NBSIR 88-3704

THE DATA EXECUTION MODULE OF THE VERTICAL WORKSTATION

January 6, 1988

By:
Thomas R. Kramer
Rebecca E. Weaver

THE DATA EXECUTION MODULE OF THE VERTICAL WORKSTATION OF THE AUTOMATED MANUFACTURING RESEARCH FACILITY AT THE NATIONAL BUREAU OF STANDARDS

Dr. Thomas R. Kramer
Guest Worker, National Bureau of Standards, &
Research Associate, Catholic University

Ms. Rebecca E. Weaver
Summer Intern, National Bureau of Standards

January 6, 1988

Funding for the research performed by Dr. Kramer and reported in this paper was provided to Catholic University under Grant No. 60NANB5D0522 and Grant No. 70NANB7H0716 from the National Bureau of Standards.

Certain commercial equipment and software are identified in this paper in order to adequately specify the experimental facility. Such identification does not imply recommendation or endorsement by the National Bureau of Standards, nor does it imply that the equipment and software identified are necessarily the best available for the purpose.

CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

THE DATA EXECUTION MODULE OF THE VERTICAL WORKSTATION OF THE AUTOMATED MANUFACTURING RESEARCH FACILITY AT THE NATIONAL BUREAU OF STANDARDS

I. INTRODUCTION

1. CONTENTS

This paper discusses the Data Execution module of the Vertical Workstation (VWS) of the Automated Manufacturing Research Facility (AMRF) at the National Bureau of Standards. The Data Execution module is where numerical control code (NC-code) for the workstation's vertical milling machine is prepared. The descriptions pertain to the system in use during the summer of 1987.

Chapter II tells what the module does in its three phases of action: initialization, stepping through the process plan and closing.

Chapter III discusses the workpiece model used by the module: how it is initialized, kept up to date as steps of the process plan are executed, and used.

Chapter IV discusses machining metal and describes the machining operations used in the system.

Chapter V discusses automatic NC-coding in the VWS2 system: what NC-code looks like, how NC-code is interpreted, the general approach to NC-coding taken by the module, and what the machining algorithms are which have been implemented.

Chapter VI discusses the LISP software which comprises the module.

2. AUDIENCE

The paper is intended to be useful to people interested in concepts and technical details of the VWS, particularly AMRF personnel who are running the VWS or maintaining or improving the software for the VWS. The paper is intended to be useful also to other researchers in automated manufacturing. Knowledge of (i) the computer language LISP, (ii) machining tools, and (iii) NC-code language is useful but not essential to reading this paper. Detailed documentation of the LISP functions that are involved with the systems described here is being prepared separately.

3. BRIEF VWS DESCRIPTION

The VWS is a computer-integrated automated machining workstation. It includes a control system, a computer-aided design system, an automatic process planning system, and an automatic NC-code generator. The principal machinery is a milling center (Monarch VMC-75 with a GE2000 controller) and a robot (Unimate 4070 with a Val II controller) to tend the milling center. There is quite a bit of ancillary hardware. The system is controlled from a microcomputer (Sun 3/160 with 6M memory, BW monitor). Running in stand-alone

mode, it is possible to design and machine a simple metal part within an hour. The VWS may also be run as an integrated part of the AMRF. The workstation is described in more detail in [K&J1].

The software for the VWS is written in the Franz LISP dialect of the computer language LISP. In this paper this software is called the VWS2 system. Six principal modules comprise the VWS2 system: the Production Management Operating System (the control system), the State Table Editor, the Equipment Program Generator, the Part Design Editor, the Process Planner, and the Data Execution module.

The Part Design Editor, Process Planning and Data Execution modules, as well as other system capabilities, may be accessed by the user through a small user interface called vws_cadm. Vws_cadm asks the user questions about what the user wants to do and then activates the appropriate module or other capability accordingly.

To produce a part from scratch, the user sits at the Sun workstation and creates a design using the Part Design Editor. The Process Planner is then called to write a plan for how to machine a part of that design. Next NC-code is generated automatically from the design and the plan by the Data Execution module. Finally the user tells the control system to make the part. The control system coordinates the activities of the workstation equipment so that the part blank is loaded onto the milling machine, the NC-code is sent to the milling machine and executed (making the part), and the finished part is unloaded.

4. DESIGN PROTOCOL

The VWS2 system uses a feature-based design protocol. The design protocol is described in detail in [K&J2]. The design of a part is expressed as a list of features on a piece of stock. The piece of stock is always a rectangular block. The design protocol currently assumes that all features are being made on one side of the block.

Although all the features and subfeatures are purely geometric, they were selected to be included in the system on the basis of being features commonly found on machined parts that could be produced in one, or at most a very few, machining operations. Each feature and subfeature is a removed volume.

The design of a part is a purely geometric description of the shape of a part and gives no idea of what machining operations are required to make the part.

The primary features in the system in September, 1987 are: chamfer_out, groove, hole, pocket, straight_groove, text, contour_groove, contour_pocket, and side_contour. There are also subfeatures which may be made on the primary features: chamfer_out, chamfer_in, countersink, and thread. A feature is specified in the system by giving its name and the values of several parameters which specify its location, shape, and size.

The design protocol includes the use of "reference features". If feature A is to be made at the bottom of feature B, then one of the parameters of feature A is "reference_feature", and the value of that parameter is the feature number of feature B. Normally, if B is the reference feature for A, the outline of feature A must fit within the outline of feature B, and the bottom of feature B must be flat (there are exceptions in special cases).

Although a design could be prepared according to the VWS2 design protocol using a text editor, the only reasonable way to make a design is by using the VWS2 Design Editor module. The Design Editor is a friendly system which runs on a Sun computer that engages the user in a dialog to find out what the user wants to make and prepares the design document for the user. An example design is shown in Table 3 (page 10).

5. PROCESS PLAN PROTOCOL

The core of a process plan which is to be executed by the Data Execution module is a set of operations, or "steps", that must be carried out in order to make some or all of the features from a design on a workpiece. In addition to a set of steps, a process plan may also have (1) a list of requirements of tools and workpieces needed to carry out the plan, (2) administrative information such as the name of the plan, the id_number of the design to which the plan is tied, the version number, etc., and (3) a list of parameters used in the plan (if the plan is parametric).

Each step in a plan describes some operation to be carried out. The description is given by naming the operation (which we will call the "work element") and giving the names and values of several parameters required to describe the operation fully. Each work element has its own set of parameters, but a given parameter type may be used for many or all types of work elements. A bare-bones step includes the work element, the number of the feature from the design to which the work element applies, the name of the type of tool needed to carry out the step, and a list of "precedent steps" which must be carried out before the step under consideration. Additional information which is added at some point usually includes a changer slot from the milling machine in which a tool of the correct type may be found, spindle speed, and feed rate.

For steps which refer to features, most or all of the geometric information needed to carry out the step is extracted from the design, and is not carried in the process plan. For steps which do not refer to features (such as zero-setting steps), geometric information is carried in the process plan.

In the VWS2 system, a process plan may have two formats: the standard AMRF format or a LISP-readable format. To be executed, a plan must be in the LISP environment in LISP-readable format. A reading facility which sets up a LISP-readable plan in the environment from a file in standard AMRF format is part of the VWS2 system, as is a facility which prints a standard file from a LISP-readable plan in the environment.

A detailed explanation of process plans for the VWS milling machine, including examples, is given in [KRA1]. An example of a LISP-readable process plan is shown in Table 2 (page 9).

6. SUMMARY OF DATA EXECUTION MODULE CAPABILITIES

The VWS2 Data Execution Module is best known as the automatic NC-code generator used in the VWS. However, it does much more than generate NC-code. The only important action the module always takes when it runs is to build a data model of a workpiece. An example of a model is shown in the bottom half of Table 1. In addition to building this model, the module has five processing and output options: (1) write NC-code, (2) enhance a process plan, (3) verify a process plan, (4) graphically emulate execution of a process plan, and (5) save the model. The five options are independent; any combination of them may be used simultaneously.

The enhancement of a process plan has several elements: adding or deleting steps of the plan, picking a specific changer slot for each tool, and calculating spindle speeds, feed rates, stepovers, and vertical pass depths, if necessary. The input process plan may already be enhanced. Any step of the input process plan may already contain values for any of: spindle speed, feed rate, stepover, or vertical pass depth.

The incoming workpiece may be a partly machined part, as long as the data model of the workpiece is correct. The same process plan may be used to finish many differently shaped partly machined workpieces, as long the features on the incoming workpieces are all given in the design which is named in the plan. This is accomplished by simply removing the steps of the plan needed to produce any feature that is already present on the incoming workpiece.

Conversely, a process plan does not need to produce all the features from a design. Thus, the VWS may produce partly machined parts. The system will produce a data set correctly representing a partly machined part.

Once the VWS2 LISP environment is set up on a Sun computer, the Data Execution module may be used via the "vws_cadm" friendly front end, or by a LISP function call to "execute_plan".

With all five options on, the Data Execution module, running in uncompiled LISP on a Sun 3/160 microcomputer with 6 megabytes of on-board memory and floating point hardware, will write NC-code at the rate of about 200 lines per minute. With only the NC-coding option on the rate roughly doubles.

