NISTIR 5970

Canonical Machining Commands

Frederick M. Proctor
Control Systems
Intelligent Systems Division

Thomas R. Kramer
Research Associate

Department of Mechanical Engineering
The Catholic University of America
Washington, DC 20064
and Intelligent Systems Division

and

John L. Michaloski
Control Systems
Intelligent Systems Division

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Bldg. 220 Rm. B124
Gaithersburg, MD 20899

NIST

NISTIR 5970

Canonical Machining Commands

Frederick M. Proctor
Control Systems
Intelligent Systems Division

Thomas R. Kramer
Research Associate

Department of Mechanical Engineering
The Catholic University of America
Washington, DC 20064
and Intelligent Systems Division

and

John L. Michaloski
Control Systems
Intelligent Systems Division

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Bldg. 220 Rm. B124
Gaithersburg, MD 20899

January 1997

U.S. DEPARTMENT OF COMMERCE
Michael Kantor, Secretary

TECHNOLOGY ADMINISTRATION
Mary L. Good, Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
Arati Prabhakar, Director

Canonical Machining Commands

Frederick M. Proctor
Thomas R. Kramer
John L. Michaloski

Intelligent Systems Division
National Institute of Standards and Technology
Technology Administration
U.S. Department of Commerce
Gaithersburg, Maryland 20899

NISTIR 5970
January 30, 1997

Disclaimer

No approval or endorsement of any commercial product by the National Institute of Standards and Technology is intended or implied. Certain commercial equipment, instruments, or materials are identified in this report to facilitate understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

Acknowledgements

Partial funding for the work described in this paper was provided to Catholic University by the National Institute of Standards and Technology under cooperative agreement Number 70NANB2H1213.

Copyright

CONTENTS

1 Introduction

The Intelligent Systems Division (ISD) of the National Institute of Standards and Technology (NIST) is carrying out an Enhanced Machine Controller (EMC) project[1]. This project is developing a testbed for open architecture controllers in which to validate potential software interface specification standards. The project is also developing and implementing an open hierarchical architecture for control of machine tools.

In the EMC control architecture, at each hierarchical level, a controller takes commands from its superior controller and gives commands to its subordinate controllers. At each interface between superior and subordinate controllers, there is a specific set of commands which the superior can send and the subordinate can receive, and the meaning of each command is specified.

This report focuses on the commands which are used at the interface between a controller which can be given a “run control program” command or an “execute a line of code” command and its subordinate(s) which can perform motion control and discrete I/O functions (such as turning switches on and off). The commands at this interface are called the “canonical machining commands.”

1.1 Numerical Control Programming Language RS274

In the EMC project, a “run control program” or “execute a line of code” command refers to a program or line of code written in the RS274 programming language. This is a programming language for numerically controlled (NC) machine tools, which has been used for many years. The most recent standard version of RS274 is RS274-D, which was completed in 1979. It is described in the document “EIA Standard EIA-274-D” by the Electronic Industries Association [EIA]. Most NC machine tools can be run using programs written in RS274. Implementations of the language differ from machine to machine, however, and a program that runs on one machine probably will not run on one from a different maker.

The EMC project has dealt with three dialects of RS274, which are being called RS274KT, RS274/NGC, and RS274/VGER.

RS274KT is for a 4-axis Kearney and Trecker 800 machining center and is described in the programming manual for that machine [K&T].

RS274/NGC is an enhanced version of RS274 developed as part of the Next Generation Controller (NGC) project. The RS274/NGC specification was originally given in a 1992 report prepared by the Allen-Bradley company, a division of Rockwell International Corp. [Allen-Bradley]. A second draft of that document was released in 1994 by the National Center for Manufacturing Sciences [NCMS]. The RS274/NGC language has many capabilities beyond those of RS274-D.

RS274/VGER is a modified version of RS274/NGC, as described below.

1.2 Work on Canonical Machining Commands and RS274 Interpreters at NIST

As part of ISD assistance to the program which developed the NGC architecture, ISD prepared a

1. The current ISD EMC project evolved from a project joint with NIST’s Automated Production Technology Division in which a Monarch VMC-75 machining center was retrofitted with an open architecture controller.

report “NIST Support to the Next Generation Controller Program: 1991 Final Technical Report,” [Albus] containing a variety of suggestions. Appendix C to that report proposed three sets of commands for 3-axis machining, one set for each of three proposed hierarchical control levels. The suite proposed for the lowest (primitive) control level was implemented in 1993 by the EMC project as a set of canonical machining functions in the C programming language.

Also in 1993, the authors developed a software system in the C language for reading machining commands in the RS274/NGC language and outputting canonical machining functions. This was called “the RS274/NGC interpreter.” A report describing that interpreter was published in April 1994 [Kramer1].

In 1994 and 1995, the EMC project team, in collaboration with the General Motors Company, retrofitted a 4-axis Kearney and Trecker 800 machine with an EMC controller [Proctor]. The goal of ISD in performing this retrofit was to test out “plug and play” capabilities for the open architecture in a shop floor environment. To deal with the Kearney and Trecker machine, the canonical machining commands were revised to include four axes. Also, version 2 of the RS274/NGC interpreter was built, as was an RS274KT interpreter, which interprets programs written in the Kearney and Trecker dialect of RS274. Separate reports [Kramer2], [Kramer3] describe these interpreters.

In 1995 the EMC project collaborated with several industrial partners in an open-architecture machine tool controller project known as VGER (which is a name, not an acronym). This project retrofitted a Shin Nippon Koki (SNK) 5-axis machine tool with a new open architecture controller. NIST provided the RS274 interpreter for the VGER project. It was intended to be able to interpret some existing programs for the SNK machine which were written for its former Fanuc controller. The NGC and Fanuc dialects of RS274 were almost identical, but there were minor differences. Thus, the RS274/VGER interpreter was written to take Fanuc-flavored RS274/NGC code as input. This language is called the RS274/VGER language in this report. The RS274/VGER interpreter is described in [Kramer4]. The canonical machining commands were further refined and extended to five axes for the VGER project.

1.3 Canonical Machining Commands

1.3.1 Objectives of Canonical Machining Commands

The canonical machining commands were devised with three objectives in mind.

First, all the functionality of common 3-axis to 6-axis machining centers (including the K&T and SNK machines, not including turning centers) had to be covered by the commands; for any function the machine can perform, there has to be a way to tell it to do that function.

Second, it was desired that it be possible to use readily available commercial motion control boards from various vendors to carry out those canonical commands which call for motion, with roughly a one-to-one correspondence between a canonical motion command and a command a commercial board recognizes.

Third, it must be possible to interpret RS274 commands into canonical commands.

The canonical machining commands are atomic commands. Each command produces a single tool motion or a single logical action. Keeping the commands atomic was a side condition we imposed, both for clarity and to make mappings to commercial control boards straightforward. RS274 commands, on the other hand, include two types: those for which a single RS274

command corresponds exactly to a canonical command, and those for which a single RS274 command will be decomposed into several canonical commands (possibly dozens). Things like “move in a straight line” or “turn flood coolant on” are of the first type. Things like “turn all coolant off” or “run a peck drilling cycle” are of the second type.

1.3.2 Implementing Canonical Machining Commands

This report gives the syntax of the canonical commands and gives function prototypes in the C or C++ programming language. The report also gives the semantics of the commands — what should happen when each command is executed. The report does not provide definitions of the commands in any programming language. Sets of definitions for the canonical machining commands have been written in C++ for the controllers we have built which use them. Executing a function from any of these sets of definitions causes a machining center to do something.

We have used two other types sets of definitions, in addition. The first alternate type simply has the command print itself; executing a command causes a line of text containing the command to be written to standard output or to a file. This type of set of definitions is useful for testing and debugging an RS274 interpreter. The second alternate type is also used for testing and debugging. In this type, executing a canonical command generates a set of graphics calls for displaying a computer picture of the tool path made by executing the command. Using this set with a stand-alone interpreter provides a graphical simulation of executing an RS274 program.

