on a circle or arc. Finally to measure a plane, DMIS requires a minimum of three points on the plane.

3.3 The coordinate measuring machine

The Coordinate Measuring Machine (see Figure 3.1) used consists of two major assemblies: the Machine Control Unit (MCU) and the Robot Assembly Unit (RAU). The MCU contains all of the electronics and software that control the measurement hardware, such as reading position scales and activating motor drives. The RAU is the mechanical part of the CMM which includes a frame, a table, motors, sensors and other mechanical assemblies.

A measurement probe (or sensor) attached to the mechanical arm on the CMM, controlled by a host computer, registers the actual measurements. The probe comes

Figure 3.1. Coordinate measuring machine

equipped with different sized ruby ball styli that have to be attached to the probe (see Figure 3.2). These styli are used to touch the object. The probe registers the touch, determines the machine coordinates at the time of contact, and reports these back to the host computer. Depending on the shape of the features, different styli may be selected and attached to the probe during the inspection process. The changing of the styli can be done manually or by using an automatic styli changer.

To get accurate measurements, it is necessary to calibrate every stylus before using it for any measurements. This calibration is achieved by measuring a reference object of known dimensions. This information is then used for the calculation of offsets which are used to correct subsequent measurements.

All measurements registered by the probe are with respect to the machine coordinate system and have to be translated to a coordinate system representing the part. By making a set of manual measurements of the part a Part Coordinate System (PCS) is calculated. This coordinate system matches the coordinate system of the geometric model so that measurement data can be compared against the nominal data.

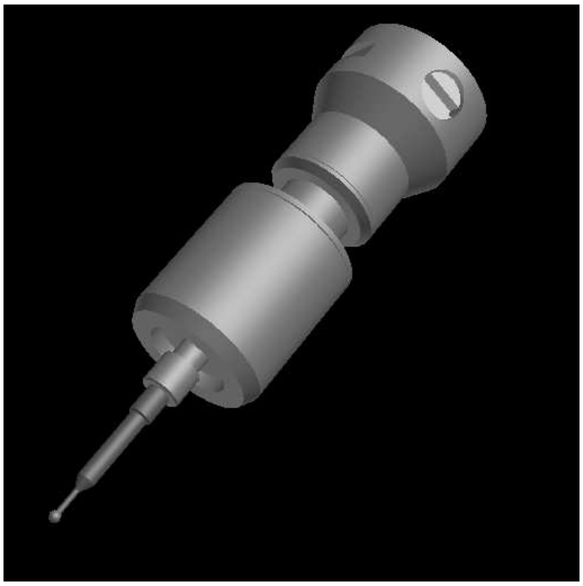


Figure 3.2. Probe and stylus

3.4 Inspection scenario

The basic scenario for inspecting a part using a CMM is as follows.

- Fixturing

  Fixture the part on the work area of the CMM.

- Calibration

  Calibrate all the styli that are going to be used during the inspection of the part.

- Establishing part coordinate system

  Determine the part coordinate system by acquiring a set of initial measurements.

- Creating inspection program

  Write an inspection program that will make the correct measurements.

- Inspecting

  Execute the inspection program on a CMM.

- Evaluation

  Evaluate the results of the measurements.

CHAPTER 4

THE API SYSTEM

4.1 General description

In this section, a general description of the implemented system is presented. The system is written in a Lisp variant, Rlisp[8], C++ and Motif. It is referred to as the Automated Part Inspection (API) system and consists of several parts.

A FANAMATION COMERO 404024 [11] Coordinate Measuring Machine (CMM) is used to perform the actual inspection. The CMM is controlled by the CMIS user interface [14] and takes Coordinate Measuring Inspection Software (CMIS)[13] as input (see section 3.2).

A *code generator* creates the DMIS inspection code (see section 4.3) and the necessary administration files (see section 4.3.4) from a list of mechanical features (see section 4.2) describing the part. This list is provided by the user.

A *simulator* (see section 6.2) parses the generated code and simulates the inspection process by animating a stylus along the inspection path.

The execution of DMIS code on the CMM is managed by the *CMIS server*. This server acts as a communication medium between the API system and the CMIS user interface (see Chapter 5). Lines of DMIS code are sent from the API system through the server to the CMIS user interface. Measurement results are sent the other way, stored in an appropriate format and used during the evaluation phase (see section 6.4).

The evaluation of inspection results is performed by the *evaluator* (see section 6.4) which presents a graphical representation of inspection results to the user.

A graphical user interface provides a user friendly way of dealing with display of parts, code generation, simulation, inspection and presentation of results. The interface manages the total information flow between the different modules (see Figure 4.1) and makes the system work fast and efficiently.

4.2 Code generation

The input for the code generator consists of a list of *features* describing the part. Features are structures that hold information on a particular detail of a part. In addition to geometric information, tolerance requirements for the feature and interactions (see section 4.2.5) with other features can be stored. The output from

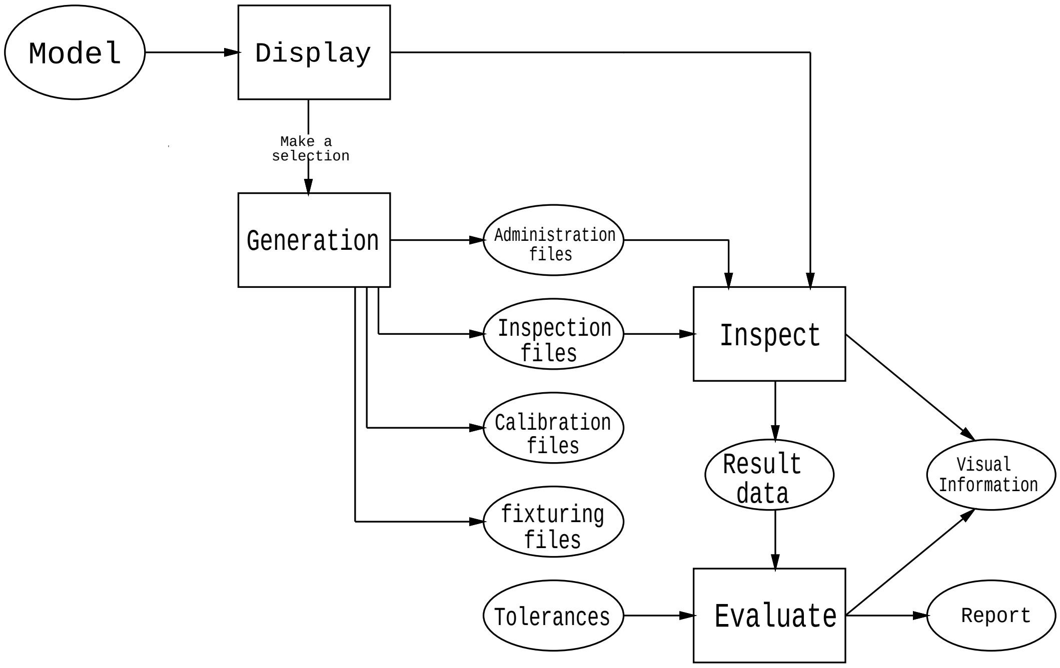


Figure 4.1. Flow diagram describing the user interface.

the code generator are the generated DMIS inspection files.

4.2.1 Available features

The following mechanical features can be inspected using the API system: holes, counterbores, countersinks, counterdrills, slots, rectangular pockets, pockets from 4 points, profile pockets and profile bosses.

4.2.2 Collision avoidance

To ensure no unexpected collisions occur during nonmeasurement moves of the probe, a safe area is defined in which all moves, styli changes and probe orientation (see section 4.2.3) changes are made. This safe area is defined by the area outside the bounding box of the inspected part.

4.2.3 Probe orientations

The probe orientation is defined by the *tilt* and *roll*. The tilt is the elevation angle of the probe head and has a value from $0°$ and $105°$ in increments of $7.5°$, the smallest supported by the CMM on which it is implemented. The roll is the rotational angle of the probe head and has a value from $-180.0°$ to $180.0°$ in increments of $7.5°$, again limited by the specific hardware. All probe orientations can be used to make the required measurements.

4.2.4 Tolerances

Although DMIS provides several tolerance related commands they are not currently used by the API system. Tolerances are specified by the user and used during the evaluation phase (see section 6.4) of the system to determine the conformance of the part. Four tolerances defined by the ANSI standard for Dimensioning and Tolerances[1] are implemented in the API system.