7. RELATED READING

This paper is one of about a dozen papers being prepared as part of the AMRF documentation to describe all aspects of the VWS. The others are [JUN], [KRA1], [KRA3], [KRA4], [KRA5], [K&J2], [K&S2], [LOVE], [NA&J], and [RUDD]. Other papers, prepared for professional meetings, also describe the VWS [KRA2], [K&J1], and [K&S1].

The brief descriptions of the design protocol and the process plan protocol given above in sections 4 and 5 are not adequate for a detailed understanding of the Data Execution Module. The reader who wants details is referred to [K&J1] or [K&J2] for the design protocol and to [KRA1] or [KRA2] for the process plan protocol.

Some of the functions of the Data Execution module are not described in detail in this paper. In particular, process plan enhancement is dealt with in Chapter V of the process planning paper [KRA1], and seven types of verification carried out in the module are described in various chapters of the verification paper [K&S2].

II. DATA EXECUTION MODULE OPERATION

1. INTRODUCTION

The Data Execution module capabilities were summarized in section 6 of Chapter I. The operation of the module may be roughly divided into three phases: initialization, stepping through the process plan, and termination. The module may be used directly from LISP. An example of a function call to the module is:

```
(execute_plan  'datex_plan  'datex_part  'datxnc  'datex_plan_enhanced'  soft  t  'vise  'top_of_part)
```

This function call means: run the Data Execution module on the plan named "datex_plan", using the workpiece "datex_part". Write NC-code to the file "datxnc". Save the enhanced version of the plan in the file named "datex_plan_enhanced". Put verification on soft. Draw a picture emulating the machining process. Assume for the purposes of verification and NC-code writing that the part is to be milled in the vise. Establish z-zero by probing the top of the part (and, possibly, offsetting from there).

What happens when this function call is made is shown in several tables and figures throughout this chapter. The function call itself appears at the top of Table 1. That table shows the messages sent back to the user during module operation. In a test run, the module carried out the function call in just over one minute. The process plan "datex_plan" which is being executed is shown in Table 2. To illustrate the flexibility of the module, this process plan contains non-sequential precedent steps, one speed, and one pass depth. This plan is not much like the one produced by the VWS2 Process Planning module, which would use sequential precedent steps and put speeds and pass depths in either all or none of the steps for which they are appropriate.

The name of the design does not appear in the function call. The name of the design is found in the process plan. In the example the design is called "datex_design". The data used by the system giving the design and the workpiece are shown in Table 3. The design has three features in it: a countersunk hole, a chamfered pocket to the right of the hole, and the letter "D" at the bottom of the pocket. The workpiece has already been partially machined. It has the hole in it, but the hole is not yet countersunk. Figure 1 (page 17) shows a picture of the part without the letter "D". The process plan is a plan for making the hole and the pocket, but not the letter "D".

In actual use, the vws_cadm friendly front end would normally be used to run the module, rather than making a direct function call. This is a great deal easier than trying to remember what all those arguments mean and what the correct order is. In the test run, it took about 30 seconds to answer vws_cadm's questions so that it could construct the function call. Using vws_cadm has the added advantage that vws_cadm will print the description of the finished workpiece to a file if the user wishes. The direct function call results in the workpiece description being the value returned by LISP. The last half of Table 1 is the final workpiece description.

Table 1. Data Execution Screen Messages

(pp_plist (execute_plan ’datex_plan ’datex_part ’datxnc ’datex_plan_enhanced ’soft t ’vise ’top_of_part))	⟵ *INITIAL COMMAND FROM USER*
Starting design enhancement. Design enhancement completed successfully.	⟵ ENHANCE THE DESIGN
Feature 1 hole is OK. Feature 2 pocket is OK. Feature 3 text is OK.	⟵ VERIFY THE DESIGN
Starting process plan enhancement, phase 1. Process plan enhancement, phase 1, completed successfully. Starting process plan enhancement, phase 2. Process plan enhancement, phase 2, completed successfully.	⟵ ENHANCE THE PROCESS PLAN
Fixturing is OK.	⟵ CHECK FIXTURING
Step 1 initialize_plan is OK.	⟵ VERIFY STEP 1 & INITIALIZE
Feature 1 hole is OK.	⟵ VERIFY WORKPIECE
Step 2 set0_corner is OK. Step 3 set0_z is OK. Step 5 mill_pocket is OK. Step 4 machine_chamfer_in is OK. Step 6 machine_countersink is OK. Step 7 close_plan is OK.	⟵ VERIFY & EXECUTE PLAN
(workpiece features (features 1 (1 countersink_diameter 0.5 feature_type hole center_x 1 center_y 1 diameter 0.316 depth 0.6 bottom_type conical) 2 (2 chamfer_in_depth 0.06 feature_type pocket_corners upper_l_x 2 upper_l_y 2.5 lower_r_x 5 lower_r_y 0.5 depth 0.3 corner_radius 0.4)) header (header workpiece_id datex_part design_id datex_design material aluminum block_size (block_size length 6.95 width 2.975 height 0.735) description "data execution demo part"))	⟵ PRINT OUT WORKPIECE

Table 2. Process Plan

This table shows the LISP-readable unenhanced process plan file for the example in Chapter II.

```
(setplist 'datex_plan '(
  header (header
        plan_id      datex_plan
        design_id  datex_design
        material    aluminum)
  steps  (steps
        1   (1   work_element      initialize_plan
                  prog_name         "data execution demo design")
        2   (2   work_element      set0_corner
                  tool_type_id       probe_0.25
                  corner             1
                  x_offset           0.0
                  y_offset           0.0
                  near_x             16.825
                  near_y             7.425
                  precedent_steps  (1))
        3   (3   work_element      machine_chamfer_in
                  feature_id         2
                  tool_type_id       chamfer_0.375_3_abs
                  speed              5103
                  precedent_steps  (4 2))
        4   (4   work_element      mill_pocket
                  feature_id         2
                  tool_type_id       end_mill_0.625_2_ab
                  precedent_steps  (5))
        5   (5   work_element      drill_hole
                  feature_id         1
                  tool_type_id       drill_0.316_2_abs
                  pass_depth         0.3
                  precedent_steps  (2))
        6   (6   work_element      machine_countersink
                  feature_id         1
                  tool_type_id       countersink_0.75_1_ab
                  precedent_steps  (3 2 4))
        7   (7   work_element      close_plan
                  precedent_steps  (6 3)))
  tool_requirements (probe_0.25   end_mill_0.625_2_ab  drill_0.316_2_abs
                          chamfer_0.375_3_abs   countersink_0.75_1_ab)))
```

Table 3. Design and Workpiece

This table shows the LISP-readable files which set up a design and a workpiece for the example used in Chapter II.

The Design

```
(setplist 'datex_design '(
 features    (features
  1 (1
        feature_type              hole
        center_x                  1
        center_y                  1
        diameter                  0.316
        depth                     0.6
        bottom_type               conical
        countersink_diameter      0.5)
  2  (2
        feature_type              pocket_corners
        upper_l_x                 2
        upper_l_y                 2.5
        lower_r_x                 5
        lower_r_y                 0.5
        depth                     0.3
        corner_radius             0.4
        chamfer_in_depth          0.06)
  3  (3
        feature_type              text
        text                      "d"
        font                      broad
        lower_l_x                 3
        lower_l_y                 1
        height                    1
        depth                     0.02
        line_width                0.1356466
        reference_feature         2))
 header      (header
   design_id     datex_design
   material      aluminum
   block_size    (block_size
                    length      6.95
                    width       2.975
                    height      0.735)
   description "data execution demo design")))
```

The Workpiece

```
(setplist 'datex_part '(
 features    (features
  1 (1
        feature_type              hole
        center_x                  1
        center_y                  1
        diameter                  0.316
        depth                     0.6
        bottom_type               conical))
 header      (header
   workpiece_id  datex_part
   design_id     datex_design
   material      aluminum
   block_size    (block_size
                    length      6.95
                    width       2.975
                    height      0.735)
   description "data execution demo part")))
```

2. INITIALIZATION

2.1. Overview

During initialization of the execution of a process plan:

A. The design is enhanced.
B. If the verification option is on, the design is verified.
C. The process plan is enhanced (unless the input plan has been enhanced).
D. Step 1 of the plan (initialize_plan) is carried out.
E. If the drawing option is on, the initial view of the workpiece is drawn, and, if the verification option is also on, the workpiece is verified.
F. The workpiece model is set up.
G. The model of the fixturing, including obstacles, is set up.
H. If the NC-coding option is on, a working data base is set up and the first seven lines of pseudocode are generated.

The verification operations carried out during initialization are quite extensive. They are described in Chapter IX of the verification paper [K&S2].

Design enhancement is also extensive. It is described in Chapter II of the design protocol paper [K&J2].

2.2. Workpiece Model

Workpiece models are described in Chapter II, section 2.12, of [K&J2]. To make the initial model of the workpiece, the module copies the property list of the workpiece named in the function call. If the workpiece is too tall, its height is changed to the height given in the design. If the workpiece has a "slab", the slab is removed. These two things are done in conjunction with adding steps to the process plan, as described next.

When the module is used from vws_cadm, the user is asked to name a workpiece. A check is made of the property list of that name. If the property list exists and is in good format, that model is used. If there is no property list for the name, vws_cadm makes one, assuming that the workpiece is a blank block the same size as the one specified in the design.

2.3. Process Plan Enhancement

Process plan enhancement takes place in two phases. These are described in detail in Chapter V of the process planning paper [KRA1]. In this section we will follow what happens to our example. Table 2 and Table 4 show the "before and after" of enhancement.