The existing interpreters do not use all of the canonical machining commands. The canonical machining commands that are used by the interpreters are listed in [Kramer1], [Kramer2], [Kramer3], and [Kramer4].

2 Canonical Machining Command View of a Machining Center

The canonical machining commands are based on a particular view of what a machining center to be controlled is like. Compared with the view of a machining center taken by RS274, this is the same mechanically, but simpler from a control point of view. For example, RS274 assumes the controller can perform cycles with many motions, such as peck drilling, while the canonical commands assume that only atomic motions can be performed.

2.1 Machine Dynamics

The canonical commands share with the RS274 language the simplifying assumption that machine dynamics can be ignored. That is, in this model, acceleration and deceleration do not occur. Components of the machine can be told to move at a specific rate, and that rate is imagined as being achieved instantaneously. Stopping is also imagined as instantaneous. This model obviously does not correspond with reality, and there are situations in which the model is not adequate. The canonical commands contain special instructions to work around these situations. It is assumed that the controller receiving the canonical commands knows how to deal with machine dynamics.

2.2 Mechanical Components

2.2.1 Coolant

A machining center has one, two, or three kinds of coolant: mist coolant, flood coolant, and through-tool coolant. Each can be turned on and off independently.

2.2.2 Axes

The work volume of a machining center is described using at least three axes labelled X, Y, and Z. The X, Y, and Z axes form a standard right-handed coordinate system of orthogonal linear axes.

Any or all of three additional axes, A, B, and C, may be used. These are rotational axes. The A axis is a rotational axis parallel to the X axis. B is parallel to the Y axis, and C parallel to the Z axis.

The rotational axes are measured as wrapped linear axes in which the direction of positive rotation is counterclockwise when viewed from the positive end of the corresponding X, Y, or Z axis. By "wrapped linear axis," we mean one on which the angular position increases without limit (goes towards plus infinity) as the axis turns counterclockwise and deceases without limit (goes towards minus infinity) as the axis turns clockwise. An alternative method of measuring the turning of a rotational axis is discussed in Appendix A.4. If an interpreter is used to generate canonical machining commands, and the language which the interpreter reads does not use the wrapped linear axis format, the interpreter must convert to the wrapped linear format for giving canonical machining commands.

Clockwise or counterclockwise is from the point of view of the workpiece. If the workpiece is fastened to a turntable which turns on a rotational axis, a counterclockwise turn from the point of view of the workpiece is accomplished by turning the turntable in a direction that (for most common machine configurations) looks clockwise from the point of view of someone standing next to the machine.

The X, Y, and Z axes are usually implemented in the structure of a machine by linear mechanisms that allow for sliding motion parallel to one or another of them. The A, B, and C axes are usually implemented by mechanical rotation about an axis. There is no requirement that there be physical mechanisms corresponding to axes.

Many machining centers also have axes parallel to the X, Y, and/or Z axes. These additional axes are generally intended for gross positioning; the axis will be moved at the beginning of the program and then left fixed. The model given here does not provide for such axes.

2.2.3 Axis Clamps

Each axis may be clamped so it does not move. The clamp for an axis usually is a physical mechanism, but it is not required to be so; software clamps may be used. Each axis clamp may be commanded to clamp or unclamp. Clamping any one axis or set of axes does not affect motion along the unclamped axes.

2.2.4 Spindle

The machine has a spindle which holds one cutting tool. The spindle can rotate in either direction, and it can be told to rotate at a constant rate, which may be changed. Except on machines where the spindle may be moved by moving a rotational axis, the axis of the spindle is kept parallel to the Z axis and is the Z axis when X and Y are zero. The spindle can be stopped in a fixed orientation or stopped without specifying orientation.

2.2.5 Pallet Shuttle

The machine may have a pallet shuttle system. Such a system has two pallets on which workpieces can be fixtured. The two pallets may be exchanged by command.

2.2.6 Tool Carousel

The machine has a tool carousel with slots for tools fixed in tool holders. There is zero or one tool assigned to each slot. There is no fixed number of slots in the abstract model of a machine, but each actual machine will have some specific number of slots, and it is an error on any actual machine to refer to a slot which the machine does not have.

2.2.7 Tool Changer

The machine has a mechanism for changing tools (fixed in tool holders) between the spindle and the tool carousel.

2.2.8 Operator Console

The machine has an operator console which can display messages and take operator input. The machine may also have ancillary panels or pendants.

2.3 Controlled Motions

The motion controller of a machining center can control axes and spindle motion simultaneously in several ways.

2.3.1 Controlled Point

The controlled point is the point whose position and rate of motion are controlled. When the tool length offset is zero (the default value), this is a point on the spindle axis (often called the gauge point) that is some fixed distance beyond the end of the spindle, usually near the end of a tool holder that fits into the spindle. The location of the controlled point can be moved out along the spindle axis by specifying some positive amount for the tool length offset. This amount is normally the length of the cutting tool in use, so that the controlled point is at the end of the cutting tool.

2.3.2 Linear Motion

The X, Y, and Z axes can be controlled simultaneously so that the motion produced is in a straight line in any direction. While linear XYZ motion is being performed, the rotational axes can be controlled simultaneously so that they start and stop when the XYZ motion starts and stops. The acceleration and deceleration of all axes can be controlled so that the path of the controlled point is what it would be if starting and stopping were instantaneous and each axis moved at a constant rate throughout the motion.

Linear motion can be performed either at the prevailing feed rate, or as fast as possible, subject to an upper bound which may be set by command.

2.3.3 Arc Motion

Any pair of the linear axes (XY, YZ, XZ) can be controlled to move in a circular or elliptical arc in the plane of that pair of axes. While this is occurring, the third linear axis and the rotational axes can be controlled to move simultaneously at effectively a constant rate. As above, the motions can be coordinated so that acceleration and deceleration do not affect the path.

2.3.4 Coordinated Motion of Axes and Spindle

The rotation of the spindle can be coordinated with axis motion or de-coupled from it.

2.3.5 Feed Rate

The rate at which the controlled point or the axes move is nominally a steady rate which may be set by the user. How this rate applies is discussed in Section 3.5.2.

2.3.6 Feed and Speed Overrides

The machine operator has a switch which (when enabled) allows the operator to control the feed rate. There is a similar switch for operator control of spindle speed. These two switches may be enabled or disabled independently.

2.3.7 Dwell

The machine may be command to keep all axes unmoving for a specific amount of time.

2.3.8 Units

Units used for distances along the X, Y, and Z axes may be measured in millimeters, centimeters, or inches. Units for all other quantities involved in machine control cannot be changed. Different quantities use different specific units. Spindle speed is measured in revolutions per minute. The positions of rotational axes are measured in degrees. Feed rates are expressed in current length units per minute or in degrees per minute, as described in Section 3.5.1.

2.3.9 Coordinate Systems

There are two coordinate systems: the absolute coordinate system of the machine and a "program" coordinate system in which all axes may be offset from their absolute position. The offsets may be changed by command. All axis values given in all commands (other than a command to set the offsets) are expressed in terms of the program coordinate system.

2.3.10 Current Position

The controlled point is always at some location called the "current position," and the controller always knows where that is. The numbers representing the current position must be adjusted in the absence of any axis motion if any of several events take place:

1. Length units are changed.
2. Tool length offset is changed.
3. Coordinate system offsets are changed.

2.3.11 Selected Plane

There is always a "selected plane" for arc motion, which must be the XY plane, the YZ plane, or the XZ plane of the machine.

2.4 Error Conditions

The controller which executes the canonical machining commands should have at least two conditions: OK and error. If the controller is in the OK condition, it should perform as described here. If the controller is in the error condition, its behavior is up to the implementation. An implementation should document what its behavior is for all errors. A generic error behavior which we believe is reasonable is to stop execution of the current command as soon as an error is detected and to be unwilling to execute any other canonical commands until the error condition is corrected.

Many situations should put the controller into the error condition, as described later in this report.

Wherever the phrase “it is an error” (or the equivalent) is used to describe a situation that may occur in the execution of a command, the controller should always check for that situation during execution of the command and should go into the error condition as soon as the situation is detected if the situation occurs.