> A positional tolerance is part of the group of tolerances of location and defines a zone within which the center, axis or center plane of a feature may vary from true position.[1]

The positional tolerance is specified by a radius of a circle indicating the zone in which the center, axis or center plane must lie. The center of the circle is the true position of the considered element.

> A diameter tolerance is part of the group of form tolerances and specifies a zone in which the considered value must be contained.[1]

The diameter tolerance is specified by a value indicating the maximum deflection from the nominal diameter.

> An angularity tolerance is part of the group of orientation tolerances and indicates the condition of a surface or axis at a specific angle (other than 90°) from a datum plane or axis. An angularity tolerance specifies a tolerance zone defined by two parallel planes at the specific basic angle from a datum plane, or axis, within which the surface or axis of the considered feature must lie.[1]

The angularity tolerance is specified by indicating the distance between the two parallel lines.

> A perpendicularity tolerance is part of the group of orientation tolerances and indicates the condition of a surface, median plane, or axis at a right angle to a datum plane or axis. A perpendicularity tolerance specifies a cylindrical tolerance zone perpendicular to a datum plane within which the axis of the considered feature must lie.[1]

The perpendicularity tolerance is specified by a radius of a circle indicating the zone in which the axis must lie. The center of the circle is the true position of the considered element.

4.2.5 Interacting features

When inspecting complex parts, problems can occur. Features might *interact*; that is, parts of the features might not exist due to intersections with other features (see Figure 4.2). If there is no information on what part of the feature can be used for measurements, problems can occur when generated code is executed. The API system requires the user to specify the relationships between features in the initial feature list. If there are no specifications on interactions between features, the API system assumes that the features are complete. Currently there are no provisions to specify feature interactions in the initial feature list.

4.3 The generation process

The generating process can be divided into the following four phases.

4.3.1 The decomposition phase

The DMIS inspection code only provides feature definitions for a set of *primitive features*. Therefore, all the mechanical features available in the system have an associated method that decomposes the feature into a set of *inspection steps*. Each inspection step of the API is associated with a primitive feature and contains the following information:

- Primitive feature

  Every inspection step includes a primitive feature. To describe what part of the primitive feature is available for measurements (see section 4.2.5) it is possible to specify a start and end angle (see Figure 4.2). The following primitive features are available:

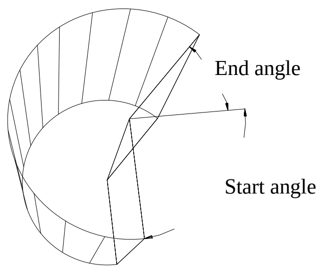


Figure 4.2. Start and end angle

1. Cone

   A *cone* is described by a center point, the offset to the bottom, the diameter at the top, the diameter at the bottom, the included angle and the orientation (see Figure 4.3 a). A cone could be used to describe a chamfer (see Figure 4.4 a).

2. Cylinder

   A *cylinder* is described by a center point, the offset to the middle, the depth, the diameter, the orientation, and a Boolean indicating if the cylinder extends through the material (see Figure 4.3 b). A cylinder could be used to describe a hole (see Figure 4.4 b).

3. Circle

   A *circle* is described by a center point, the offset to the center, the diameter, a Boolean indicating whether the outside or the inside of the circle should be measured and the orientation (see Figure 4.3 c). A circle could be used to describe an arc in a profile curve (see Figure 4.4 f).

4. Rectangular plane

   A *rectangular plane* is described by the four points defining the rectangle, a save point from where to start the measurement, the normal of the plane and the orientation of the original feature the plane is contained in (see

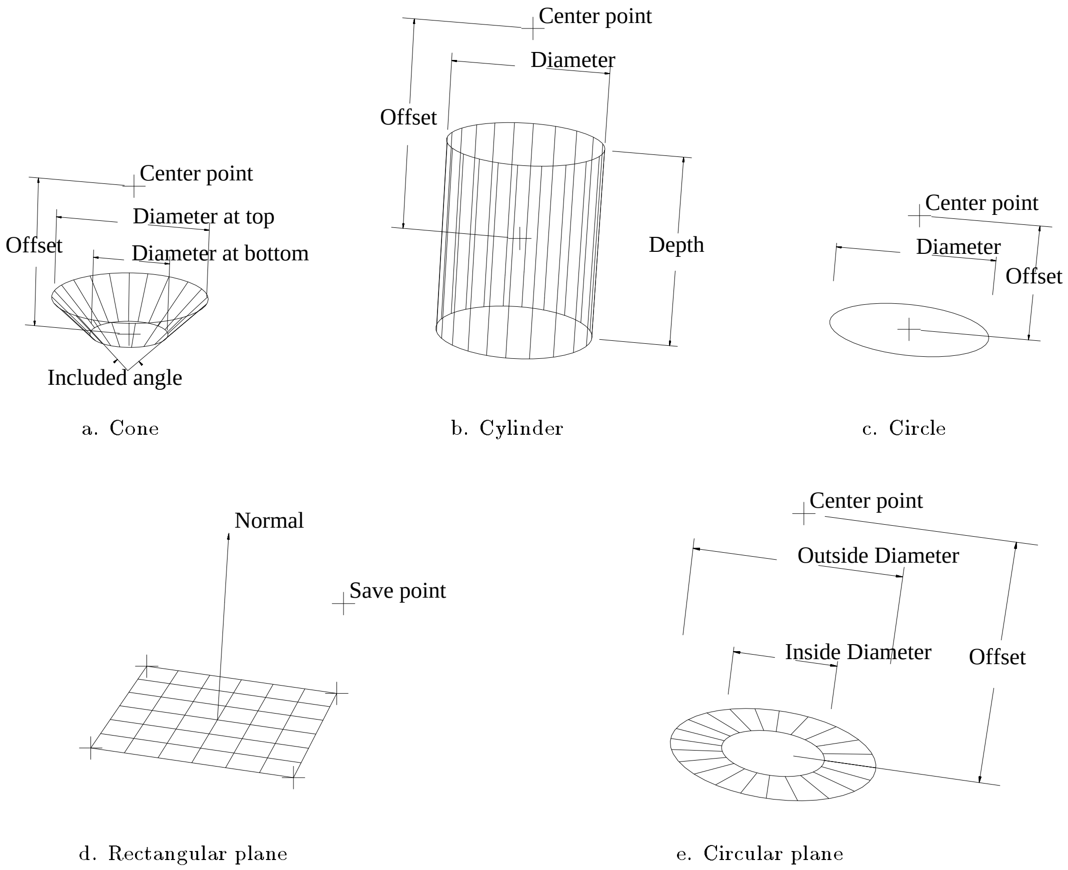


Figure 4.3. Primitive features

Figure 4.3 d). A rectangular plane could be used to describe the sides in a slot (see Figure 4.4 e).

5. Circular plane

   A *circular plane* is described by a center point, the offset to the plane, the outside diameter, the inside diameter and the orientation (see Figure 4.3 e). A circular plane could be used to describe a step in a counterbore (see Figure 4.4 b).

- Name

Every inspection step includes a unique name, consisting of a one character code describing the original feature type, a number to identify features of the