In phase 1 of enhancement, because the user has asked to have the top of the workpiece used for establishing z-zero, a new step, set0_z, is added as step 3. Then the plan is renumbered so that there are no duplicate numbers. Precedent step numbers must be updated as part of the renumbering. In order that set0_z be carried out immediately after step 2 (which is set0_corner), the precedent step for step 3 is step 2, and any operation which has step 2 as a precedent step has step 3 added to its list of precedent steps.

If the workpiece in our example had no features and was too thick, a face milling operation would have been added to bring the workpiece down to the proper thickness. If the workpiece had had the "slab" property (see the design protocol paper [K&J2] Chapter II, section 2.12), a face milling operation to remove the slab and a set0_corner operation to find the block below the slab would have been added.

In phase 2 of enhancement, the module looks at the process plan and culls out any steps that would make features or subfeatures which are already on the workpiece. In our example, step 5 of the process plan is a hole drilling operation to make feature 1. Since feature 1 is already in the part, step 5 is removed from the process plan. After removal, any other step which had the removed step as a precedent step (step 4 in the example) has the precedent steps for the removed step added to its list of precedent steps. Thus, step 2 is added to the list of precedent steps of step 4 in our example. Then the plan is renumbered so that there are no gaps in the numbering. Precedent step numbers must be updated as part of the renumbering.

Notice that the countersinking step is not culled out because the hole in the workpiece is not yet countersunk.

Although only one step has been added and one deleted, the net effect on the step numbers and the precedent steps is significant, as may be seen by comparing these items on Table 2 with the same items from Table 4.

Also in phase 2 of enhancement, the slot number of a changer slot on the milling machine which has a tool of the tool_type_id given in each step is inserted in the step. This is done by examining the data model of the tools currently on the milling machine. If any necessary tool is not on the machine, an error message is sent and the module quits work. Also in phase 2, any step which requires a stepover, speed, feed_rate or pass_depth, has values for the appropriate parameters added. If there are existing values, such as the speed for the chamfering operation, these values are used.

Finally a new tool_requirements list is prepared from the enhanced steps. In our example, since the hole drilling step has been deleted, the drill which appears in the list of tool requirements for the unenhanced plan does not appear in the enhanced plan.

TABLE 4. Enhanced Process Plan

This table shows the LISP-readable enhanced process plan file for the example of Chapter II.

```
(setplist 'datex_plan '(
  header (header
        plan_id       datex_plan
        design_id     datex_design
        material      aluminum)
  steps  (steps
     1   (1
           work_element     initialize_plan
           prog_name
             "data execution demo design")
     2   (2
           work_element       set0_corner
           tool_type_id       probe_0.25
           changer_slot       40
           corner             1
           x_offset           0.0
           y_offset           0.0
           near_x             16.825
           near_y             7.425
           precedent_steps    (1))
      3   (3
           work_element       set0_z
           tool_type_id       probe_0.25
           changer_slot       40
           x_loc              20.3
           y_loc              8.9125
           offset             0.0
           precedent_steps   (2))
      4   (4
           work_element      machine_chamfer_in
           feature_id        2
           tool_type_id      chamfer_0.375_3_abs
           changer_slot      6
           speed             5103
           feed_rate         28
           precedent_steps (3 5 2))
      5   (5
           work_element      mill_pocket
           feature_id        2
           tool_type_id      end_mill_0.625_2_ab
           changer_slot      12
           stepover          0.3125
           speed             2750
           feed_rate         17
           pass_depth        0.3125
           precedent_steps   (3 2))
     6   (6
           work_element    machine_countersink
           feature_id      1
           tool_type_id    countersink_0.75_1_ab
           changer_slot    4
           stepover        0.375
           speed           2291
           feed_rate       5
           precedent_steps (3 4 2 5))
      7   (7
           work_element      close_plan
           precedent_steps   (6 4)))

tool_requirements (
          probe_0.25
          chamfer_0.375_3_abs
          end_mill_0.625_2_ab
          countersink_0.75_1_ab)))
```

2.4. Process Plan Sequencing

A list of step numbers giving the order in which the plan will actually be executed is constructed during initialization as follows. This sequencing is done using a working copy of the enhanced plan.

A. It is assumed that step 1, initialize_plan, will be executed first.
B. Step 1 is removed from the list of precedent steps of every step.
C. All the steps which have no remaining precedent steps are found and ordered so as to minimize tool changes. This group of step numbers is added to the sequence number list, and the steps themselves are removed from the working copy of the plan.
D. The numbers found in C are removed from the precedent steps of the steps left in the working copy of the plan.

Items C and D repeated until the group of steps found in C is empty. If there are any steps left in the working copy of the plan at this point, there must have been an error in the assignment of precedent steps in the original process plan, so an error message is returned and the module stops work. If the working copy is now empty, the sequence list is returned.

This method of sequencing was chosen to ensure that precedent requirements are followed, to check the validity of the requirements, and to give an efficient sequence. Alternatives to steps C and D which result in even fewer tool changes are feasible but more complicated; none has been tried.

As may be seen in Table 1 and Table 5, the order in our example is 1, 2, 3, 5, 4, 6, 7.

2.5. Data Base Initialization

The VWS2 system includes a "world model". This is a hierarchically arranged database of information about the workstation. When the Data Execution module starts up, the "fixturing" branch of the world model is reset by looking at the function call to see whether the vise or the pallet area is being used, and extracting the correct description from elsewhere in the world model. If the vise is being used, the description of the obstacles in the vicinity of the fixture is modified according to the size of the workpiece. This is described in detail in Chapter VI of the verification paper [K&S2].

Several large LISP property lists are set up under the following names:
A. drawp - for use by the graphics system if the drawing option is on
B. mockup - for use by execute_plan
C. mtool - for use by the NC-coding system.

3. STEPPING THROUGH THE PROCESS PLAN

3.1. Introduction

Once initialization is complete, the Data Execution module feeds the steps of the enhanced

process plan (in the order specified in the sequence list) into the execute_step function, which handles all but the init_plan and close_plan steps. This section describes the operation of the execute_step function on a single step.

First, if verification is on, the step is verified. If verification fails while verification is "on hard", or if verification fails while verification is "on soft" and the user elects not to continue, the module quits.

Second, if the NC-coding option is on, an NC-coding function is called on to write pseudocode.

Third, if the drawing option is on, and the step requires drawing, a drawing function is called.

Fourth, the workpiece model is updated. If the update fails, an error message is sent and the module quits. Note that workpiece updating is the only thing that always happens. Of course, some steps have no effect on the model, but that is not determined by execute_step itself.

3.2. Function Call Assembly

A somewhat intricate method of constructing function calls is used for verification, NC-coding and drawing. Each function call is built from four parts.

The first part is the name of the function. It is obtained from the "machine_ops" database.

The second part is the values of parameters present in the step. The names of these are obtained from the "machine_ops" database, and then their values are extracted from the step.

The third part is the values of parameters present in the feature named in the step. The names of these are obtained from the "machine_ops" database and then their values are extracted from the enhanced design.

The fourth part is the values of some local variables present in the execute_step function. The names of these are obtained from the "machine_ops" database and then the names are simply evaluated to get the current local value. The drawing system does not use this fourth part. The fourth part used by the drawing system is the number of the changer slot given in the step.

To carry out a function, parts two, three, and four are joined together into a single list, and the chosen function is applied to the list (by the LISP "apply" function).

There are 19 machine operations that may be carried out by execute_step. Each one has a verification function and an nc-coding function. Most of them have drawing functions. Thus, over 50 different functions for verification, NC-coding and drawing may be called through this function call assembly process.

4. CLOSING

4.1. Introduction

Stepping through the process plan stops with the next-to-last step on the sequenced list. The last step on the list must be close_plan or execution of the plan would have aborted earlier. If verification is on, this step is verified. If drawing is on, the drawing is remasked. If NC-coding is on the last six lines of pseudocode are written, and the pseudocode is printed as real code to a file. The features on the part model, which may have been added in random order, are sorted into numerical order. The property list of "mockup" is wiped out. The drawing is not wiped out but is left on the screen for further use. If drawing is on, a copy of the pseudocode will be given to the drawing system.

4.2. Printing Pseudocode

If execution of the enhanced process plan has continued to completion and the NC-coding option is on, pseudocode which was assembled while stepping through the plan will be printed to the file named in the function call. In our example the file is "datxnc". A copy of the file is shown in Table 5. Blocks of code in the table have been shaded and numbered with the number of the step from the enhanced process plan which caused the code to be written. Note that the order of the blocks of code is the order shown in Table 1, which was established the by the module's plan sequencer.