In this report, error detection is described as being performed only by the controller receiving the canonical commands. In an implementation, error detection might also be performed by the controller sending the commands.

3 The Canonical Machining Commands Defined

3.1 Preliminaries

3.1.1 Syntax

The canonical machining commands are defined here using C++ syntax in `courier` type font. The same syntax is usable in ANSI C. Syntaxes for other computer languages may be derived readily without changing the semantics of the commands. All the canonical functions return void, so that bit of syntax is suppressed here.

Where an attribute may take on discrete values from a fixed set of values, the set is given here. The values in such sets are represented here as symbols beginning with “CANON_”. In our implementations we have used both `#define`’s and `enum`’s for these symbols.

As given here, the commands include the A, B, and C rotational axes. If any of the rotational axes is not to be used, just delete the references to that axis. In C++, it is feasible to do that using `#ifdef`’s as shown in Appendix B.

3.1.2 Command Call Errors

It is an error to call any command incorrectly. The arguments to a command must be as described here. This applies to the number and type of the arguments and also to the stated constraints on the arguments (such as being non-negative or being positive).

3.1.3 Groups of Commands

The canonical machining commands are grouped here into related sets. The grouping is for convenience only, and is not part of the definition. Within each group, the commands are listed alphabetically.

3.1.4 Coordinated Motion

To drive a tool along a specified path, a machine tool must often coordinate the motion of several axes. We use the term “coordinated linear motion” to describe the situation in which, nominally, each axis moves at constant speed and all axes move from their starting positions to their end positions at the same time. If only the X, Y, and Z axes (or any one or two of them) move, this produces motion in a straight line, hence the word “linear” in the term. In actual motions, it is often not possible to maintain constant speed because acceleration or deceleration is required at the beginning and/or end of the motion. It is feasible, however, to control the axes so that, at all times, each axis has completed the same fraction of its required motion as the other axes. This moves the tool along same path, and we also call this kind of motion “coordinated linear motion”.

The axes may also be controlled to produce specific trajectories other than straight lines —

circular arcs, for example. This is also coordinated motion, but it is not linear.

3.1.5 Rotational Motion Required

In the descriptions of the commands, the phrase “if there is rotational motion” is often used. In all cases, there is rotational motion if any of the rotational axis values in the command differs from the current value of that axis.

3.2 Initialization and Termination

3.2.1 `INIT_CANON ()`

Do whatever initialization is required to be ready to execute other canonical machining commands. This command should always be issued before any other canonical machining command. Of course, the controller must already have been brought up to the state where it can read and execute this command before the command is received. That phase of initialization is outside the scope of this report.

3.2.2 `END_CANON ()`

Do whatever is required to shut down in an orderly fashion. After this command has been executed, if it is desired to begin executing commands again, the first command must be an INIT_CANON command.

3.3 Representation

3.3.1 `SELECT_PLANE (CANON_PLANE plane)`

Use the plane designated by `plane` as the selected plane. Acceptable values of `plane` are CANON_PLANE_XY, CANON_PLANE_XZ, and CANON_PLANE_YZ.

3.3.2 `SET_ORIGIN_OFFSETS (double x, double y, double z, double a, double b, double c)`

Set the program origin at the point with absolute coordinates `x`, `y`, `z`, `a`, `b`, and `c`. The units for `x`, `y`, and `z` are whatever length units are being used at the time this command is given. The units for `a`, `b`, and `c` are degrees. The effective location of the program origin should not change when units change. It is expected that controllers dealing with the program origin will ensure this. In a typical implementation, the numbers representing the coordinates will be changed when units change.

3.3.3 `USE_LENGTH_UNITS (CANON_UNITS units)`

Use the specified `units` for length. Acceptable values of `units` are CANON_UNITS_INCHES (inches), CANON_UNITS_MM (millimeters), and CANON_UNITS_CM (centimeters). Changing units changes the effective numerical value of all stored values which involve length units, including: current position, coordinate system offsets, tool length offsets, tool diameters, feed rate, and traverse rate.

It is up to an implementation to decide how to do this. One approach is to store all data in one unit only and apply an appropriate conversion factor any time the data is retrieved. Another approach is to have a separate table for each unit and switch back and forth between tables, as needed. In the second approach, whenever an entry is made while one unit is being used, the entry is

converted appropriately and entered into all three tables.

3.4 Free Space Motion

3.4.1 SET_TRAVERSE_RATE (double rate)

Set the upper limit on the rate that will be used during rapid axis motion, motion during which cutting does not normally take place. During moves conducted at traverse rate, the machine should produce coordinated linear motion as fast as possible or at this `rate`, whichever is less. The `rate` must be positive. The application of the rate is as described in Section 3.5.1 for when the feed reference mode is CANON_XYZ.

The traverse rate changes automatically if length units change.

3.4.2 STRAIGHT_TRAVERSE (double x, double y, double z, double a, double b, double c)

Make a coordinated linear motion at traverse rate from the current position to the point given by `x`, `y`, `z`, `a`, `b`, and `c`. The application of the rate is as described in Section 3.5.1 for when the feed reference mode is CANON_XYZ, regardless of the setting of feed reference mode.

It is expected that no cutting will occur while a traverse move is being made.

3.5 Machining Attributes

3.5.1 SET_FEED_RATE (double rate)

Set to `rate` the feed rate that will be used when the controlled point is told to move at the currently set feed rate. The `rate` must be non-negative. The number representing feed rate changes if length units are changed, so that the effective feed rate does not change.

1. If the feed reference mode is CANON_WORKPIECE:
the `rate` means length units per minute of the controlled point along the programmed path as seen by the workpiece.

2. If the feed reference mode is CANON_XYZ:

A. For motion involving one or more of the X, Y, and Z axes (with or without simultaneous rotational axis motion), the `rate` means length units per minute along the programmed XYZ path, as if the rotational axes were not moving.

B. For motion of one rotational axis with X, Y, and Z axes not moving, the `rate` means degrees per minute rotation of the rotational axis.

C. For motion of two or three rotational axes with X, Y, and Z axes not moving, the rate is applied as follows. Let *dA*, *dB*, and *dC* be the angles in degrees through which the A, B, and C axes, respectively, must move. Let $D = \sqrt{(dA)^2 + (dB)^2 + (dC)^2}$. Conceptually, D is a measure of total angular motion, using the usual Euclidean metric. Let *T* be the amount of time required to move through *D* degrees at the `rate` in degrees per minute. The rotational axes should be moved in coordinated linear motion so that the elapsed time from the start to the end of the motion is *T*.

If only one rotational axis moves, the method in paragraph C above gives the same result as that described in paragraph B.

The canonical machining commands (unlike RS274) do not include the notion of inverse time feed rate, so the question of behavior under inverse time feed rate does not arise.

The feed rate may also be affected by the feed override switch, if overrides are enabled, and by the force on the spindle, if the USE_SPINDLE_FORCE command is in effect.

3.5.2 `SET_FEED_REFERENCE (CANON_FEED_REFERENCE reference)`

This sets the feed reference mode. Acceptable values of `reference` are CANON_WORKPIECE and CANON_XYZ.

The meaning of feed rate changes depending on the feed reference mode. See the discussion immediately above.

The CANON_WORKPIECE feed reference mode is more natural and general, since the rate at which the tool passes through the material must be controlled for safe and effective machining. This mode does introduce complications, however.

First, some rule is required to define what the path should be. We have adopted the rule that the path should be the same as it is in the CANON_XYZ mode.

Second, computing the feed rate for each axis may be time-consuming because the trajectories that result from motion in four or more axes may be complex. Computation of axis feed rates when only XYZ motion is considered is relatively simple for two of the standard motion types (straight lines and helical or circular arcs).

Third, in CANON_WORKPIECE mode, some motions cannot be carried out as fast as the programmed feed rate would require because axis motions may tend to cancel each other. For example, an arc in the XZ plane can exactly cancel a rotation around the B axis, so that the location of the controlled point with respect to the workpiece does not change at all during the motion; in this case, the motion should take no time, which is impossible at any finite rate of axis motion. In such cases, the axes should be moved as fast as possible consistent with accurate machining.