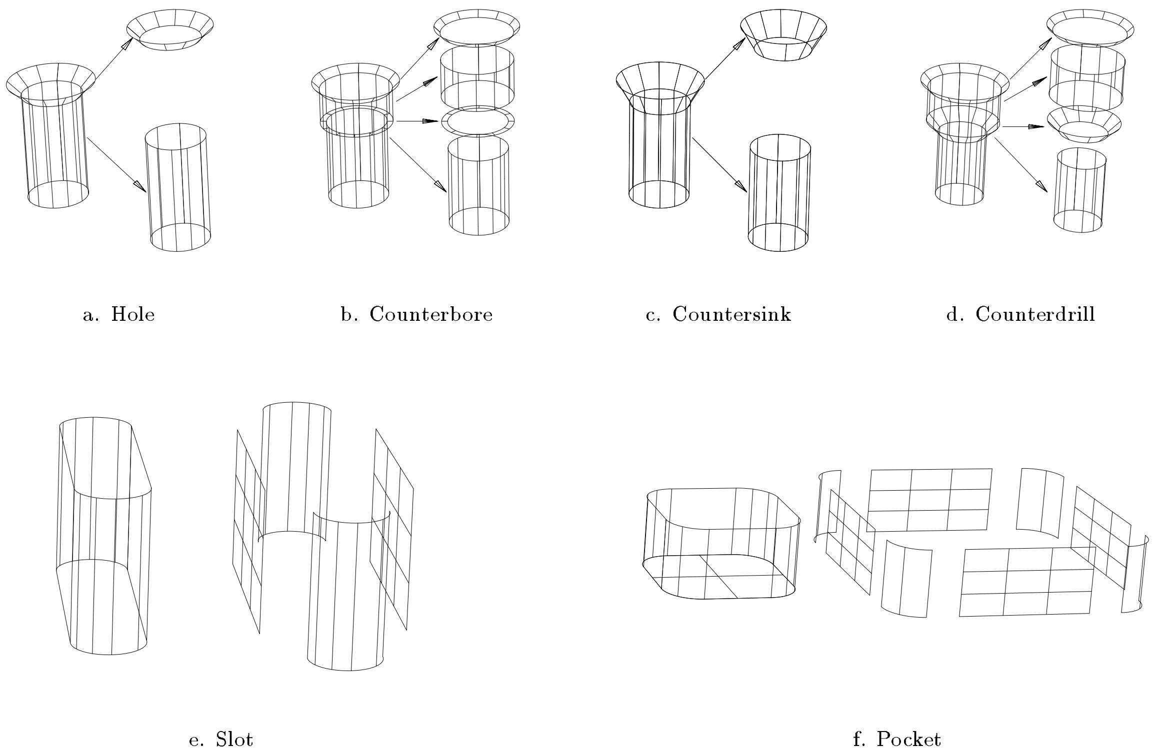


Figure 4.4. Decomposition of features.

same type, a one character code describing the primitive feature and a number identifying primitive features of the same type.

- Styli set

  Every inspection step includes a *styli set* (see section 4.3.2) that can be used to perform the required measurements.

- Code fragment

  Every inspection step includes a DMIS *code fragment* to perform the required measurements.

- Start position

  Every inspection step includes a safe *start position* from where to start the measurements described in the code fragment. This positioning is not part of the code fragment because it might not be performed if the stylus is already in a safe position.

- End position

  Every inspection step includes a safe *end position* from where to continue to the next inspection step. This positioning is not part of the code fragment because it might not be performed if the next inspection step is at the same location as the current one.

Every mechanical features is decomposed into different inspection steps. A hole is decomposed into two inspection steps. A cone representing the chamfer and a cylinder representing the actual hole (see Figure 4.4 a).

A counterbore is decomposed into four inspection steps—a cone representing the chamfer, a cylinder representing the bore, a circular plane representing the divider between the hole and bore and a cylinder representing the hole (see Figure 4.4 b).

A countersink is decomposed into two inspection steps—a cone representing the sink and a cylinder representing the hole (see Figure 4.4 c).

A counterdrill is decomposed into four inspection steps—a cone representing the chamfer, a cylinder representing the bore, a cone representing the counterdrill and a cylinder representing the hole (see Figure 4.4 d).

A slot is decomposed into four inspection steps—two cylinders representing the two semicylinders and two rectangular planes representing the two sides (see Figure 4.4 e).

A rectangular pocket is decomposed into eight inspection steps—four cylinders representing the four corners and four rectangular planes representing the four sides (see Figure 4.4 f). A pocket from four points is decomposed in the same manner as a rectangular pocket.

A profilepocket is decomposed into as many inspection steps as there are arcs and lines in the profile. This means that the pocket is first described in terms of arcs and lines. Then an inspection step is created for every cylinder represented by the arc and for every plane represented by the line (see Figure 4.4 f). A profileboss is decomposed in the same manner as a profilepocket.

The start and end position for all inspection steps is a point on the axis of the primitive feature outside the bounding box of the part (see section 4.2.2).

After the decomposition, the original list of features describing the part does not exist anymore. Therefore, an administration file is created describing the original mechanical features in terms of its primitives. This file is used during the evaluation phase to piece the features back together and return meaningful results. Mechanical features that cannot be described in terms of the DMIS supported primitive features cannot be inspected by the system. These could include freeform surfaces and other complex shapes. The result of the decomposition phase is a list of inspection steps describing the part.

4.3.2 The selection phase

For every primitive feature, the API system must select an appropriate stylus from a library of ruby ball styli, extensions and adaptors. *Ruby ball styli* are used for all probing applications in the API system, *extensions* provide extra probing penetration for the styli and *adaptors* allow you to use all styli on all probes.

To maximize accuracy it is recommended[24] to keep the styli short and stiff. The more a stylus bends or deflects the lower the accuracy. It is also recommended to keep the stylus ball as large as possible. With large balls, surface finish has less effect, and more flexibility is gained due to the ball/stem clearance (see Figure 4.5).

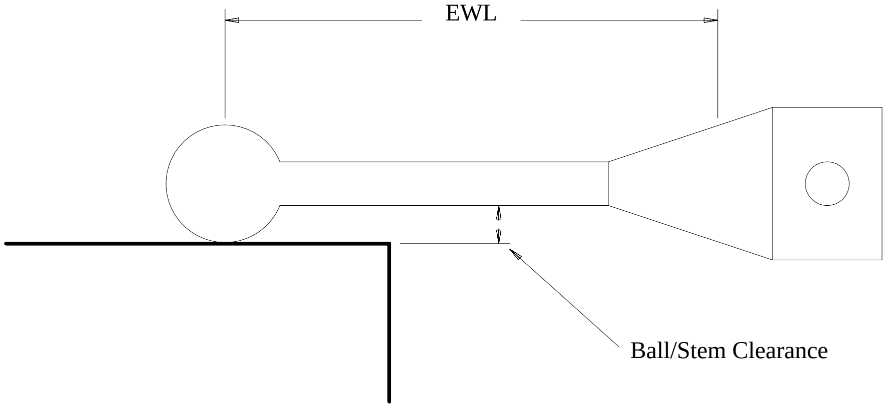


Figure 4.5. Ruby ball stylus

To select a suitable stylus the *effective working length* (EWL) and the ball diameter are taken into account. The EWL of a stylus (see Figure 4.5) determines how deep the stylus can safely penetrate a feature and still get a measurement without fouling the stem on the component. Generally the larger the ball, the greater the EWL. If the EWL of a stylus is the same as the actual length, a stylus extension can be used to lengthen the EWL. The ball diameter of a stylus is determined from the smallest hole to be measured.

A styli set is established for every inspection step. This set contains all the extension-styli combinations that can be used to perform the required measurements. The set is created by eliminating all unsuitable styli. The eliminations are accomplished by applying length and diameter filters. The *diameter filter* removes all styli that have a diameter larger than the one specified in the filter. The *length filter* removes all styli/extensions with an effective working length (EWL)/length smaller than the one specified in the filter. If the length of a stylus is equal to its EWL, the stylus is not removed from the set, because an extension could be used to lengthen the EWL. The following filters are applied to get the styli set for the different primitive features:

- Cones

  The value of the diameter filter is equal to the minimum of:

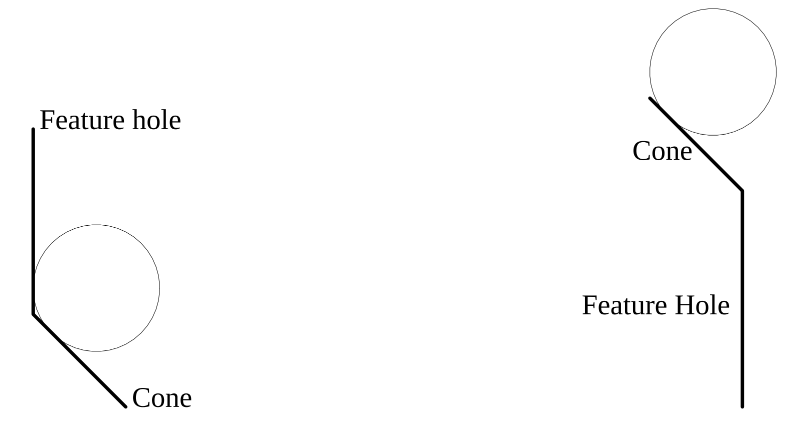
  1. The diameter of the circle tangent to the two lines representing the feature hole and the cone (see Figure 4.6 a) minus a default moving clearance. If the cone is not located inside the feature (see Figure 4.6 b) a default diameter is used.
  2. The diameter at the bottom of the cone (see Figure 4.3 a) minus a default moving clearance.