4.3. Use of Graphics for Flashing and Tool Path

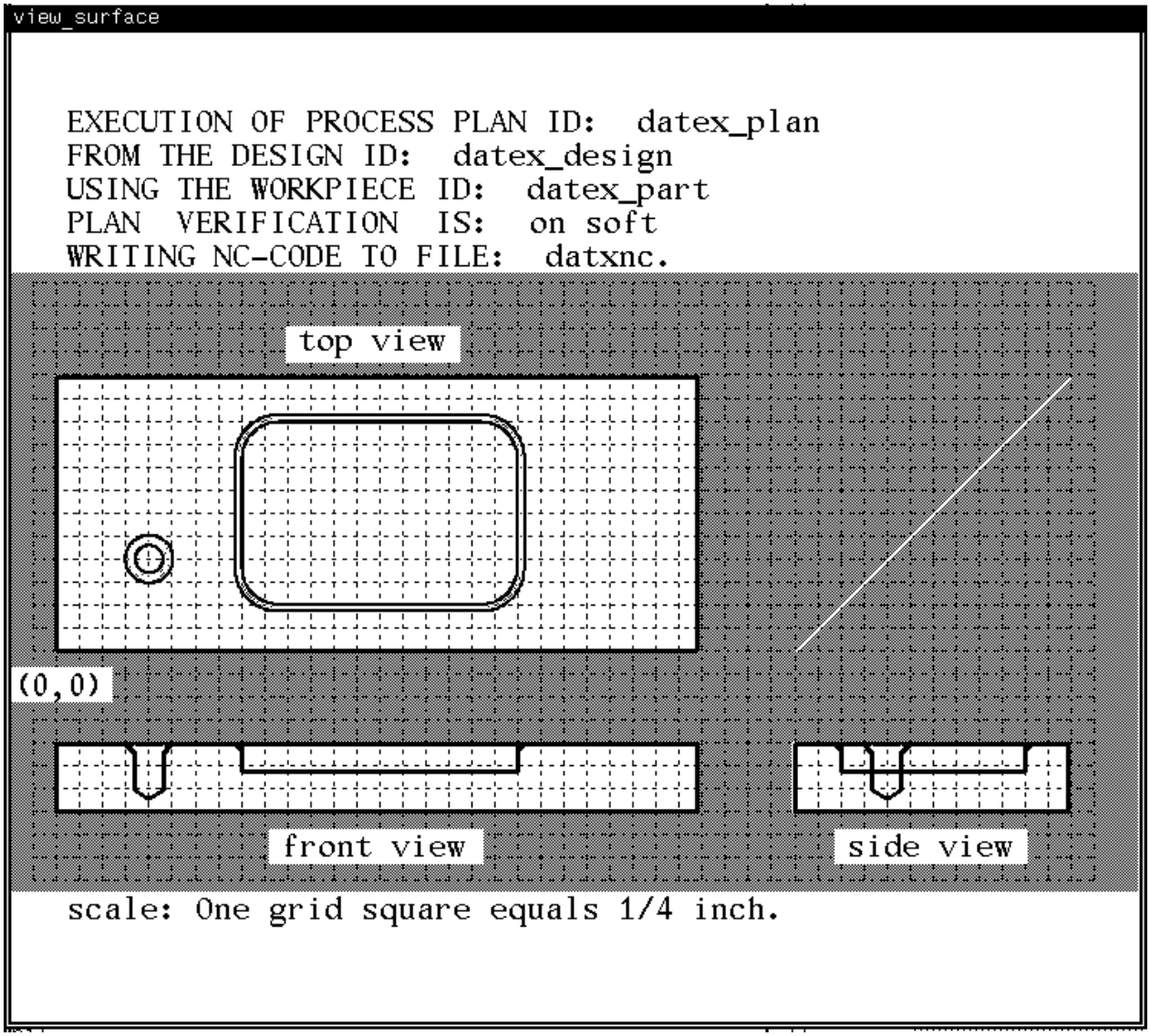
After the end of module execution, three graphics facilities may be used to help the user understand the machining process better: flash_step, flash_feature, and draw_tool_path. All three of these are available to the expert user from LISP, but only draw_tool_path is available through vws_cadm.

The LISP command (flash_step 5) will cause the part of the drawing that was generated by execution of step 5 in the enhanced plan to flash off and on a few times. Any other step number can be used in place of 5. If nothing was drawn as a result of step 5, the system will print a message that says as much.

The LISP command (flash_feature 2) will cause the drawing of feature 2 to flash off and on a few times. If feature 2 has not been drawn, the system will print a message that says so.

The tool path drawer will draw a picture of the path of the center of the tip of the tool. The drawing is done from the pseudocode. A full description of tool path drawing is given in Chapter XII of the verification paper [K&S2]. Magnified examples of tool path drawings (top view only) are shown in Figure 8 (page 53) and Figure 9 (page 54).

Figure 1. Data Execution Drawing



This figure shows the drawing made during operation of the data execution module.

Table 5. NC-Code

```
n0001 (ID,PROG,datxnc,data execution demo design,1)
n0002 g53
n0003 p69 = +0.735
n0004 p68 = +0.0                                        1
n0005 g90 g0 w(p69+(p68-10.5)) m6
n0006 p91 = 1.5
n0007 p12 = 91 m950
n0008 p90=50 p88=-.25 p89=40
n0009 p83=+16.825 p84=+7.425 p85=1
n0010 p70=0
n0011 g53 m9
n0012 g0 g90 m5 m6
n0013 g90 g0 x+36.5 y+15.0
n0014 ! Changing tool to probe for setting x_zero and y_zero
n0015 t(p89) m28 m67 m6
n0016 x(p83) y(p84)
n0017 (GSUB,OUTVWS)
n0018 p66=(p97+0.0) p67=(p98+0.0)                       2
n0019 g56 g90 x(p66) y(p67)
n0020 p90=50 p89=40
n0021 p77=+20.3 p78=+8.9125
n0022 g53 m9
n0023 g0 g90 m5 m6
n0024 g90 g0 x+36.5 y+15.0
n0025 g90 t(p89) m28 m67 m6
n0026 ! Changing tool to probe for setting z_zero
n0027 p70=0 x(p77) y(p78)
n0028 (GSUB,INTVWS)
n0029 p91 = (p92+4.424+0.0)
n0030 p12 = 91 m950                                     3
n0031 g56 g90 x(p66) y(p67)
n0032 ! 0.3 by 3 by 2.0 pocket
n0033 ! Changing tool to 0.625 inch diameter end_mill
n0034 g90 g0 m6 m9
n0035 g53
n0036 g90 g0 x+36.5 y+15.0
n0037 g90 g0 s2750 t12 d12 m3 m6
n0038 g56 g90 x(p66) y(p67)
n0039 m8
n0040 x+4.0 y+1.5
n0041 g0 z+0.1
n0042 g1 z+0.0 f5
n0043 x+3.0 y+1.5 z-0.2679
n0044 x+4.0 y+1.5 z-0.3
n0045 x+3.0 y+1.5
n0046 g0 z+1.0
n0047 x+4.0 y+1.5
n0048 z+0.1
n0049 g1 z-0.3 f40 m8 m72
n0050 f17
n0051 x+4.3125 y+1.1875
n0052 x+2.6875
n0053 y+1.8125
n0054 x+4.3125
n0055 y+1.1875
n0056 g1 x+4.6 y+0.875
n0057 g1 x+2.4
n0058 g2 x+2.375 y+0.9 r+0.025
n0059 g1 y+2.1
n0060 g2 x+2.4 y+2.125 r+0.025                          5
n0061 g1 x+4.6
n0062 g2 x+4.625 y+2.1 r+0.025
n0063 g1 y+0.9
n0064 g2 x+4.6 y+0.875 r+0.025
n0065 g1 x+4.6 y+0.8225
n0066 g1 x+2.4
n0067 g2 x+2.3225 y+0.9 r+0.0775
n0068 g1 y+2.1
n0069 g2 x+2.4 y+2.1775 r+0.0775
n0070 g1 x+4.6
n0071 g2 x+4.6775 y+2.1 r+0.0775
n0072 g1 y+0.9
n0073 g2 x+4.6 y+0.8225 r+0.0775
n0074 g0 z+1.0
n0075 x+4.6 y+0.8125
n0076 z+0.1
n0077 g1 z-0.3 f8
n0078 f17
n0079 g3 x+4.6875 y+0.9 r+0.0875
n0080 g1 y+2.1
n0081 g3 x+4.6 y+2.1875 r+0.0875                        5
n0082 g1 x+2.4                                          (cont.)
n0083 g3 x+2.3125 y+2.1 r+0.0875
n0084 g1 y+0.9
n0085 g3 x+2.4 y+0.8125 r+0.0875
n0086 g1 x+4.6
n0087 ! Changing tool to 0.375 inch diameter chamfer
n0088 g90 g0 m6 m9
n0089 g53
n0090 g90 g0 x+36.5 y+15.0
n0091 g90 g0 s5103 f28 t6 d6 m3 m6
n0092 g56 g90 x(p66) y(p67)
n0093 x+2.4 y+0.5938
n0094 m8
n0095 g0 z+0.1 ! 0.06 wide chamfer
n0096 g1 z-0.1538
n0097 g1 x+4.6
n0098 g3 x+4.9062 y+0.9 r+0.3062
n0099 g1 y+2.1
n0100 g3 x+4.6 y+2.4062 r+0.3062
n0101 g1 x+2.4
n0102 g3 x+2.0938 y+2.1 r+0.3062                        4
n0103 g1 y+0.9
n0104 g3 x+2.4 y+0.5938 r+0.3062
n0105 ! Changing tool to 0.75 inch diameter countersink
n0106 g90 g0 m6 m9
n0107 g53
n0108 g90 g0 x+36.5 y+15.0
n0109 g90 g0 s2291 f5 t4 d4 m3 m6
n0110 g56 g90 x(p66) y(p67)
n0111 x+1.0 y+1.0
n0112 m8                                                6
n0113 g82 r+0.1 z-0.2744 d4 p3=.5 ! 0.5 dia
n0114 g53 m9 m5
n0115 g90 g0 w-9.0 m6
n0116 p91 = 0.0
n0117 p12 = 91 m950
n0118 g90 g0 x+0.5 y+19.5                               7
n0119 (END,PROG)
```

5. DATA REQUIREMENTS

This section briefly describes data requirements of the Data Execution module. The data requirements of the VWS2 system and the proper formats for data structures are given in detail in [KRA4].