Some (perhaps most or all) existing dialects of RS274 use only the CANON_XYZ mode [K&T, page 3.9]. The specification of RS274/NGC is not clear [NCMS, page 22], but appears to intend CANON_XYZ mode. Some dialects avoid the problem of dealing with how to interpret a per-minute feed rate when rotational axis motion occurs simultaneously with XYZ motion by suggesting [K&T, page 3.9] or requiring [Monarch, page 17-3] that the programmer use inverse-time feed mode. It may be that the calculations required in CANON_WORKPIECE mode were too extensive to be carried out sufficiently fast by real-time processors that existed at the time these languages were defined. Current real-time processors should be sufficiently fast to handle the calculations.

3.5.3 `SET_MOTION_CONTROL_MODE (CANON_MOTION_MODE mode)`

Set the motion control mode. Acceptable values of `mode` are CANON_EXACT_STOP, CANON_EXACT_PATH, and CANON_CONTINUOUS. This affects the way in which motion commands are carried out.

In CANON_EXACT_STOP mode, the control stops motion at the end of each move exactly (within the tolerance the control system can achieve) at the programmed or calculated end point.

The stop is preceded by deceleration at the maximum normal rate, so that motion is kept at the set feed rate for as long as possible. If there is a subsequent move, the stop is as brief as possible and is followed by acceleration to the programmed feed rate at the maximum normal rate of acceleration.

In CANON_EXACT_PATH mode, the control keeps the controlled point on the programmed or calculated path within the tolerance the control system can achieve at all times. At points which are the end point of one move and the start point of the next move, the feed rate is kept constant if possible. It should be possible to avoid changing the rate of motion at such a juncture if the direction of the path of the controlled point does not change sharply at the juncture, for example if one straight move is in the same direction as the previous one, or if a straight move is tangent to a preceding or following arc move.

In CANON_CONTINUOUS mode, the control tries to keep the feed rate constant and does not try to keep the controlled point exactly on the path at all times. Rather, at junctures between moves where the direction changes sharply, the corner is rounded. There is a maximum allowable deviation at such junctures, and the control should never allow that to be exceeded; acceleration and deceleration may be performed if necessary to do this.

Currently, there is no command to set the maximum deviation allowable in CANON_CONTINUOUS mode. There probably should be such a command.

3.5.4 `START_SPEED_FEED_SYNCH ()`

Begin exact synchronization of spindle turning with feed motion.

The primary purpose of this synchronization is to provide for effective tapping of holes. In order to make a clean thread, the axial motion of a tap must be synchronized with its turning motion. In addition to this synchronization, the feed and speed rates must be set so their ratio is suitable for the pitch of the thread.

3.5.5 `STOP_SPEED_FEED_SYNCH ()`

Stop forcing synchronization of spindle turning with feed motion. Deal with acceleration and deceleration of the spindle and the axes independently.

3.6 Machining Functions

3.6.1 `ARC_FEED (double first_end, double second_end, double first_axis, double second_axis, int rotation, double axis_end_point, double a, double b, double c)`

Move in a helical arc from the current position at the existing feed rate. The axis of the helix is parallel to the X, Y, or Z axis, according to which one is perpendicular to the selected plane. The helical arc may degenerate to a circular arc if there is no motion parallel to the axis of the helix.

If the selected plane is the XY plane:

1. `first_end` is the X coordinate of the end of the arc.
2. `second_end` is the Y coordinate of the end of the arc.
3. `first_axis` is the X coordinate of the axis (center) of the arc.
4. `second_axis` is the Y coordinate of the axis (center) of the arc.
5. `axis_end_point` is the Z coordinate of the end of the arc.

If the selected plane is the YZ plane:

1. `first_end` is the Y coordinate of the end of the arc.
2. `second_end` is the Z coordinate of the end of the arc.
3. `first_axis` is the Y coordinate of the axis (center) of the arc.
4. `second_axis` is the Z coordinate of the axis (center) of the arc.
5. `axis_end_point` is the X coordinate of the end of the arc.

If the selected plane is the XZ plane:

1. `first_end` is the Z coordinate of the end of the arc.
2. `second_end` is the X coordinate of the end of the arc.
3. `first_axis` is the Z coordinate of the axis (center) of the arc.
4. `second_axis` is the X coordinate of the axis (center) of the arc.
5. `axis_end_point` is the Y coordinate of the end of the arc.

If rotation is positive, the arc is traversed counterclockwise as viewed from the positive end of the coordinate axis perpendicular to the currently selected plane. If rotation is negative, the arc is traversed clockwise. If rotation is 0, `first_end` and `second_end` must be the same as the corresponding coordinates of the current position and no arc is made (but there may be translation parallel to the axis perpendicular to the selected plane and rotational axis motion). If `rotation` is 1, more than 0 but not more than 360 degrees of arc should be made. In general, if `rotation` is n, n is not 0, and we let N be the absolute value of n, the absolute value of the amount of rotation in the arc should be more than ([N-1] x 360) but not more than (N x 360).

The radius of the helix is determined by the distance from the current position to the axis of helix or by the distance from the end location to the axis of the helix. It is an error if the two radii are not the same (within some tolerance, to be set by the implementation).

The feed rate applies to the distance traveled along the helix. This differs from many existing systems, which apply the feed rate to the distance traveled by a point on a circle which is the projection of the helix on a plane perpendicular to the axis of the helix.

Rotational axis motion along with helical XYZ motion has no known applications, but is not illegal. Rotational axis motion is handled as follows, if there is rotational motion.

1. If the feed reference mode is CANON_XYZ: Perform XYZ motion as if no rotational motion were specified. While the XYZ motion is going on, move the rotational axes in coordinated linear motion.
2. If the feed reference mode is CANON_WORKPIECE: the path to follow is the path that would be followed if the feed reference mode were CANON_XYZ, but the rate along that path should be kept constant at the programmed feed rate. This will usually cause variable rates along all moving axes.

3.6.2 `DWELL (double seconds)`

Do not move the axes for the time specified by the `seconds` argument, which must be positive.

3.6.3 `ELLIPSE_FEED (double major, double minor, double angle_to_first, double first_end, double second_end, double first_axis, double second_axis, int rotation,`

`double axis_end_point, double a, double b, double c)`

Move in an elliptical helical arc from the current position at the existing feed rate. The axis of the helix is parallel to the X, Y, or Z axis, according to which one is perpendicular to the selected plane. The elliptical helical arc may degenerate to an elliptical arc if there is no motion parallel to the axis of the helix.

The length of the major axis (not the semi-major axis) of the ellipse is given by `major`, and the length of the minor axis by `minor`.

If the selected plane is the XY plane:

1. `angle_to_first` is the angle between the major axis of the ellipse and the +X axis.
2. `first_end` is the X coordinate of the end of the arc.
3. `second_end` is the Y coordinate of the end of the arc.
4. `first_axis` is the X coordinate of the axis (center) of the arc.
5. `second_axis` is the Y coordinate of the axis (center) of the arc.
6. `axis_end_point` is the Z coordinate of the end of the arc.

If the selected plane is the YZ plane:

1. `angle_to_first` is the angle between the major axis of the ellipse and the +Y axis.
2. `first_end` is the Y coordinate of the end of the arc.
3. `second_end` is the Z coordinate of the end of the arc.
4. `first_axis` is the Y coordinate of the axis (center) of the arc.
5. `second_axis` is the Z coordinate of the axis (center) of the arc.
6. `axis_end_point` is the X coordinate of the end of the arc.

If the selected plane is the XZ plane:

1. `angle_to_first` is the angle between the major axis of the ellipse and the +Z axis.
2. `first_end` is the Z coordinate of the end of the arc.
3. `second_end` is the X coordinate of the end of the arc.
4. `first_axis` is the Z coordinate of the axis (center) of the arc.
5. `second_axis` is the X coordinate of the axis (center) of the arc.
6. `axis_end_point` is the Y coordinate of the end of the arc.