  The value of the length filter is equal to the offset from the centerpoint to the bottom of the cone (see Figure 4.3 a).

- Cylinders

  The value of the diameter filter is equal to the diameter of the cylinder minus a default moving clearance.

  The value of the length filter is equal to the depth of the deepest point measured in the cylinder.



A. Inside feature B. Outside feature

Figure 4.6. Selecting a probe for a cone.

- Circles

  The value of the diameter filter is equal to the diameter of the circle minus a default moving clearance.

  The value of the length filter is equal to the offset from the centerpoint to the center of the circle (see Figure 4.3 c).

- Circular planes

  The value of the diameter filter is equal to the difference of the outside and inside diameter (see Figure 4.3 e) minus a default moving clearance.

  The value of the length filter is equal to the offset from the centerpoint to the the plane.

- Rectangular planes

  A rectangular plane does not have an associated method for selecting styli. It uses styli selected by other primitive features.

A length filter is only applied to those extensions that are used for styli that have an EWL which needs to be lengthened. The value of this length filter is equal to required EWL for that stylus minus the actual EWL of the stylus. After the selection phase, each inspection step includes a set of styli.

4.3.3 The optimization phase

During the optimization phase the inspection steps are regrouped into *inspection segments*. An inspection segment is a group of inspection steps with common styli in their styli sets. By creating the inspection segments, a minimum number of styli changes during the inspection process is ensured. If an inspection segment has multiple common styli in the styli sets, the *best stylus* is selected. The best stylus is the stylus that provides the greatest accuracy (see section 4.3.2). After the optimization phase, each inspection segment has only one associated stylus, thus ensuring the minimal amount of styli changes.

4.3.4 The generation phase

During the generation phase, the API system creates two *administrative files*. One of the files (see Appendix C) describes the original features in terms of its primitive features (see section 4.3.1). The other file contains information on the styli to be used during the inspection (see section 4.3.2).

Each time the CMM is started, all styli used during inspection need to be calibrated to insure optimal accuracy. Calibration is needed to establish the exact ball diameter and the relationship between the center of the ruby ball and the probe head. By measuring a calibration ball of known dimensions and comparing the results against the nominal dimensions, the styli definition is adjusted by the CMIS software to minimize the effect of measurement errors. For every stylus selected during the optimization phase, a *calibration file* (see Appendix B) is generated to calibrate all needed orientations (see section 4.2.3) of the stylus. Before inspecting the part, all calibration files need to be executed.

Because a part can be located anywhere on the work area of the CMM it is necessary to create a Part Coordinate System (PCS) which is used as a datum by the inspection programs. To create a PCS, the API system creates a *fixturing file* (see Appendix D) to determine the position of the part. When executing this file the user is asked to manually make a set of measurements on the part. The PCS is then calculated and saved for use during the actual inspection. A simular file is generated for determining the PCS of the calibration ball.

For every inspection step (see section 4.3.1) a fragment of DMIS inspection code is generated using the DMIS command groups (see section 3.2). Together these fragments form the complete inspection program (see Appendix E) which is stored in an *inspection file.*

The command groups can be separated into two main command classes. The first class is the motion class which includes the motion and measurement sequence groups. The code generated using these commands is used to position the probe and make measurements. The other class is the nonmotion class which includes all of the other groups. The code generated using these commands is used to define features, set velocities, define and select sensors, invoke sensor settings and control output to files or communication ports. The following outline is used for the code fragments of all the primitive features:

- Select a sensor.
- Invoke the sensor settings.
- Set the positioning velocity.
- If necessary, go to the safe start position (see section 4.2.2 and 4.3.1).
- Perform an initial positioning.
- Set the measurement velocity.
- Define the feature.
- Start the measurement.
- Measure a point pattern to calculate the geometry of the feature (see below).
- End the measurement.
- Output the data.
- Perform a final positioning.
- If necessary, go to the safe end position. (see section 4.2.2 and 4.3.1).

Although the pattern of nonmotion commands is almost identical for all features, the motion commands are very different. For every primitive feature, a different pattern of points is used to make measurements. The pattern for a cone and cylinder consists of two parallel circles on the feature (see Figure 4.7 a,b). A

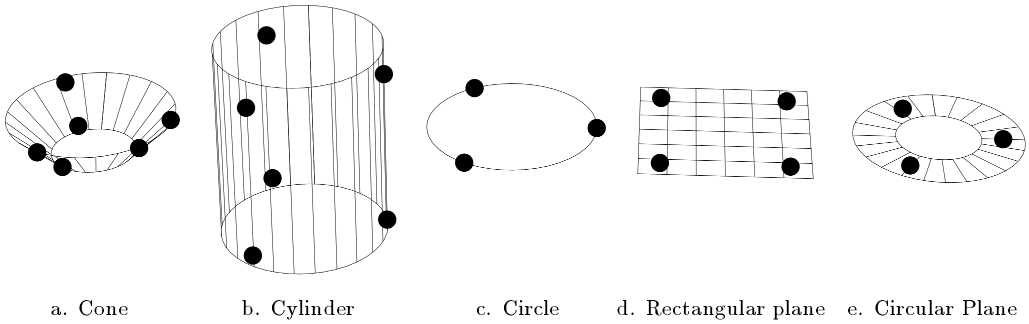


Figure 4.7. Point selection for primitive features.

circle is determined by measuring three points on the circle (see Figure 4.7 c). A rectangular plane by four points (see Figure 4.7 d) on the plane and finally a circular plane by three points (see Figure 4.7 e).

CHAPTER 5

CODE EXECUTION

The execution of code is performed by the CMIS[13] user interface which controls the CMM. The CMIS interface executes one line of DMIS code at a time and waits for a reply from the CMM before executing the next one.

5.1 Communication

The communication with the CMIS interface is controlled by the *CMIS server* which is part of the API system. The server establishes a connection and communicates with the interface through an RS232 port. Any client program can connect to the server and send requests. A reply is given to every request.

When sending the request *BUFFER n*, n being a number greater than zero, the server creates a buffer to store n lines of DMIS code. The reply to this request is *Buffer set*. By using a buffer, the client can send a DMIS code fragment to the CMIS server without having to wait for execution of the code. The CMIS server executes the code, takes care of errors and stores the replies from the CMM in a queue. The client can continue performing other tasks while the code is being executed, and replies can be requested at a later time. If a buffer already exists upon receiving the buffer request, the old buffer is emptied and the code stored is lost.

When sending a line of DMIS code as a request, three replies are possible. If a buffer exists, the line is stored in the buffer and the reply *Added to buffer* is sent to the client. If the line sent is the last line (number n), the CMIS server will send the reply *Emptying buffer* and start executing the buffered code. If no buffer exists, the line is executed by the CMIS server and the reply from the CMM is sent to the client. In this case, the client has to wait for the CMM to finish execution of the line.

By sending the request *REPLY*, the CMIS server pops a reply from the queue and sends it to the client. If the reply queue is empty when a reply request is received, the server will reply with an *ENDREPLY* message. If a reply queue exists upon creating a new buffer, the queue is emptied and the replies stored are lost.

Because the CMIS server is a separate program, it is possible to specify an inspection file name on the command line. In this case, the server does not accept connections with any clients but executes the DMIS code in the inspection file and sends the CMM replies to standard output.

5.2 Error detection

When the CMM replies with an error after attempting to execute a line of DMIS code, the CMIS server discontinues execution of the buffered code and informs the client of the error upon receiving the next *REPLY* request. After informing the client, the server will break the connection with the CMIS interface, initialize all variables and try to establish a new connection. The client has to deal with the error in an appropriate way to minimize its impact. For example, the API system will inform the user of the error and asks to restart the inspection.

CHAPTER 6

USER INTERFACE

The user interface makes the use of the API system more convenient by providing a user-friendly way of operating the system. The user interface consist of several parts.

6.1 Code generation

The interface provides an area in which a mechanical part can be displayed. By selecting a set of features in this area, the user can create a feature list (see section 4.2) to be used for generating inspection code.

6.2 Simulation

After generating a DMIS inspection file, it is possible to simulate the inspection process. Along with the commands in the motion and measurement sequence groups, sensor selections and velocity settings are simulated by the API system.

In order to perform a simulation, the inspection file is parsed and an *inspection object* is created. An inspection object contains all the information needed to simulate or perform the execution of a single inspection file. An inspection object consists of several items.