Two portions of the VWS2 world model are particularly important: the description of the tools currently in the milling machine, and the description of the geometry of the fixturing area.

For each tool in the machine, the following items (and others not being used) are in the world model: id, changer_slot, type, tool_type_id, cutting_depth, exposed_length, cutting_diameter, shank_diameter, tip description, number of flutes, materials the tool can cut, and material the tool is made of. As long as a process plan uses only tools that are already in the tool magazine of the milling machine, the tooling information in the world model can be regarded as fixed, and once the model is set up, the user does not need to worry about it.

The fixturing geometry information includes: obstacles, maximum run-offs in five directions, and a safe_z_plane. It is described in detail in [K&S2], Chapter VI, section 2.

The module requires a process plan, a design, and a workpiece description. Examples of these have already been shown in Table 2 and Table 3. An enhanced process plan (an example is shown in Table 4) may be either input to the module or output from it.

The module makes heavy use of the "machine_ops" database, as described in section 3.

If verification is used, the "features" database must be set up. This database includes information about tests for parameters, names of feature verifiers, and names of reference feature fit functions.

III. WORKPIECE MODEL

1. INITIALIZATION

The workpiece model, construction of which is the core of the Data Execution module, is initialized simply by copying the workpiece description of the workpiece named in the call to execute_plan. The module checks that the workpiece features are all features from the design (possibly missing subfeatures). The module prints an error message and halts if this is not so. The design and the material named in the workpiece description must match those in the process plan. If verification is on, the workpiece is verified.

2. CONSTRUCTION

As a step of the enhanced process plan is executed, any feature or subfeature produced by the step is copied from the unenhanced design onto the workpiece model. If the step is making a primary feature (such as a pocket) which has a subfeature (such as a chamfer), the subfeature is not copied to the model until the step which makes the subfeature is carried out.

3. USES

When execution is completed, the model may be saved as a description of the workpiece. Also, the model is used to check that a step is reasonable to execute, as follows.

If a step makes a feature which has a reference feature, the model is checked to be sure the reference feature already exists. If a step makes a subfeature, the model is checked to be sure the parent feature already exists. In either case, if the required feature is not present on the model, an error message is sent and the module quits work.

If a step makes a feature or subfeature that is already present on the model, once again an error message is sent and the module quits work.

If the step is a counterbore, the hole being counterbored must already exist or an error message is sent and the module quits work.

IV. MACHINING

1. OVERVIEW

The approach to machining embodied in the VWS2 system is partly the result of evolution and testing, and partly the application of the advice of expert machinists by the authors and their predecessor, Mr. Alton Quist of General Dynamics. The authors consulted with expert machinists Mr. Robert Lach of NBS, and Mr. Ken Woodall of Texas Instruments. Any misapplication of their advice is due to the authors. Mr. Quist also used the advice of expert machinists.

Most data for surface speeds and amount cut per tooth were extracted from published handbooks, as noted below. On the advice of both machinists, however, the published values for pass depths and stepovers were not used.

2. VERTICAL PASS INCREMENTS

If a feature is fairly deep, it will be unsafe to try to machine the entire depth of the feature at once. Instead, the feature is machined in several passes, with a small amount removed by each pass. The depth which can be milled on each pass is dependent upon the size and type of tool. For drills, the pass depth is one tool diameter, whereas for end mills and ball-nosed end mills the pass depth is half the tool diameter. For fly cutters the pass depth is 0.01 inches or the cutting depth of the tool, whichever is smaller. For face mills, the pass depth is 0.1 inches or the cutting depth of the tool, whichever is smaller.

Vertical pass increments are used for the following types of machining: slot milling, peripheral milling, drilling, face milling, and fly cutting. The name of the parameter used to indicate the vertical pass increment is "pass_depth".

3. TYPES OF METAL CUTTING

3.1. Overview

This subsection discusses types of metal cutting used in the VWS2 system. There are other types which may be performed by a milling machine but are not used in the system (reaming and rough milling, for example). These are not discussed here.

All but three of the machining operations specified in process plans require only a single type of cutting. Mill_pocket, mill_contour_pocket, and mill_side_contour, however, each require three types of cutting (slot milling, peripheral milling, and finish milling) in most circumstances.

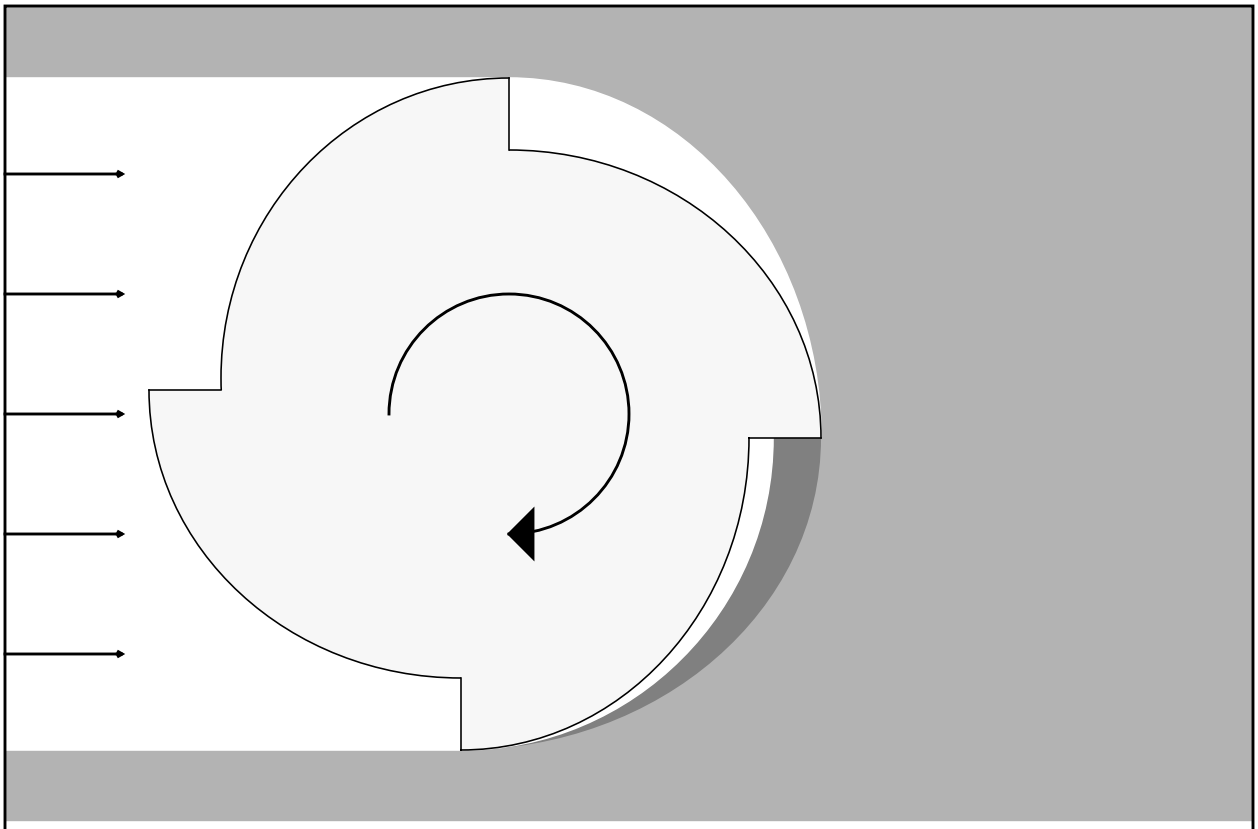
3.2. Slot Milling

3.2.1. Introduction

Slot milling is a material removal operation performed with an end mill or a ball nosed end mill in which the tool is cutting across the entire width of the tool. In other words, the tool is cutting a slot.

Figure 2 shows a cross-sectional view of a milling tool with four teeth which is cutting a slot. The figure describes the meaning of four basic milling terms: inches per tooth, feed rate, spindle speed, and surface speed. These four terms also apply to peripheral milling.

Figure 2. Slot Milling



This figure is a cross-sectional view of a milling tool with four teeth cutting a slot. The tool, shown in light gray, is moving from left to right and rotating clockwise. The dark gray patch is the material that will be removed by the rightmost tooth as it rotates one more quarter turn. The length of the flat side of that patch is the thickness of the chip being removed (the "inches per tooth" parameter used in most machining handbooks). The rate at which the tool moves from left to right is called the "feed rate". The rate of revolution of the tool (usually expressed in rpm) is the spindle speed. The speed at which the tip of a tooth is moving is called "surface speed", or simply the "speed".