If rotation is positive, the arc is traversed counterclockwise as viewed from the positive end of the coordinate axis perpendicular to the currently selected plane. If rotation is negative, the arc is traversed clockwise. If rotation is 0, `first_end` and `second_end` must be the same as the corresponding coordinates of the current position and no arc is made (but there may be translation parallel to the axis perpendicular to the selected plane and rotational axis motion). If `rotation` is 1, more than 0 but not more than 360 degrees of arc should be made. In general, if `rotation` is n, n is not 0, and we let N be the absolute value of n, the absolute value of the amount of rotation in the arc should be more than ([N-1] x 360) but not more than (N x 360).

Rotational axis motion along with elliptical helix XYZ motion has no known applications, but is not illegal. Rotational axis motion is handled as follows, if there is rotational motion.

1. If the feed reference mode is CANON_XYZ: Perform XYZ motion as if no rotational motion were specified. While the XYZ motion is going on, move the rotational axes in coordinated linear motion.

2. If the feed reference mode is CANON_WORKPIECE: the path to follow is the path that would be followed if the feed reference mode were CANON_XYZ, but the rate along that path should be kept constant at the programmed feed rate. This will usually cause variable rates along all moving axes.

The ellipse_feed command is much like the arc_feed command, except that the tool path is an elliptical helix rather than a circular helix. With an elliptical helix, there are at least two choices for how to coordinate motion parallel to the axis of the helix with motion around the axis. In both cases, we focus on a point *P* traveling along the elliptical helix. The "projected ellipse" of the helix is the projection of the helix on any plane perpendicular to the axis of the helix (which is, by definition, an ellipse). We define *P'* to be the projection of *P* on the projected ellipse and *P"* to be the projection of *P* on the axis.

1. The slope along the helix is constant. In other words, there is some constant *k*, such that for any distance *d* traveled by *P'* around the projected ellipse, the distance traveled along the axis by *P"* is *kd*.
2. The amount of travel along the axis by *P"* is proportional to the angle swept by a line from the center of the projected ellipse to *P'*. This rule produces a variable slope which is steeper on the pointy ends of the ellipse.

We have implemented the second of these — because that is what the motion control board we were using would do. It might be useful to add a setting for the machine or another argument to the command so that either of the two trajectory types could be selected.

The feed rate applies to the distance travelled along the helix by *P*. This differs from the one existing system we have used, which applies the feed rate to the distance traveled by a point on a circle which is a projection of the projected ellipse.

It is an error if the projections of the current point and the end point of the arc do not lie on the projected ellipse (within some tolerance, to be set by the implementation).

The functionality of the arc_feed command is the same as what the ellipse_feed command will do if the major and minor axes of the ellipse are equal. In that case, the value of the angle of the X axis with the major axis of the ellipse is irrelevant, and the two trajectory rules give the same trajectory. It is useful to have both commands, however, because many applications may wish to implement only arc_feed, and because the arc_feed command is simpler.

3.6.4 `STOP ()`

Stop axis motion. Regardless of the motion mode currently in use, come briefly to a stop at the end point of the last programmed move before going on to the next move.

3.6.5 `STRAIGHT_FEED (double x, double y, double z, double a, double b, double c)`

If there is no rotational motion, move the controlled point in a straight line at feed rate from the current position to the point given by the `x`, `y`, and `z` arguments. Do not move the rotational axes.

If there is rotational motion:

1. If the feed reference mode is CANON_XYZ, perform XYZ motion as if there were no rotational motion. While the XYZ motion is going on, move the rotational axes in

coordinated linear motion.

2. If the feed reference mode is CANON_WORKPIECE, the path to follow is the path that would be followed if the feed reference mode were CANON_XYZ, but the rate along that path should be kept constant.

3.6.6 `STRAIGHT_PROBE (double x, double y, double z, double a, double b, double c);`

This performs a probing operation. A probe must be in the spindle. The probe may be a touch probe or a non-contact probe. There must be no rotational motion. The probe must not be already tripped when execution of this command starts. The spindle must not be turning. The probe must be turned on. It is an error if any of those “musts” does not hold.

To execute this command, start by moving the probe in a straight line at the currently programmed feed rate from the current position toward the programmed XYZ position. The next steps depend on what type of probe is being used.

For a touch probe, it is expected that the probe will be tripped before the programmed position is reached. It is an error for the probe to reach that position without being tripped. If the probe is tripped before the programmed position is reached, the control should stop moving the axes as quickly as possible without damaging the machine, move the axes back to where they were when the probe tripped, and stop at that point.

For a non-contact probe, it is expected that the probe will reach the programmed position. It is an error if not. The probe should either decelerate and stop at that point (gathering data all the way), or continue at full feed rate through the point (gathering data all the way), stop gathering data and stop moving as fast as feasible, then move back to the programmed position and stop.

An implementation may put limits on the acceptable range of feed rates for probing, as strict as requiring a specific value. It is an error to execute a straight_probe command if the feed rate is not within the allowed range.

It is expected that appropriate interfaces will be constructed for getting and using the probe data, but those interfaces are outside the scope of this report.

More sophisticated types of probe moves may be desirable so that faster probing is feasible, moves which do not require the probe to be stopped at the trip point.

3.7 Spindle Functions

3.7.1 `ORIENT_SPINDLE (double orientation, CANON_DIRECTION direction)`

Turn the spindle (if necessary) in the direction specified by `direction` to the angle specified by the `orientation` in degrees. Stop the spindle at that angle. Acceptable values of `direction` are CANON_DIRECTION_CLOCKWISE and CANON_DIRECTION_COUNTERCLOCKWISE. `Orientation` must be between 0 and 359.999. Do not make a full turn or more.

For all machine configurations in which it is applicable, if the spindle axis is parallel to the Z axis, the zero point of spindle rotation is along the positive X axis.

3.7.2 SET_SPINDLE_SPEED (double speed)

Set to `speed` the spindle speed that will be used when the spindle is turning. `Speed` is given in rpm and refers to the rate of spindle rotation. `Speed` must be positive. If the spindle is already turning and is at a different speed, change to the speed given with this command. This command does not start the spindle if it is not turning.

An implementation should set an upper limit on spindle speed. It is an error if `speed` is larger than that limit.

3.7.3 SPINDLE_RETRACT ()

Retract the spindle at the current feed rate to the fully retracted position.

3.7.4 SPINDLE_RETRACT_TRAVERSE ()

Retract the spindle at traverse rate to the fully retracted position.

3.7.5 START_SPINDLE_CLOCKWISE ()

Turn the spindle clockwise at the currently set speed rate. If the spindle is already turning that way, this command has no effect. If the spindle speed is set outside the allowable range, it is an error to execute this command.

3.7.6 START_SPINDLE_COUNTERCLOCKWISE ()

Turn the spindle counterclockwise at the currently set speed rate. If the spindle is already turning that way, this command has no effect. If the spindle speed is set outside the allowable range, it is an error to execute this command.

3.7.7 STOP_SPINDLE_TURNING ()

Stop the spindle from turning. If the spindle is already stopped, this command has no effect.

3.7.8 USE_NO_SPINDLE_FORCE ()

Do not consider spindle force. Instead, use the feed rate to determine spindle motion.

3.7.9 USE_SPINDLE_FORCE (double force)

If the force on the spindle exceeds `force` (in newtons), reduce the feed rate until the force on the spindle drops below that amount. Otherwise, use the programmed feed rate.

This is intended for adaptive machining. It requires that the machine have a mechanism to measure the force. It is an error to execute this command if the machine does not have such a mechanism.

The meaning of this command might be refined. Different components of the force might be considered. Torque on the spindle might be used instead of force; it is more readily measured and may be more useful. The spindle speed might be changed as well as the feed rate.

3.8 Tool Functions

3.8.1 CHANGE_TOOL (int slot)

The slots of a tool changer must be numbered consecutively from 1 to however many slots there

are in the changer. It is an error if `slot` is not the number of an actual changer slot.