The first item is the *initial code fragment* which is used when performing the actual inspection (see section 6.3). It is not needed for the simulation and consists of commands to initialize the inspection process, such as file name definitions and parameter settings. The initial code fragment is executed a single time at the beginning of every part inspection.

Another item is the *final code fragment* which is also used when performing the actual inspection. It consists of commands to finalize the part inspection, such as selecting a sensor orientation and moving to a default position. The final code fragment is executed after completing a part inspection.

Finally, for every inspection segment (see section 4.3.3) a *simulation object* is created and added to a list stored in the inspection object. A simulation object contains all the information needed to simulate or perform the inspection of a single inspection segment.

The simulation object contains several items, such as a *start time* at which the first motion in the inspection segment occurs and an *end time* for the last motion.

It also contains a list of *action objects.* Action objects contain the actions performed at specific time steps during the simulation. Besides the *time* at which to perform the action, an action object contains any combination of the following items:

- The *position* to where the stylus will move during this action.
- The *point* that is being measured during this action.
- The *orientation* change of the sensor during this action.
- The *message* that needs to be displayed at this time.
- The *polyline* representing the path of the stylus during this action.

Another item in the simulation object is a list of *inspection objects.* An inspection object contains a *code fragment* consisting of DMIS commands that must be executed together. These fragments include feature measurements, changing of sensors, orientation of sensors, positioning of sensors and output of data. The larger code fragment associated with the inspection segment is decomposed into smaller code fragments which are stored in the inspection objects. An inspection object also contains the *time* at which to execute the code fragment and a list of *replies* for all the commands in the code fragment.

Finally the simulation object contains a *motion curve* and a *stylus.* A motion curve is a function that specifies the values of several transformations over time and is used by Alpha_1 to animate models. By using the time, position and orientation slots in the action objects, translation and rotation transformations over time are specified and stored in the motion curve. The geometry of the stylus is used as the model that is animated by the motion curve.

During the simulation, the user can control several options (see Figure 6.1), including selecting an inspection segment, displaying the part, displaying the inspection path, displaying the stylus, obtaining information on the stylus (see Figure 6.2), toggling the sound and starting the actual simulation. The actual simulation is controlled by the preview widget (see Figure 6.3). This widget consists of a set of VCR control buttons indicating stop, play forward, play backward, fast forward and fast backward. The preview widget executes the action objects at the correct time. During the simulation, the code fragments in the inspection objects are not executed.

6.3 Inspection

Before starting the inspection process, a connection with the CMIS server is automatically established. The server executes the DMIS commands (see Chapter 5) and stores replies from the CMM. The following options are available in the user interface and must be performed in order to inspect a part:

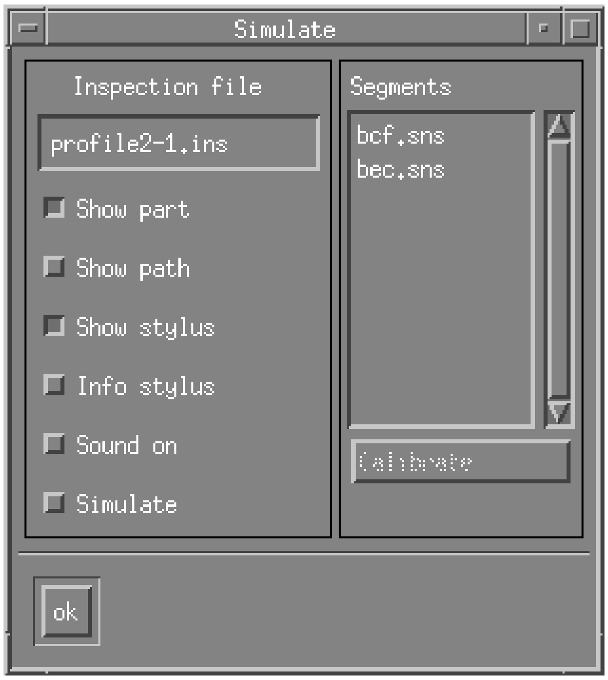


Figure 6.1. Options during simulation.

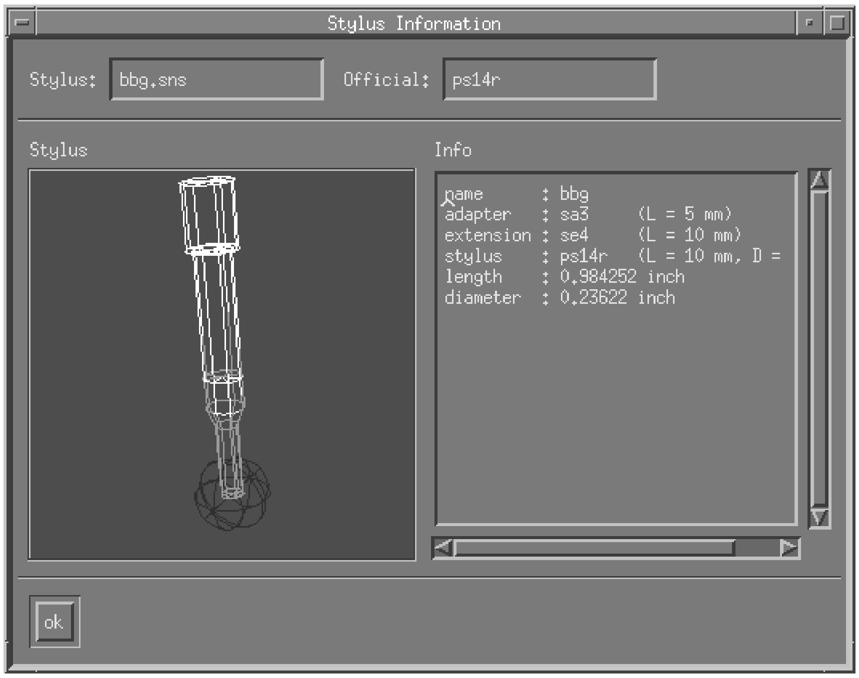


Figure 6.2. Stylus information window.

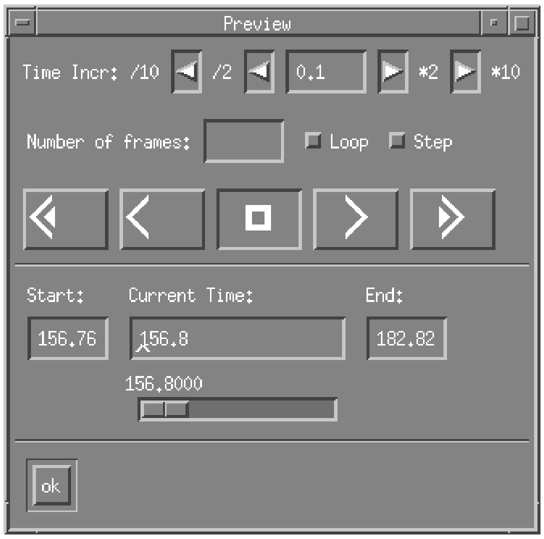


Figure 6.3. Preview widget.

- Locate Calibration Ball

  Before calibrating the styli needed for the inspection, the Part Coordinate System (see section 4.3.4) of the calibration ball must be calculated. This is accomplished by sending the pregenerated fixturing file to the CMIS server. Because the PCS is used by the CMIS interface, it is stored locally and no reply requests are sent to the CMIS server.

- Calibrate Styli

  Before inspecting the part all styli needed during the inspection must be calibrated (see section 4.3.4) by sending the different calibration files (see section 4.3.4) to the CMIS server. Because the calibration results are used by the CMIS interface they are stored locally and no reply requests are sent to the CMIS server.

- Locate Part

  Before inspecting the part, the PCS of the part must be calculated. This is accomplished by sending the pregenerated fixturing file to the CMIS server.

Because the PCS is used by the CMIS interface and the code generator, a reply request is send to the CMIS server to obtain the correct orientation of the part.

- Inspect Part

  The inspection object described in section 6.2 is also used for the actual inspection. During the inspection the user, has the same control as in the simulation. The only difference is the fact that the code fragments stored in the inspection objects are executed at the correct time. Before sending a code fragment to the server, the API system requests all replies from previous code fragments and stores them in a *result file*. This file contains all measurement results for the inspection. By sending the entire code fragment to the CMIS server, the API system does not have to wait for the execution of the code and can continue with the simulation.