3.2.2. Ramping

When a slot milling operation is performed, the first step must necessarily be to get the tool into the material at a certain depth. Since end mills do not cut effectively with the center of the tool, simply plunging the end mill into the material vertically is usually unsafe. In soft materials like aluminum, it will work, but the result is that the material under the center of the tool is not cut, but simply pushed out of the way. In order to solve this problem, the tool is moved diagonally into the material, creating a ramp. This allows more cutting to be done with the side of the tool. The maximum angle at which this ramping may be done depends on the hardness of the material. For the VWS2 system, the values used are 15 degrees for aluminum and brass, and 5 degrees for steel and monel. In some cases where it is convenient, a smaller angle is used.

Ramping is used for the slot milling phase of making pockets, contour pockets, and side contours. Ramping is also used for all types of grooves (which are made by pure slot milling). In principle, ramping is not needed when the slot to be milled intersects the side of the part, since in that case the tool does not need to move vertically into the material -- it can move to the correct depth outside the part. In practice, since changing the cutting algorithms to suit the circumstances is difficult, ramping is used even if the feature passes outside the part.

3.3. Peripheral Milling

3.3.1. Introduction

Peripheral milling is a material removal operation performed with an end mill or a ball nosed end mill in which the tool is cutting more material away from an existing edge. The tool follows a path around the periphery of the material being removed, hence the name "peripheral milling".

3.3.2. Conventional vs. Climb Cutting

Peripheral milling may be done with either conventional or climb cutting. The distinction between conventional cutting and climb cutting is illustrated in Figure 3 and Figure 4.

In climb cutting the material being cut is to the right of the path of the tool (for tools that turn clockwise, which is the norm) and the teeth of the tool first contact the part at the wide end of the chip being removed. Climb cutting may lead to tool chatter. Climb cutting tends to force the tool away from the material being cut.

In conventional cutting the material being cut is to the left of the path of the tool and the teeth of the tool first contact the part at the narrow end of the chip being removed. Conventional cutting tends to pull the tool into the material being cut.

The distinction between these two types of cutting also applies to chamfering and face milling.

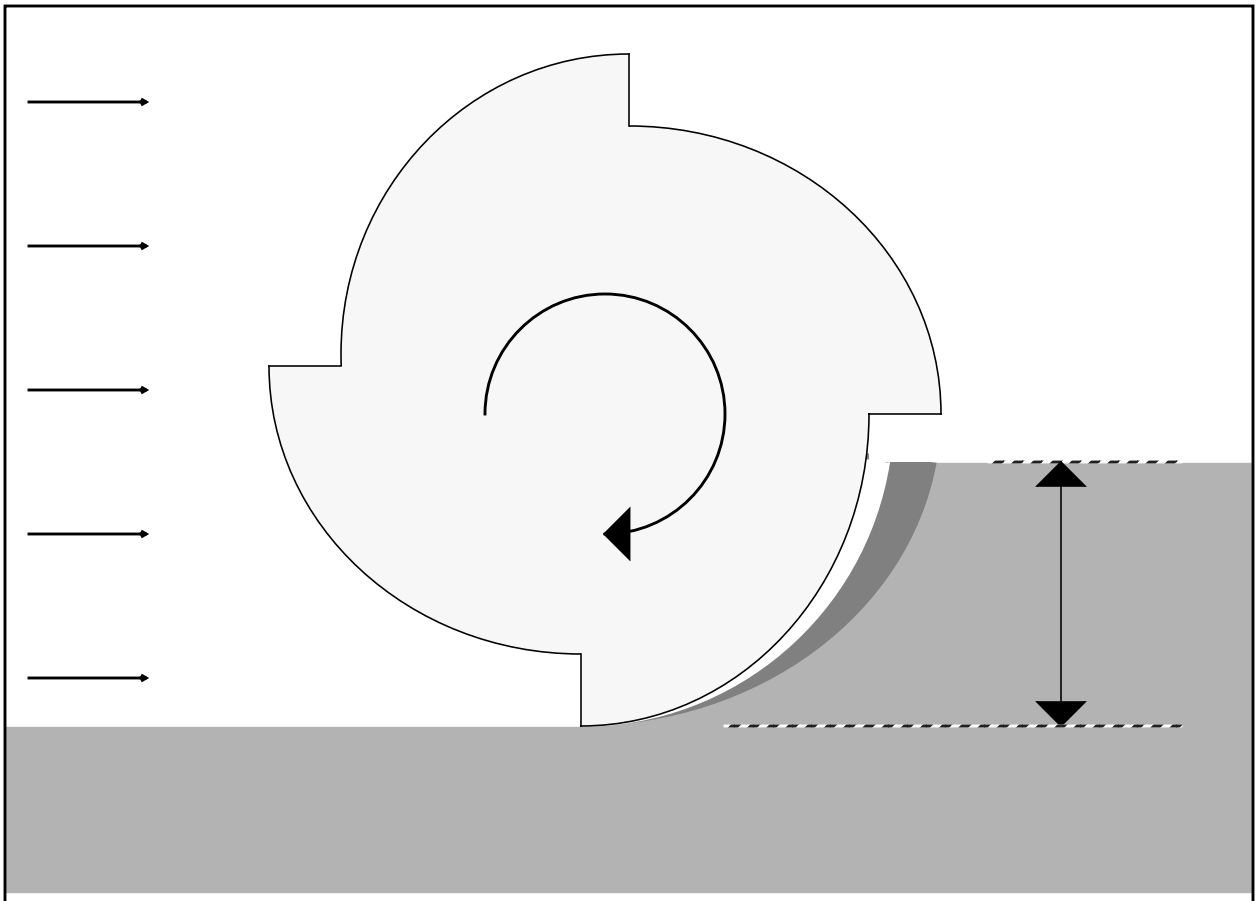
3.3.3. Stepover

When a lot of material is to be removed by peripheral milling, a number of passes are made with the tool. Between each pass the tool is moved horizontally so that the same size bite is taken by the tool on each pass. The horizontal distance between the tool paths on successive passes is called the "stepover". The size of the stepover is determined by the tool size and the material being milled. For aluminum, brass and steel, the stepover is half the tool diameter. For monel it is one fourth of the tool diameter. Stepovers are shown in Figure 3 and Figure 4.

According to our expert machinists, a larger stepover is feasible with conventional cutting than with climb cutting because of the possibility of chatter in climb cutting. This led us to structure our cutting algorithms so than only conventional peripheral milling is done.

Stepover applies to face milling as well as to peripheral milling.

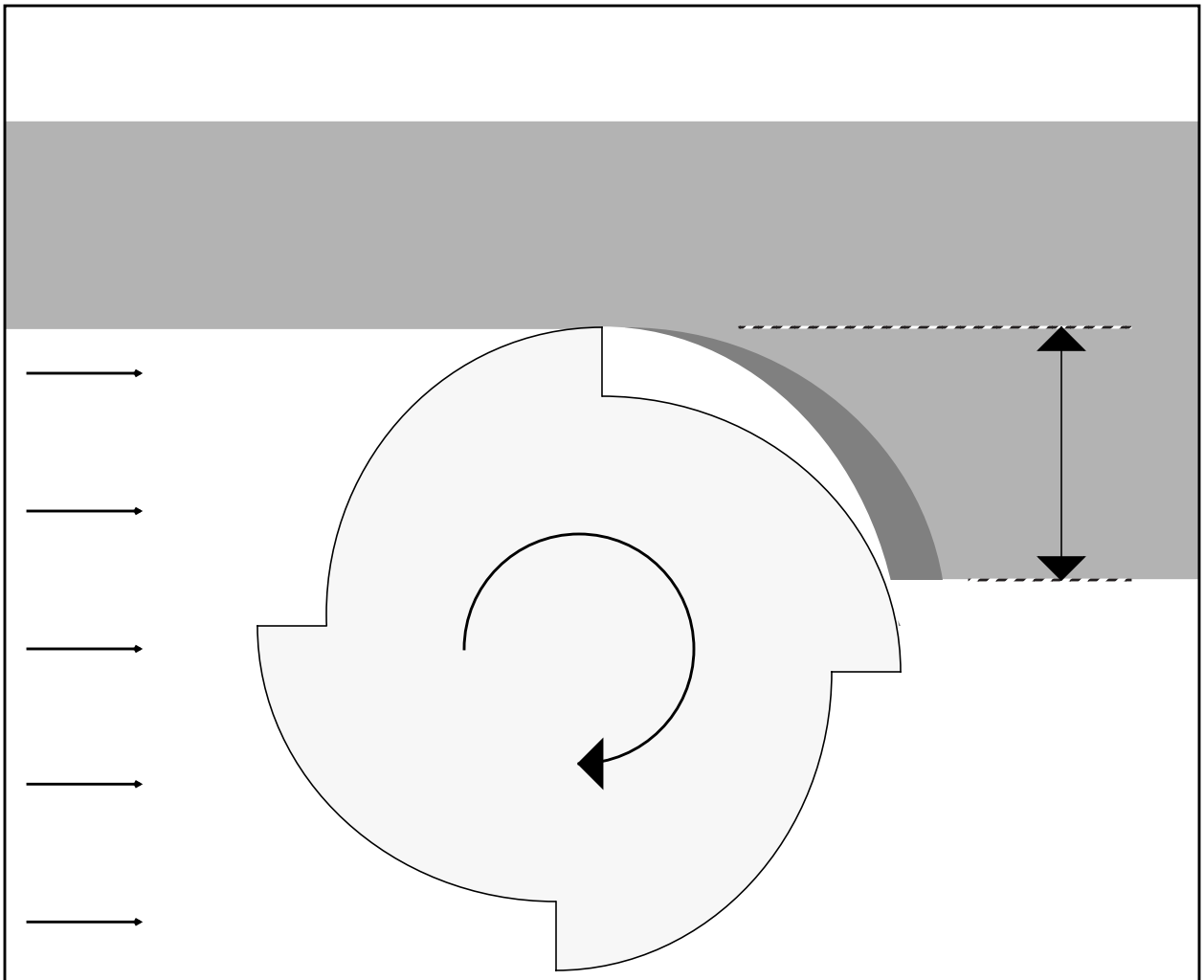
Figure 3. Climb-Cut Peripheral Milling



This figure is a cross-sectional view of a milling tool with four teeth which is doing climb-cut peripheral milling. The tool, shown in light gray, is moving from left to right and rotating clockwise. The dark gray patch is the material that will be removed by the rightmost tooth as it rotates one more quarter turn. The double-headed arrow at the right shows the stepover of this cut.