This command results in the tool currently in the spindle (if any) being returned to its slot, and the tool from the slot designated by `slot` (if any) being inserted in the spindle.

If there is no tool in the slot designated by `slot`, there will be no tool in the spindle after this command is executed and no error condition will result in the controller. Similarly, if there is no tool in the spindle when this command is executed, no tool will be returned to the carousel and no error condition will result in the controller, whether or not a tool was previously selected in the program. On the machines we have used, these actions do not harm the machine. If the machine design is such that these actions will harm the machine, we suggest that the superior controller check for potential harm before giving the CHANGE_TOOL command.

It is expected that when the machine tool controller is initialized, the designated slot for a tool already in the spindle will be established. This may be done in any manner deemed fit, including (for, example) recording that information in a persistent, crash-proof location so it is always available from the last time the machine was run, or having the operator enter it. It is expected that the machine tool controller will remember that information as long as it is not re-initialized; in particular, it will be remembered between programs.

For the purposes of this command, the tool includes the tool holder.

For machines which can carry out a SELECT_TOOL command separately from a CHANGE_TOOL command, the SELECT_TOOL command must have been executed before the CHANGE_TOOL command, and the value of `slot` in the CHANGE_TOOL command must be the slot number of the selected tool.

If the spindle is turning before a tool change, the spindle should be stopped by the CHANGE_TOOL command and should not be restarted.

During a tool change, the machine may move the axes, but when the tool change is complete, the axes should all be back to where they were before the tool change began. If the new tool is a different length than the old one, the tool tip will be in a different location.

If the spindle was oriented before a CHANGE_TOOL command, changing the tool may disorient it.

A tool change has no effect on coolant use. Coolant must be turned on and off by a coolant command.

3.8.2 `SELECT_TOOL (int slot)`

Select the tool in the given `slot`. It is an error if `slot` is not the number of an actual changer slot. If it is possible and efficient to do so, move the tool carousel so that the selected slot is in position for access by the tool changer.

If the changer mechanism can handle only one tool at a time, moving the carousel when this command is executed is not usually an efficient thing to do, since the tool in the spindle must usually be removed and put into its slot first when the CHANGE_TOOL command is executed. If the changer mechanism can handle two tools simultaneously, moving the carousel is usually an efficient thing to do.

3.8.3 USE_TOOL_LENGTH_OFFSET (double length)

Set the tool length offset to the given `length` (in current length units). The `length` must be non-negative. The effective value of a tool length offset changes if length units are changed, as discussed in Section 3.3.3.

3.9 Miscellaneous Functions

3.9.1 CLAMP_AXIS (CANON_AXIS axis)

Clamp the given `axis`. Acceptable values of `axis` are CANON_AXIS_X, CANON_AXIS_Y, CANON_AXIS_Z, CANON_AXIS_A, CANON_AXIS_B, and CANON_AXIS_C, provided the machining center has such an axis and that axis can be clamped. It is an error if the machining center does not have a clamp for the given `axis`.

It is an error to execute a command which would move an axis while the axis is clamped.

An alternative to having this command and the unclamp_axis command would be to have commands for turning automatic axis clamping on and off. If automatic axis clamping were on for an axis, the axis would automatically be clamped when it was not being moved and would automatically be unclamped when any command that would move it was executed.

3.9.2 COMMENT (char * text)

This command has no physical effect. If commands are being printed or logged, the comment command is printed or logged, including the string which is the value of `text`.

3.9.3 DISABLE_FEED_OVERRIDE ()

Do not pay attention to the setting of the feed override switch. Make the feed rate have its programmed value immediately.

3.9.4 DISABLE_SPEED_OVERRIDE ()

Do not pay attention to the setting of the speed override switch. Make the spindle speed have its programmed value immediately.

3.9.5 ENABLE_FEED_OVERRIDE ()

Pay attention to the setting of the feed override switch. Modify feeds in accordance with the switch settings.

3.9.6 ENABLE_SPEED_OVERRIDE ()

Pay attention to the setting of the speed override switch. Modify spindle speeds in accordance with the switch settings.

3.9.7 FLOOD_OFF ()

Turn flood coolant off. It is an error to execute this command if the machine does not have flood coolant.

3.9.8 FLOOD_ON ()

Turn flood coolant on. It is an error to execute this command if the machine does not have flood coolant.

3.9.9 MESSAGE (char * text)

Display the `text` to the machine operator on the operator console.

3.9.10 MIST_OFF()

Turn mist coolant off. It is an error to execute this command if the machine does not have mist coolant.

3.9.11 MIST_ON ()

Turn mist coolant on. It is an error to execute this command if the machine does not have mist coolant.

3.9.12 PALLET_SHUTTLE()

If the machining center has a pallet shuttle mechanism (a mechanism which switches the position of two pallets), this command should cause that switch to be made. If either or both of the pallets are missing, this will not result in an error condition in the controller. It is an error to execute this command if the machine does not have a pallet shuttle.

3.9.13 THROUGH_TOOL_OFF ()

Turn through-tool coolant off. It is an error to execute this command if the machine does not have through-tool coolant.

3.9.14 THROUGH_TOOL_ON ()

Turn through-tool coolant on. It is an error to execute this command if the machine does not have through-tool coolant.

3.9.15 TURN_PROBE_OFF ()

Turn the probe off. It is an error to execute this command if the tool in the spindle is not a probe.

3.9.16 TURN_PROBE_ON ()

Turn the probe on. It is an error to execute this command if the tool in the spindle is not a probe.

3.9.17 UNCLAMP_AXIS (CANON_AXIS axis)

Unclamp the given `axis`. It is an error if the machining center does not have a clamp for the given `axis`.

References

[Albus] Albus, James S; et al; *NIST Support to the Next Generation Controller Program: 1991 Final Technical Report*; NISTIR 4888; National Institute of Standards and Technology, Gaithersburg, MD; July 1992

[Allen-Bradley] Allen-Bradley; *RS274/NGC for the Low End Controller*; First Draft; Allen-Bradley; August 1992

[EIA] Electronic Industries Association; *EIA Standard EIA-274-D Interchangeable Variable Block Data Format for Positioning, Contouring, and Contouring/ Positioning Numerically Controlled Machines*; Electronic Industries Association; Washington, DC; February 1979

[Kramer1] Kramer, Thomas R.; Proctor, Frederick M.; Michaloski, John L.; *The NIST RS274/NGC Interpreter, Version 1*; NISTIR 5416; National Institute of Standards and Technology, Gaithersburg, MD; April 1994

[Kramer2] Kramer, Thomas R.; Proctor, Frederick M.; *The NIST RS274KT Interpreter*; NISTIR 5738; National Institute of Standards and Technology, Gaithersburg, MD; October 1995

[Kramer3] Kramer, Thomas R.; Proctor, Frederick M.; *The NIST RS274/NGC Interpreter; Version 2*; NISTIR 5739; National Institute of Standards and Technology, Gaithersburg, MD; October 1995

[Kramer4] Kramer, Thomas R.; Proctor, Frederick M.; *The NIST RS274/VGER Interpreter*; NISTIR 5754; National Institute of Standards and Technology, Gaithersburg, MD; November 1995

[K&T] Kearney and Trecker Co.; *Part Programming and Operating Manual, KT/CNC Control, Type C*; Pub 687D; Kearney and Trecker Corp.; 1980

[Monarch] Monarch Cortland; *Programming Manual for Monarch VMC-75 with General Electric 2000MC Controls*; Monarch Cortland Publication Number PRG GE2000MC-4

[NCMS] National Center for Manufacturing Sciences; *The Next Generation Controller Part Programming Functional Specification (RS-274/NGC)*; Draft; NCMS; August 1994

[Proctor] Proctor, Frederick M; et al; *Simulation and Implementation of an Open Architecture Controller*; Proceedings of the SPIE International Symposium on Intelligent Systems and Advanced Manufacturing; Philadelphia, PA; 1995

Appendix A Issues Regarding Canonical Machining Commands

The canonical machining commands presented here should not be regarded as a finished product. Further work remains to be done on them. This appendix discusses issues to be considered in further refinement of the canonical machining commands.