6.4 Evaluation

The evaluation section of the API system uses data stored in the result file and information from the administration file (see section 4.3.4) to present the user a graphical and written representation of the inspection results.

The written report gives an overview of all the inspected features. It reports the nominal and actual dimensions of the part and also indicates the difference between these dimensions. According to the tolerances specified by the user (see section 4.2.4) it indicates whether the features are within or out of tolerance. Tolerances are globally specified in the tolerance information area of the user interface (see Figure 6.4).

By selecting a feature detailed information is requested, the feature is displayed in a different area (see Figure 6.5) and numerical information is presented to specify

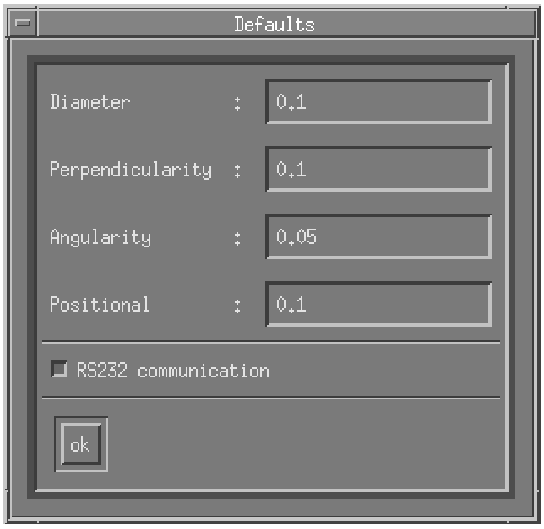


Figure 6.4. Tolerance information.

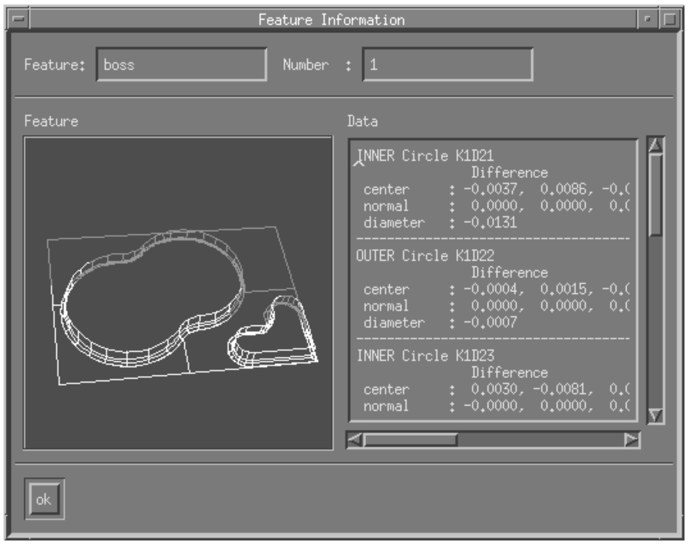


Figure 6.5. Evaluation results.

which part of the feature is out of tolerance. The graphical representation of the inspection results uses a color coding to indicate which features are out of tolerance. Features within tolerance are green and features out of tolerance are red.

CHAPTER 7

FUTURE WORK

Although the current version of the API system is complete and can be used for a variety of inspection problems, much work can be done to improve the system.

Currently only inspection steps (see section 4.3.1) associated with the five primitive features exist. These inspection steps are used to measure a primitive feature and compare the actual dimensions against the nominal dimensions after the measurements. The user is always forced to inspect a feature in a predefined manner. To provide for multiple methods of decomposition, more inspection steps should be implemented. Some examples are inspection steps associated with the possible ANSI tolerances (see section 4.2.4). Also, it should be possible for the user to influence the decomposition of features. This could be done by storing specific tolerance requirements with every feature and using this information to perform the decomposition. Another option would allow the user to set default parameters for decomposition, such as the number of points measured on a circle.

Only four global tolerances (see section 4.2.4) can currently be specified in the system, they apply to all measurements and are not feature specific. More tolerances should be available, and, in particular it ougth to be possible to specify one for each separate feature. The API system should also use the tolerance related commands in DMIS to implement the inspection steps for these tolerances.

Only the features mentioned in section 4.2.1 can be inspected. All features defined in the Alpha_1 system should be supported by the API system.

One probe orientation is chosen for every primitive feature. The tilt and roll of the probe are specified in increments of 7.5° (see section 4.2.3) and features oriented using the same orientations can be inspected without a problem. Features that are not oriented in the same manner might need more than one probe orientation to perform all necessary measurements. An algorithm is needed to select the probe orientations necessary to inspect a primitive feature. Also the outline of the code fragment (see section 4.3.4) for the primitive features will need to be adjusted to allow for multiple probe orientations within one measurement.

The only safety feature is the collision avoidance provided by the bounding box (see section 4.2.2). The user has real time control over the inspection process and can cause unexpected collisions. For instance, if the user selects a different inspection segment (see section 4.3.3) while the inspection is paused (stop button pressed) and the stylus is located inside a hole, a collision will occur because the probe orientation will change while the stylus is inside the hole. The system should provide a method to ensure that when a request for a sensor change is made, a move to a safe position will always be made.

Currently, the initial feature list (see section 4.2) holds no information on interaction between features (see section 4.2.5). Therefore the API system always assumes that the entire feature is available for measurement. A method should be developed to determine what part of a feature is available for measurement, and this information should then be provided to the API system to ensure correct code generation.

CHAPTER 8

SUMMARY

This thesis describes an approach for an easy, accurate, fast and efficient feature-based inspection system for mechanical parts. The code generator provides a fast and accurate method of creating correct DMIS inspection code from geometric data. It eliminates the need for manual programming which can be an error prone and time consuming task. The simulation of the inspection gives the user better feedback during the execution of programs and can provide a testing ground for experimental code. By establishing direct communication with the CMIS interface, user input during inspection is reduced to a minimum. Results are available after every measurement, and errors are dealt with in an appropriate way. The evaluation phase of the system provides an intuitive graphic representation of the inspection results and generates written reports on the inspected parts. Finally, the user interface pieces the different segments together and makes the system efficient, and easy to use. The API system meets the needs of many users, especially, in an environment where new parts are manufactured on a regular basis.

APPENDIX A

AN EXAMPLE

In appendices B,C,D and E we will show the different files generated when using the system for a part including a counterbore (see Figure A.1) of the following dimensions:

- Location = 0,0,0
- Bore diameter = 0.6
- Bore Depth = 0.35
- Hole diameter = 0.5
- Hole depth = 1
- Chamfer= 0.05

The corresponding Alpha_1 command is:

```
CBore1 := counterBore( origin, 0.6, 0.35, 0.5, 1, 0.05, T );
```

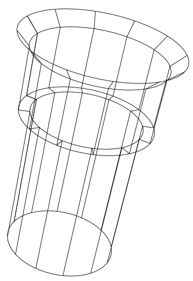


Figure A.1. A counterbore.

APPENDIX B

CALIBRATION FILE

The calibration file for the part in Appendix A will look as follows:

```
DMISMN/'bcf.cal 3.22'
FILNAM/'ON'
FILNAM/COORD,'setcal.crd',INPUT
WKPLAN/XYPLAN
PRCOMP/ON
FINPOS/OFF
UNITS/INCH,ANGDEC
MODE/PROG,MAN
FEDRAT/POSVEL,MPM,9.2
GOTO/15,15,20
DISPLY/STOR,DMIS
SAVE/D(MCS)
MODE/AUTO,PROG,MAN
RECALL/D(MCS)
S(home)=SNSDEF/PROBE,INDEX,POL,90.0,0.0,0,0,-1,5.35433,0.23622
SNSLCT/S(home)
FEDRAT/POSVEL,MPM,9.2
GOTO/20,40,20
TEXT/OPER,'insert stylus bcf !'
FILNAM/SENS,'bcf.sns',OUTPUT,OVERWR
RECALL/D(CAL)
FEDRAT/POSVEL,MPM,9.2
GOTO/0,0,7.84646
FEDRAT/MESVEL,MPM,0.48
MODE/AUTO,PROG,MAN
SNSET/APPRCH,0.5
SNSET/RETRCT,0.5
SNSET/SEARCH,0.5
SNSET/CLRSRF,1.5
S(0_0)=SNSDEF/PROBE,INDEX,POL,0.0,0.0,0,0,-1,5.74803,0.19685
SNSLCT/S(0_0)
```

```
GOTO/0,0,1.5
FEDRAT/POSVEL,MPM,1.5
F(CALBALL)=FEAT/SPHERE,OUTER,CART,0.0,0.0,0,1
CALIB/SENS,S(0_0),F(CALBALL),9
ENDMES
DISPLY/STOR,DMIS
SAVE/S(0_0)
FEDRAT/POSVEL,MPM,9.2
GOTO/0,0,7.84646
RECALL/D(MCS)
MODE/AUTO,PROG,MAN
FILNAM/COORD,'setcal.crd',INPUT
RECALL/D(MCS)
S(home)=SNSDEF/PROBE,INDEX,POL,90.0,0.0,0,0,-1,5.74803,0.19685
SNSLCT/S(home)
FEDRAT/POSVEL,MPM,9.2
GOTO/15,15,20
ENDFIL
```