Notice that the rightmost tooth is going to contact the material at the wide end of the patch. This is the hallmark of climb cutting and may lead to tool chatter.

Figure 4. Conventional Peripheral Milling



This figure is a cross-sectional view of a milling tool with four teeth which is doing conventional peripheral milling. The tool, shown in light gray, moves from left to right and rotates clockwise. The dark gray patch is the material that will be removed by the topmost tooth as it rotates one more quarter turn. The double-headed arrow at the right shows the stepover of this cut.

Notice that the topmost tooth is going to contact the material at the narrow end of the patch. This is the hallmark of conventional cutting.

3.4. Finish Milling

Finish milling is a type of peripheral milling in which an end mill is used to make one last light cut at full depth on a pocket, contour pocket, or side contour. The size of the stepover is normally 0.01 inch in the VWS2 system. Finish milling is normally preceded by slot milling and peripheral milling.

Without finish milling, features made in several vertical passes often have horizontal lines on the walls of the feature, one line at the bottom of each pass.

3.5. Drilling

In drilling (sometimes called twist drilling by other authors), a drill with a conical tip cuts a round hole in the material. The actual cutting is done by the tip of the drill only. In the VWS2 system, all drills are assumed to have an included angle of 118 degrees at the tip, the standard angle.

A pass depth is used in drilling. The drill is retracted for a moment between passes to allow cutting fluid to re-enter the hole. Otherwise, the drill tip would not be properly lubricated.

3.6. Tapping

In tapping, a tool with a screw thread on the outside is used to form threads on the inside of a hole. The tool is always called a tap, but it may work either by cutting material out of the hole or by deforming the material inside the hole. In the former case, the tool has grooves on its outside parallel to the axis of the tool. In the latter case there are no grooves and the tool is called a "roll form tap". The VWS2 system is set up for right-handed threads only.

On the Monarch vertical milling machine used in the VWS, tapping is performed by pushing the tool lightly down into the hole with air pressure while turning the spindle clockwise. The tool literally screws itself down into the hole. When the tap reaches the desired depth, the spindle reverses and screws the tool out of the hole.

3.7. Countersinking

In a countersinking operation, a tool with a conical tip is inserted in the center of an existing hole so that the edge of the hole is cut away. In the VWS2 system the included angle at the tip of the countersink is always 82 degrees, since that is the included angle of the head of a standard machine bolt.

3.8. Chamfering

In a chamfering operation a sharp edge on a part is blunted by milling a little of it away. In the VWS2 system, the edge is always the 90 degree angle between a vertical surface and a horizontal surface -- the edge between the walls of a pocket and the top of the part, for example. In the VWS2 system, a conical chamfer tool with a 90 degree included angle at

the tip is passed around the edge to mill it flat at a 45 degree angle to both existing surfaces. In principle, any other angle might be used, or the edge might be rounded rather than flattened, but only the flat 45 degree chamfer has been implemented.

3.9. Center Drilling

A center drilling operation is performed to make a small starting hole for a drill. It is made with a center drill, which is a rigid tool with a conical tip. The tip angle of the center drill should be smaller than the tip angle of the drill. In the VWS these angles are 90 degrees and 118 degrees, respectively.

Since drills usually have a high length to diameter ratio, they are usually flexible and tend to wander across the surface of a part before the cut starts. When the entry point of a hole is off center, the entire drill bends during cutting, and the axis of the hole is tilted from the original axis of the drill. The bending may break the drill. Center drilling prevents this wandering. Thus, center drilling improves the accuracy of the location of a hole, helps keep the axis of the hole aligned correctly, and helps prevent drill breakage.

3.10. Counterboring

A counterboring operation is performed to finish up an existing hole. An end mill is plunged into the hole as deeply as desired (but not deeper than the existing hole). As used in the VWS2 system, the existing hole must be the same diameter as the end mill. This is not a requirement for counterboring in principle and is imposed to keep life simple for the VWS2 modeling system.

3.11. Face Milling

Face milling is a bulk material removal operation which leaves a good finish. Normally, the entire top surface of a part will be removed in a face milling operation, but the only requirement is that the material to be removed must be accessible from the side (since a face mill will not cut vertically). A single face milling cut may remove up to 0.1 inch of material. Typical size for a face mill is one to three inches in diameter. A face mill will typically have four to eight cutting teeth.

3.12. Fly Cutting

Fly cutting is a surfacing operation. It is performed to give a high quality surface finish or to remove small irregularities, not to remove a lot of material. Normally a fly cut will remove 0.005 to 0.01 inch of material from the entire top surface of a part. A fly cutter is a tool with one tooth at the end of an arm. The fly cutter used in the VWS has a four inch diameter.

4. SPEEDS AND FEED RATES

Spindle speeds for the different operations were calculated using data from the Machining Data Handbook, [METC], published by Metcut Research Associates, Inc., along with advice from experienced machinists. A range of values for surface speed are given in the book, depending on the hardness of the material being machined. Since the hardness of the material is not known in the system, the value used for surface speed was slightly lower than the value given for the hardest material of the given type in the relevant hardness range. The spindle speed in revolutions per minute is then calculated as the surface speed (feet per minute) times 12 (inches per foot) divided by the circumference (inches per revolution = pi times cutter_diameter in inches). The cutter_diameter is halved for chamfers, since cutting is done starting at the middle of the tool, where the diameter is half the diameter of the tool. The maximum spindle speed allowed is 5200 rpm.

Feed rates were calculated using data from the same sources. The values for feed rates in inches per tooth were chosen in the same way as the values for surface speed. The feed rate in inches per minute is then calculated as the feed (inches per tooth) times the number of flutes (teeth per revolution) times the spindle speed (revolutions per minute). For drills, end mills, and chamfers, the feed was also multiplied by a scaling factor of the tool diameter in inches. This reduces the feed rate for smaller tools. For taps, the feed rate is always 300, since this is required by the canned cycle used to do the tapping.

Both spindle speeds and feed rates depend upon the type of material being milled, so in both cases, the top-level function for calculating the values calls a subordinate function which is appropriate for the given type of material.

5. ZERO FINDING

5.1. Introduction

The origin of coordinates for machining is at the front left top corner of the workpiece. This is the same xy-location as in the coordinates for the design protocol, but the z-location in the design protocol is at the bottom of the workpiece. The setting of the xy-zero is done in one step. Setting z-zero is a another, very different operation.

5.2. Setting X-Zero and Y-Zero

5.2.1. Introduction

When a workpiece is placed in the vise on the milling machine, it is centered in the vise to within plus or minus roughly an eighth of an inch. The probe is used to locate the part exactly. There are two geometric configurations that may be probed: corners and holes. In both cases the surface of the workpiece in the vicinity must be roughly flat.

5.2.2. Probing a Corner

A corner to be probed must be a convex corner formed by the intersection of a plane parallel to the xz-plane with a plane parallel to the yz-plane. As shown in Figure 5a (page 34), which is a top view of a part, this is your garden variety corner. There are four types of corner that may be formed this way, corresponding to the four corners of a block. The corner types are numbered 1, 2, 3, and 4. On the figure there are three corners of type 1, two of type 2, one of type 3, and one of type 4. The corner being probed does not have to be an exterior corner of the workpiece. It may be a configuration appearing inside. Any of the three type 1 corners on the figure could be probed.

To probe a corner, the user must give the approximate location and the type of corner. When a rectangular block is placed in the vise, the approximate location of any of the four corners may be determined automatically. At the lower left-hand corner, for example, the y-value is the y-value of the fixed side of the vise (which would be at the top of Figure 5a) minus the width of the part. The x-value is the x-value of the middle of the vise minus half the length of the part. This is the corner that is usually probed. The automatic process planner uses this corner.

The method of probing is illustrated by the schematically drawn tool path at the lower left corner of the part in Figure 5a. The real tool path overlaps itself and would be hard to understand.

The probe comes down vertically at a location offset towards the interior of the part from the given approximate location of the corner until it hits the part. This is only to find the top of the part, not to set z-zero. Then the probe lifts up a little and moves outside the part, comes down below the top of the part and approaches the part again slowly until it hits. When it hits, the x-value of the contact point is recorded; that will be x-zero. Next the probe backs off, moves over to the other side of the corner, and approaches the part again slowly until it hits. When it hits, the y-value of the contact point is recorded; that will be y-zero.

If some corner other than the lower left-hand corner is probed, the values of x-zero and y-zero are changed by an x-offset and a y-offset provided in the process plan. Inserting these offsets in the plan must be done manually and requires knowing the geometry of the part.