A.1 How Generic

The canonical machining commands comprise an application programming interface (API). It is desirable in any API to make the commands as broadly applicable as possible, so that the same API can be used to control a wide variety of machines. The canonical machining commands are suitable for machining centers but not for other types of machines.

This report has described the canonical machining commands as they apply to typical 3-axis to 6-axis machining centers which conform to the model given in Section 2.2. If the model is changed, one or several of the commands may no longer be suitable.

For machining centers with radically different geometry, such as those using a Stewart Platform, it will be desirable to decouple the definition of canonical machining commands from the machine geometry and focus on the interaction between the workpiece and the cutting tool. The 3-axis reduction of the commands in this report just described will be applicable to almost any machine geometry because they comprise a method of working in Euclidean 3-space (the one we live in).

A.2 Program Functions

The canonical machining commands do not include the notion of a program. Earlier versions included three commands for dealing with programs: optional_program_stop, program_end, and program_stop. We found that these were not necessary because stopping at the appropriate time could be accomplished at the control level above the level executing canonical machining commands by simply not giving any more commands. Hence, the program commands were deleted.

A.3 Arc format

The format of the ARC_FEED command defined in Section 3.6.1 specifies the in-plane shape of an arc by giving its end point, its center and the number of full or partial turns. Call this the end-center-turns format. This format has several desirable features, as indicated in Table 1:

Two other formats of describing the in-plane shape of an arc are in common use in RS274, where they are used with the G2 and G3 codes:

1. End-center-direction: Give the arc end point and the center of the arc and specify whether the arc is to be made clockwise or counterclockwise.
2. End-radius-direction: Give the arc end point and the radius of the arc and specify whether the arc is to be made clockwise or counterclockwise. Provide a flag that specifies whether the arc is less than or equal to a semicircle or greater than or equal to a semicircle (usually done by signing the radius).

These other formats can only produce an arc of less than 360 degrees. Both use separate commands for clockwise and counterclockwise arcs. Both have sets of values which are impossible (if the end point is farther from the center than the current position in the end-center-

direction format, and if the end point is farther from the current position than twice the radius in the end-radius-direction format). The end-center-direction format is redundant because the radius can be calculated two ways, requiring a rule for how close they must be to be considered equal. The end-center-direction format is ambiguous if the end point is the same as the current position; does this mean make a full circle, or do not move at all? An extra rule is needed to deal with this ambiguity.

The advantages of each of the three formats and a fourth format, the “center-turn” format, formerly used in the canonical machining commands, are summarized in the following table, where an X indicates an advantage. In the center-turn format, the center of the arc and the number of degrees of turn are specified, but the end point and the radius are not given.

Table 1. Arc Format Advantages

	center-turn	end-center-direction	end-radius-direction	end-center-turns
center provided	**X**	**X**		**X**
radius provided			**X**	
end-point provided		**X**	**X**	**X**
amount of turn provided	**X**			
not redundant	**X**		**X**	
unlimited arc size	**X**			**X**
all argument values legal	**X**			
no flag needed	**X**	**X**		**X**
no rule needed if end = current	**X**		**X**	**X**
same command CW or CCW	**X**			**X**
no error buildup		**X**	**X**	**X**
numerically stable		**X**		**X**

Two other aspects of arc commands are numerical stability and error buildup. If a small change in any argument of a format can produce a change in the position and shape of an arc which is much larger, we will say the format is numerically unstable; if not, we will say the format is numerically stable. If the errors from many successive commands of the same type can accumulate, we will say the command allows error buildup.

The center-turn format has several positive features, but it allows error buildup because if many arcs are made successively, the errors in the end point can accumulate. The other three formats do not allow error buildup.

The end-radius-direction format is numerically unstable when the amount of turn of the arc is near a semicircle. For example, if the current position is at the origin, and an arc of radius 1.0 is made

to the point (2.0, 0.0), this is a semicircle with its center on the X axis. If an arc of the same radius is made from the origin to the point (1.9999, 0.0), the middle of the arc moves 0.01 compared with its position on the semicircle, an amount 100 times as large as the change in the end point. This format is wildly unstable when the amount of turn is near a circle. For example, if a nearly circular clockwise arc of radius one is made from the origin to (0.0002, 0.0), the center is at (0.0001, 1.0). If an arc of the same radius is made from the origin to (0.0001, 0.0001), the center is at (-0.7071, 0.7072); the circle is in a totally different place. These instabilities have nothing to do with rounding error. The same shifts will be observed if the calculations are done to 15 decimal places.

The center-turn format is numerically unstable when the radius is large because the length of an arc equals the product of the radius and the amount of turn. Thus, for example, if the radius is 1000 and the turn changes by 0.0001, the position of the end of the arc changes by 0.1.

The problems of numerical instability and error buildup outweigh the other advantages of the center-turn format.

A.4 Rotational Axis Position

In this report, we have used wrapped linear axes for describing rotation about the rotational axes. In the wrapped linear axis format, whenever the tool turns *x* degrees counterclockwise around the workpiece, the axis position is increased by *x* (and clockwise decreases the axis position by *x*).

There is an alternative representation for turning a rotational axis, the position-direction format. In the position-direction format, the final position is always between 0.0 and 359.999 and an "axis_turn" number tells how far to turn to get there. Axis_turn is an integer representing the number of full or partial rotations of the axis. Zero means "don't move the axis." One means "turn counterclockwise more than 0 but not more than 360 degrees." Two means "turn counterclockwise more than 360 but not more than 720 degrees." Minus one means "turn clockwise more than 0 but not more than 360 degrees," and so on.

We have used the position-direction format extensively. In fact, we used it in all of the machine tool controllers we have built. On the basis of experience, we can say unequivocally that the wrapped linear axis format is superior (we often use naughty words when discussing the position-direction format).

Conceptually, the wrapped linear axis format is most suitable if the rotational axis actually wraps up and has to be unwrapped if it goes too far in either direction. If the rotational axis does not actually wrap up, there is no reason why the axis cannot turn to an arbitrarily large position in the wrapped axis format. In this case, the position-direction format matches the physical situation more closely. It is still easy to use the wrapped linear axis format for an unlimited rotational axis. The position reading just needs to be reset to be between 0 and 360 occasionally (between programs, for example).

A.5 Control Parameters

It may be useful to add canonical commands for fine control of acceleration, deceleration, and tolerances. This should make it possible to improve uniformity of execution from machine to machine. For example, a SET_ACCELERATION_RATE command might be defined. There might be different rates for different axes. For each machine model, there would be a maximum

allowable acceleration rate, depending on the kinematics and dynamics of the particular model. The allowable degree of deviation between path segments that are not tangential should also be specified.

A.6 Allocation of Functionality to Hierarchical Levels

In hierarchical control systems, there is often no compelling reason to put a particular functionality at a specific hierarchical level. In some cases, the same functionality may reasonably be placed in either of two adjacent levels.

Functionality which might be performed either by the controller executing canonical machining commands or the superior of that controller includes:

change of length units (currently canonical level),
change of feed rate mode (currently superior level),
change of feed reference mode (currently canonical level),
execution of canned cycles (currently superior level),
cutter radius compensation (currently superior level),
using offset origin (currently canonical level),
inverse time feed rate (currently superior level),
enabling or disabling automatic axis clamping (currently superior level), and
clamping or unclamping an axis (currently canonical level).

It is not clear how decisions on allocating these functionalities should be made. In the EMC project, the rule of thumb that canonical commands should map closely to functionality provided by existing commercial motion control boards has been used.

A.7 NURBS and Parametric Axis Control

Further consideration should be given to defining a NURBS (Non-Uniform Rational B-Spline) command on either the canonical level or at the level above, in RS274. The authors have already implemented such a command experimentally at the canonical level. The decision about which level is appropriate hinges on whether commercial motion control boards provide the capability to execute NURBS motion. If they do, the canonical level is appropriate, if not, the level above.