APPENDIX C

STYLUS INFORMATION FILE

The stylus information file for the part in Appendix A will look as follows:

```
For setting the Calibration Coordinate system use :

name      : bbg
adapter   : sa3     (L = 5 mm)
extension : se4     (L = 10 mm)
stylus    : ps14r   (L = 10 mm, D = 6 mm)
length    : 0.984252 inch
diameter  : 0.23622 inch

For setting the Part Coordinate System use :

name      : bbg
adapter   : sa3     (L = 5 mm)
extension : se4     (L = 10 mm)
stylus    : ps14r   (L = 10 mm, D = 6 mm)
length    : 0.984252 inch
diameter  : 0.23622 inch

For inspecting the part use the following setups :

name      : bcf
adapter   : sa3     (L = 5 mm)
extension : se5     (L = 20 mm)
stylus    : ps13r   (L = 10 mm, D = 5 mm)
length    : 1.37795 inch
diameter  : 0.19685 inch

There is no stylus to measure B1PLANE !
```

APPENDIX D

FIXTURING FILE

The fixturing file for the part in Appendix A will look as follows:

```
DMISMN/'part.fix 3.22'
FILNAM/'ON'
FILNAM/COORD,'part.crd',OUTPUT,OVERWR
FILNAM/DATA,'part.pos',OUTPUT,OVERWR
WKPLAN/XYPLAN
PRCOMP/ON
FINPOS/OFF
UNITS/INCH,ANGDEC
MODE/PROG,MAN
FEDRAT/POSVEL,MPM,9.2
GOTO/15,15,20
DISPLY/STOR,DMIS
SAVE/D(MCS)
MODE/AUTO,PROG,MAN
RECALL/D(MCS)
S(home)=SNSDEF/PROBE,INDEX,POL,90.0,0.0,0,0,-1,5.35433,0.23622
SNSLCT/S(home)
FEDRAT/POSVEL,MPM,9.2
GOTO/20,40,20
TEXT/OPER,'insert stylus bbg !'
FILNAM/SENS,'bbg.sns',INPUT
GOTO/15,15,20
RECALL/S(0_0)
SNSLCT/S(0_0)
DISPLY/STOR,DMIS
SAVE/D(MCS)
MODE/MAN
F(OrigLine)=FEAT/LINE,UNBND, CART, 0, 0, 0, 0, 0, 1,0.0,-1,0.0
F(Origin)=FEAT/POINT,CART, 0, 0, 0,0.0,0.0,1
WKPLAN/XYPLAN
TEXT/OPER,'Measure XY plane'
```

```
F(XYPlane)=FEAT/PLANE,CART,0.0,0.0,0.0,0.0,0.0,1
MEAS/PLANE,F(XYPlane),3
ENDMES
DISPLY/COMM,DMIS,STOR,DMIS
OUTPUT/FA(XYPlane)
WKPLAN/ZXPLAN
TEXT/OPER,'Measure ZX plane'
F(ZXPlane)=FEAT/PLANE,CART,0.0,0.0,0.0,0.0,1,0.0
MEAS/PLANE,F(ZXPlane),3
ENDMES
DISPLY/COMM,DMIS,STOR,DMIS
OUTPUT/FA(ZXPlane)
WKPLAN/YZPLAN
TEXT/OPER,'Measure YZ plane'
F(YZPlane)=FEAT/PLANE,CART,0.0,0.0,0.0,1,0.0,0.0
MEAS/PLANE,F(YZPlane),3
ENDMES
DISPLY/COMM,DMIS,STOR,DMIS
OUTPUT/FA(YZPlane)
CONST/LINE,F(OrigLine),INTOF,FA(XYPlane),FA(ZXPlane)
CONST/POINT,F(Origin),INTOF,FA(YZPlane),FA(OrigLine)
D(PCS)=ROTATE/XAxis,FA(XYPlane),ZDIR
D(PCS)=ROTATE/YAxis,FA(XYPlane),ZDIR
D(PCS)=ROTATE/ZAxis,FA(YZPlane),-XDIR
D(PCS)=ROTATE/ZAxis,FA(ZXPlane),-YDIR
D(PCS)=TRANS/XOrig,FA(Origin)
D(PCS)=TRANS/YOrig,FA(Origin)
D(PCS)=TRANS/ZOrig,FA(Origin)
DISPLY/STOR,DMIS
SAVE/D(PCS)
MODE/MAN
TEXT/OPER,'Get proper clearence'
MODE/AUTO,PROG,MAN
FILNAM/COORD,'part.crd',INPUT
RECALL/D(MCS)
S(home)=SNSDEF/PROBE,INDEX,POL,90.0,0.0,0,0,-1,5.35433,0.23622
SNSLCT/S(home)
FEDRAT/POSVEL,MPM,9.2
GOTO/15,15,20
ENDFIL
```

APPENDIX E

INSPECTION FILE

The inspection file for the part in Appendix A will look as follows:

```
DMISMN/'CBore1-1.ins 3.22'
FILNAM/'ON'
FILNAM/COORD,'part.crd',INPUT
FILNAM/DATA,'CBore1-1.dat',OUTPUT,OVERWR
WKPLAN/XYPLAN
PRCOMP/ON
FINPOS/OFF
UNITS/INCH,ANGDEC
MODE/PROG,MAN
FEDRAT/POSVEL,MPM,9.2
GOTO/15,15,20
DISPLY/STOR,DMIS
SAVE/D(MCS)
MODE/AUTO,PROG,MAN
RECALL/D(MCS)
S(home)=SNSDEF/PROBE,INDEX,POL,90.0,0.0,0,0,-1,5.74803,0.19685
SNSLCT/S(home)
FEDRAT/POSVEL,MPM,9.2
GOTO/20,40,20
RECALL/D(PCS)
TEXT/OPER,'insert stylus bcf !'
FILNAM/SENS,'bcf.sns',INPUT
GOTO/0,0,7.84646
RECALL/S(0_0)
SNSLCT/S(0_0)
SNSET/APPRCH,0.0656168
SNSET/RETRCT,0.0656168
SNSET/SEARCH,0.5
SNSET/CLRSRF,0.5
SNSET/DEPTH,0.0712598
```

```
FEDRAT/POSVEL,MPM,9.2
GOTO/0.0,0.0,2.5
GOTO/0.0,0.0,0.5
FEDRAT/POSVEL,MPM,1.5
FEDRAT/MESVEL,MPM,0.48
MODE/PROG,MAN
F(B1A1)=FEAT/CYLNDR,INNER,CART,0.0,0.0,-0.12126,0.0,0.0,1.0,0.6
MEAS/CYLNDR,F(B1A1),10
PTMEAS/CART,0.3,0.0,-0.19252,-1.0,0.0,0.0
PTMEAS/CART,0.0927051,0.285317,-0.19252,-0.309017,-0.951057,0.0
PTMEAS/CART,-0.242705,0.176336,-0.19252,0.809017,-0.587785,0.0
PTMEAS/CART,-0.242705,-0.176336,-0.19252,0.809017,0.587785,0.0
PTMEAS/CART,0.0927051,-0.285317,-0.19252,-0.309017,0.951057,0.0
PTMEAS/CART,0.3,0.0,-0.05,-1.0,0.0,0.0
PTMEAS/CART,0.0927051,0.285317,-0.05,-0.309017,-0.951057,0.0
PTMEAS/CART,-0.242705,0.176336,-0.05,0.809017,-0.587785,0.0
PTMEAS/CART,-0.242705,-0.176336,-0.05,0.809017,0.587785,0.0
PTMEAS/CART,0.0927051,-0.285317,-0.05,-0.309017,0.951057,0.0
ENDMES
DISPLY/COMM,DMIS,STOR,DMIS
OUTPUT/F(B1A1)
DISPLY/COMM,DMIS,STOR,DMIS
OUTPUT/FA(B1A1)
GOTO/0.0,0.0,0.5
RECALL/S(0_0)
SNSLCT/S(0_0)
SNSET/APPRCH,0.0656168
SNSET/RETRCT,0.0656168
SNSET/SEARCH,0.5
SNSET/CLRSRF,0.5
SNSET/DEPTH,0.325
FEDRAT/POSVEL,MPM,9.2
GOTO/0.0,0.0,0.5
FEDRAT/POSVEL,MPM,1.5
FEDRAT/MESVEL,MPM,0.48
MODE/PROG,MAN
F(B1A2)=FEAT/CYLNDR,INNER,CART,0.0,0.0,-0.675,0.0,0.0,1.0,0.5
MEAS/CYLNDR,F(B1A2),10
PTMEAS/CART,0.25,0.0,-1.0,-1.0,0.0,0.0
PTMEAS/CART,0.0772542,0.237764,-1.0,-0.309017,-0.951057,0.0
PTMEAS/CART,-0.202254,0.146946,-1.0,0.809017,-0.587785,0.0
PTMEAS/CART,-0.202254,-0.146946,-1.0,0.809017,0.587785,0.0
PTMEAS/CART,0.0772542,-0.237764,-1.0,-0.309017,0.951057,0.0
```

```
PTMEAS/CART,0.25,0.0,-0.35,-1.0,0.0,0.0
PTMEAS/CART,0.0772542,0.237764,-0.35,-0.309017,-0.951057,0.0
PTMEAS/CART,-0.202254,0.146946,-0.35,0.809017,-0.587785,0.0
PTMEAS/CART,-0.202254,-0.146946,-0.35,0.809017,0.587785,0.0
PTMEAS/CART,0.0772542,-0.237764,-0.35,-0.309017,0.951057,0.0
ENDMES
DISPLY/COMM,DMIS,STOR,DMIS
OUTPUT/F(B1A2)
DISPLY/COMM,DMIS,STOR,DMIS
OUTPUT/FA(B1A2)
GOTO/0.0,0.0,0.5
RECALL/S(0_0)
SNSLCT/S(0_0)
SNSET/APPRCH,0.02
SNSET/RETRCT,0.02
SNSET/SEARCH,0.5
SNSET/CLRSRF,0.5
MODE/PROG,MAN
FEDRAT/POSVEL,MPM,9.2
GOTO/0.0,0.0,0.5
FEDRAT/POSVEL,MPM,1.5
FEDRAT/MESVEL,MPM,0.48
F(B1B)=FEAT/CONE,INNER,CART,0.0,0.0,0.0,0.0,0.0,1.0,90
MEAS/CONE,F(B1B),6
PTMEAS/CART,0.305,0.0,-0.045,-0.707107,0.0,0.707107
PTMEAS/CART,-0.1525,0.264138,-0.045,0.353553,-0.612372,0.707107
PTMEAS/CART,-0.1525,-0.264138,-0.045,0.353553,0.612372,0.707107
PTMEAS/CART,0.345,0.0,-0.005,-0.707107,0.0,0.707107
PTMEAS/CART,-0.1725,0.298779,-0.005,0.353553,-0.612372,0.707107
PTMEAS/CART,-0.1725,-0.298779,-0.005,0.353553,0.612372,0.707107
ENDMES
DISPLY/COMM,DMIS,STOR,DMIS
OUTPUT/F(B1B)
DISPLY/COMM,DMIS,STOR,DMIS
OUTPUT/FA(B1B)
GOTO/0.0,0.0,0.5
FEDRAT/POSVEL,MPM,9.2
GOTO/0.0,0.0,2.5
FEDRAT/POSVEL,MPM,9.2
GOTO/0.0,0.0,8.87008
MODE/AUTO,PROG,MAN
FILNAM/COORD,'part.crd',INPUT
RECALL/D(MCS)
```

```
S(home)=SNSDEF/PROBE,INDEX,POL,90.0,0.0,0,0,-1,5.74803,0.19685
SNSLCT/S(home)
FEDRAT/POSVEL,MPM,9.2
GOTO/15,15,20
ENDFIL
```

REFERENCES

[1] American National Standard. *ANSI Y14.5M - 1982*, 1982.

[2] Bhanu, B. Guest editor's introduction, special isue on cad-based robot vision. *Computer* (Aug. 1987).

[3] Bhanu, B., and Ho, C.-C.Plossl, K. Cad-based 3d object recognition for robot vision. *Computer Vol. 20* (Aug 1987), pp. 19–35.

[4] Camps, O.I. Shapiro, L., and Haralick, R. Premio: The use of prediction in a cad-model-based vision system. Tech. rep., Dep. Elec. Eng. Univ. of Washington, Tech. Rep. EE-ISL-89-01, 1989.

[5] Chen, C., and Kak, A. A robot vision system for recognizing 3-d objects in low-order polynominal time. *IEEE Trans. Syst. Man Cybern. vol. 19*, no. 6 (Nov./Dec. 1989), pp. 1535–1563.

[6] Computer Aided Manufacturing International, Arlington, Texas. *ANSI/CAM-I 101-1990*.

[7] Computer Aided Manufacturing International, Arlington, Texas. *CAMI*.

[8] Computer Science Department, University of Utah. *PSL 3.4 User's Manual*, 1987.

[9] Computer Science Department, University of Utah. *Alpha_1 User's Manual*, June 1992.

[10] Corrigall, M., and Bell, R. An inspection plan and code generator for coordinate measuring machines. In *9th International Conference on Automated Inspection and Product Control* (Loughborough University of Technology, UK, 1989), pp. 145–154.

[11] Fanamation, Compton, C. Fanamation comero 404024 specification. Tech. rep., August 1991.

[12] Fanamation, Compton, California. *Airfoil Inspection Software*, 1990.

[13] Fanamation, Compton, California. *CMIS Command Reference Manual*, version 3.20 1990.

[14] Fanamation, Compton, California. *CMIS User Interface*, 1990.

[15] Fanamation, Compton, California. *Intergrated Surface Inspection Software*, 1990.

[16] Fanamation, Compton, California. *Tubular Shape Analyses Software*, 1990.

[17] Fanamation, Compton, California. *CMIS Command Reference Manual*, version 3.20 1990, Ch.1, pp. 1-13.

[18] Flynn, p. J., and Jain, A. K. Cad-based computer vision: From cad models to related graphs. *Pattern Analyses and Machine Intelligence vol 13.*, no. 2 (Feb. 1991), pp. 114–132.

[19] Hansen, C., and T., H. Cagd-based computer vision. *IEEE Trans. Pattern Analyses and Machine Intelligence vol 11.*, no. 11 (Nov. 1989), pp. 1181–1193.

[20] K., I., and T., K. Automated generation of object recognition programs. *Proc. IEEE vol. 76*, no. 8 (Aug. 1988), pp. 1016–1035.

[21] McDonnel Douglas, Manufacturing & Engineering. *Valisys*, 1990.

[22] Merat, F. L., and Radack, G. M. Automatic inspection planning within a feature-based cad system. *Robotics and Computer-Intergrated Manufacturing Vol. 9*, No. 1 (1992), pp. 61–70.

[23] Plossl, K. Production in the factory of the future. *International Journal of Production Research Vol. 26*, No. 3 (1988), pp. 501–506.

[24] Renishaw. Probing for productivity on co-ordinate measuring machines. Tech. rep., RENISHAW, 1988.

[25] SILMA incorporated, Cupertino, California. *CimStation*, May 1990.

[26] Stile, E. M. Engineering the 1990s inspection function. *Quality Progress Vol. 10*, No. 11 (1987), pp. 70–71.

[27] three-dimensional part orientation system, D. A. Bolles, r.c. and horaud, p. *Int. J. Robotics Res. vol. 5*, no. 3 (Fall 1986), pp. 3–26.

[28] Villers, P. *Intelligent Robots: Moving Towards Meggassembly In The AI Business.* MIT Press, Cambridge, MA, 1984, pp. pp. 205–222.

[29] Wisdom Systems, Pepper Pike, Ohio. *Concept Modeler.*