Similarly, parametric axis feed should be provided at some level. This is probably less suitable than NURBS for the canonical level, because the capability to evaluate parametric expressions will be required. In EMC, such capability is provided already in two of the three RS274 interpreters.

A.8 Cutter Radius Compensation

Cutter radius compensation has not been included in the canonical machining commands. It might be included. We believe it is better to do cutter radius compensation at the control level which is making calls to the canonical machining commands. This is because there are many different ways to perform cutter radius compensation, most of which put too much of a computational burden on the control level at which the canonical commands are aimed. These generally require that the controller perform look-ahead (which the controller usually does anyway) and make calculations involving every line (which are not otherwise required).

If cutter radius compensation were to be added to the canonical machining commands, it should

probably use a method which does not require look-ahead.

Our implementation of cutter radius compensation in the control level above the canonical machining commands is discussed on pages 42 - 51 of [Kramer4].

Canonical commands for cutter radius compensation could be as follows.

`SET_CUTTER_RADIUS_COMPENSATION (double radius)`

Set to `radius` the radius to use when performing cutter radius compensation. The `radius` must be positive. The effective cutter radius changes if length units are changed.

`START_CUTTER_RADIUS_COMPENSATION (CANON_COMP_DIRECTION direction)`

This starts cutter radius compensation. Acceptable values of `direction` are CANON_COMP_LEFT and CANON_COMP_RIGHT, where left means the cutter stays to the left of the programmed path (when viewed from the positive end of the axis perpendicular to the currently selected plane) as the cutter moves forward and right means the cutter stays to the right of the programmed path.

`STOP_CUTTER_RADIUS_COMPENSATION ()`

Do not apply cutter radius compensation when executing spindle translation commands.

Appendix B Header File

```
#ifndef CANON_HH

#define CANON_HH

/*

This is a C or C++ header file for canonical commands for 3-axis to 6-axis machining. The X, Y, and Z linear orthogonal axes are always used. The A, B, and C rotational axes (parallel to the X, Y, and Z axes, respectively) are conditional on the definition of A_AXIS, B_AXIS, and C_AXIS, respectively.
```

begin typedefs

```
Each required type is defined here using a “typedef” and two or more “#defines”. Alternatively, the same types might be defined with the same names using one “enum” for each.

*/

typedef int CANON_PLANE;
#define CANON_PLANE_XY 1
#define CANON_PLANE_YZ 2
#define CANON_PLANE_XZ 3

typedef int CANON_UNITS;
#define CANON_UNITS_INCHES 1
#define CANON_UNITS_MM 2
#define CANON_UNITS_CM 3

typedef int CANON_MOTION_MODE;
#define CANON_EXACT_STOP 1
#define CANON_EXACT_PATH 2
#define CANON_CONTINUOUS 3

typedef int CANON_SPEED_FEED_MODE; /* used in tracking state, not in commands */
#define CANON_SYNCHED 1
#define CANON_INDEPENDENT 2

typedef int CANON_DIRECTION;
#define CANON_DIRECTION_STOPPED 1 /* used in tracking state, not in commands */
#define CANON_DIRECTION_CLOCKWISE 2
#define CANON_DIRECTION_COUNTERCLOCKWISE 3

typedef int CANON_FEED_REFERENCE;
#define CANON_WORKPIECE 1
#define CANON_XYZ 2

typedef int CANON_AXIS;
#define CANON_AXIS_X 1
#define CANON_AXIS_Y 2
#define CANON_AXIS_Z 3
#ifdef A_AXIS #define CANON_AXIS_A 4 #endif
```

```
#ifdef B_AXIS #define CANON_AXIS_B 5 #endif
#ifdef C_AXIS #define CANON_AXIS_C 6 #endif

/*
```

end typedefs

begin function prototypes

Throughout these function prototypes, arguments named x, y, z, a, b, and c represent values on the respective axes. All functions prototyped here return void. Functions are arranged by type and alphabetically within type.

```
*/

/************************************************************************/

/* Initialization and termination*/

extern void INIT_CANON();

extern void END_CANON();

/************************************************************************/

/* Representation */

extern void SELECT_PLANE(CANON_PLANE plane);

extern void SET_ORIGIN_OFFSETS(
  double x,
  double y,
  double z
#ifdef A_AXIS , double a #endif
#ifdef B_AXIS , double b #endif
#ifdef C_AXIS , double c #endif
);

extern void USE_LENGTH_UNITS(CANON_UNITS units);

/************************************************************************/

/* Free Space Motion */

extern void SET_TRAVERSE_RATE(double rate);

extern void STRAIGHT_TRAVERSE(
  double x,
  double y,
  double z
#ifdef A_AXIS , double a #endif
#ifdef B_AXIS , double b #endif
#ifdef C_AXIS , double c #endif
);

/************************************************************************/
```

```
/* Machining Attributes */

extern void SET_FEED_RATE(double rate);

extern void SET_FEED_REFERENCE(CANON_FEED_REFERENCE reference);

extern void SET_MOTION_CONTROL_MODE(CANON_MOTION_MODE mode);

extern void START_SPEED_FEED_SYNCH();

extern void STOP_SPEED_FEED_SYNCH();

/************************************************************************/

/* Machining Functions */

extern void ARC_FEED(
  double first_end,
  double second_end,
  double first_axis,
  double second_axis,
  int rotation,
  double axis_end_point
#ifdef A_AXIS , double a #endif
#ifdef B_AXIS , double b #endif
#ifdef C_AXIS , double c #endif
);

extern void DWELL(double seconds);

extern void ELLIPSE_FEED(
  double major,
  double minor,
  double angle_to_first,
  double first_end,
  double second_end,
  double first_axis,
  double second_axis,
  int rotation,
  double axis_end_point
#ifdef A_AXIS , double a #endif
#ifdef B_AXIS , double b #endif
#ifdef C_AXIS , double c #endif
);

extern void STOP();

extern void STRAIGHT_FEED(
  double x,
  double y,
  double z
#ifdef A_AXIS , double a #endif
#ifdef B_AXIS , double b #endif
```

```
#ifdef C_AXIS , double c #endif
);
extern void STRAIGHT_PROBE(
  double x,
  double y,
  double z
#ifdef A_AXIS , double a #endif
#ifdef B_AXIS , double b #endif
#ifdef C_AXIS , double c #endif
);
/************************************************************************/
/* Spindle Functions */
extern void ORIENT_SPINDLE(double orientation, CANON_DIRECTION direction);
extern void SET_SPINDLE_SPEED(double speed);
extern void SPINDLE_RETRACT();
extern void SPINDLE_RETRACT_TRAVERSE();
extern void START_SPINDLE_CLOCKWISE();
extern void START_SPINDLE_COUNTERCLOCKWISE();
extern void STOP_SPINDLE_TURNING();
extern void USE_NO_SPINDLE_FORCE();
extern void USE_SPINDLE_FORCE(double force);
/************************************************************************/
/* Tool Functions */
extern void CHANGE_TOOL(int slot);
extern void SELECT_TOOL(int slot);
extern void USE_TOOL_LENGTH_OFFSET(double length);
/************************************************************************/
/* Miscellaneous Functions */
extern void CLAMP_AXIS(CANON_AXIS axis);
extern void COMMENT(char * text);
extern void DISABLE_FEED_OVERRIDE();
extern void DISABLE_SPEED_OVERRIDE();
extern void ENABLE_FEED_OVERRIDE();
extern void ENABLE_SPEED_OVERRIDE();
extern void FLOOD_OFF();
```

```
extern void FLOOD_ON();
extern void MESSAGE(char * text);
extern void MIST_OFF();
extern void MIST_ON();
extern void PALLET_SHUTTLE();
extern void THROUGH_TOOL_OFF();
extern void THROUGH_TOOL_ON();
extern void TURN_PROBE_OFF();
extern void TURN_PROBE_ON();
extern void UNCLAMP_AXIS(CANON_AXIS axis);
/************************************************************************/
#endif
```