5.2.3. Probing a Hole

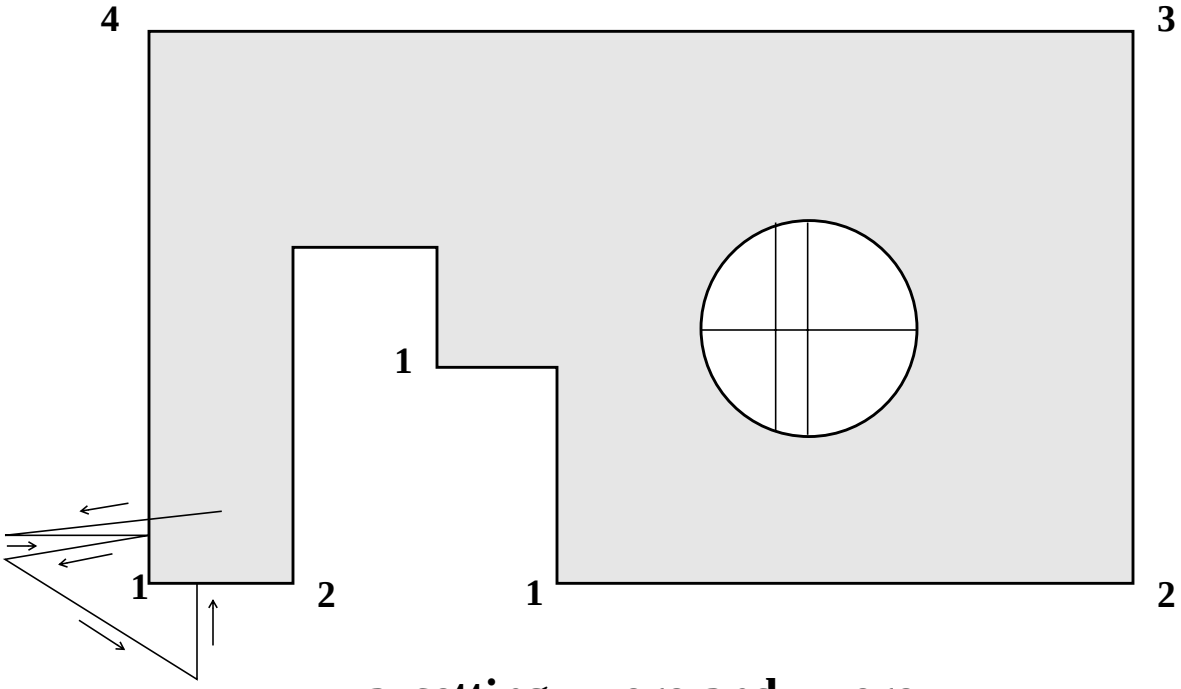
The hole to be probed should be circular. The user must provide the approximate location and diameter of the hole.

A simplified drawing of the hole probing routine is shown by the three lines crossing the hole in Figure 5a. The probe routine starts by finding the top of the part (that is why the approximate diameter is needed - so that the system knows when the probe may be assumed to be outside the hole); this is not shown on the figure. Then the probe is inserted into the hole at the given approximate center (where the vertical line on the left crosses the horizontal line on the figure). The probe moves back and forth to find y-values where it hits

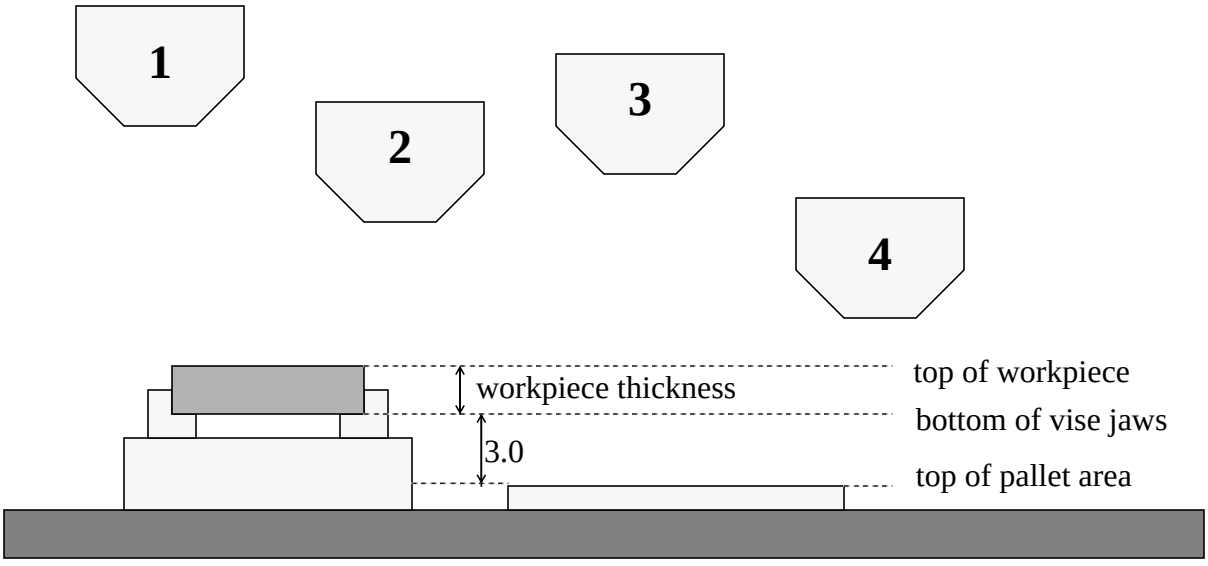
the side of the hole. The average of these two values is used as a first approximation to the y-value of the center of the hole. The probe moves to this y-value and then moves back and forth in the x-direction to find x-values where it hits the side of the hole. The average of these x-values is the x-value of the center of the hole, and their difference is the first approximation to the diameter. Finally the probe moves to the center of the hole and then moves back and forth in the y-direction to find two more y-values. The average of these new y-values is the final y-value of the center of the hole and their difference is a second approximation to the diameter. The average of the two approximations to the diameter is returned as the diameter (but this value is not being used for anything).

Offsets may be used with hole probing exactly as with corner probing.

Figure 5. Setting Zero



a. setting x-zero and y-zero



1. Head set at W= -9.0 for setting tool length offsets.
2. Head moved 1.5 inches closer to workpiece.
3. Head moved up by thickness of workpiece.
4. Head moved down three inches for milling in pallet area.

b. setting the W-axis

5.3. Setting Z-Zero

5.3.1. W-axis Setting

The entire head of the milling machine moves vertically with respect to the table of the machine. Setting z-zero is complicated by location of the head of the machine (the W-axis setting). There is no absolute zero in the vertical direction on the milling machine which can be used by the machine tool controller. The quill of the machine (the large metal cylinder that holds the spindle) has an absolute zero, and the position of the quill is what is controlled by changing a z-value, but the quill's motion is relative to the current position of the head of the machine. If the W-axis is set at -8.0 and a program is run that expects the W-axis to be set at -9.5, all the motions of the tools will be 1.5 inches above where they are expected to be.

A second complicating factor is that tool offsets are provided to compensate for the fact that different tools are different lengths. The tool offsets are set with the W-axis in a given position, and if the location of the W-axis is changed, the tool offsets must be changed.

In the VWS, tool offsets are taken where W=-9.0 and are measured to the bottom of the vise jaws, where the workpiece sits. However, to be sure that the tools will be able to go through the bottom of the workpiece without exceeding the limits of quill travel, during cutting the W-axis is positioned 1.5 inches closer to the top of the workpiece. Thus the W-axis would be set at W=-10.5 if the top of the workpiece were at the bottom of the vise jaws. This also means that all tool length offsets must be adjusted by 1.5 inches.

To compensate for the thickness of the workpiece, since NC-programs expect z=0 to be at the top of the workpiece, the head of the machine is moved upwards along the W-axis by the design thickness of the workpiece. If the workpiece is thicker than the design thickness, it will be face milled down to the correct height before any other milling is done. During this face milling, z-values will be positive.

If machining is to be done in the pallet area, the head of the machine is moved downwards 3.0 inches along the W-axis, since the difference in height between the bottom of the vise jaws and the top of the pallet holder is 3.0 inches.

The three adjustments of the W-axis and the one adjustment of the tool length offset just described suffice to set z-zero when z-zero is to be set with respect to the fixture. Figure 5b shows a side view of the pallet and the vise along with four positions of the spindle as the head of the machine is moved.

5.3.2. Setting Z-zero from the Top of the Part

In some cases it will be desirable to set z-zero at the top of the part or some distance offset from the top of the part. To do this the probe is moved to an xy-location specified by the user and brought down slowly until it hits the part. Then the z-value of the quill is recorded. The difference between this z-value and the corresponding z-value that was recorded when tool offsets were measured (plus any additional offset the user may have specified) is used to adjust all the tool length offsets.

6. TOOL CHANGING

All tool changing is done at a fixed xy location on the milling machine that is known to be free of obstacles. If tools were changed near a part being milled, there would be a risk of knocking into the part or fixturing. The height of the machine head above the table is always set to be sufficient to allow for changing the longest tools at the preset xy location.

When a tool change is required, the x and y values of the current origin are temporarily cancelled, so that absolute coordinates are in effect. The spindle is retracted, and flood coolant is turned off. The spindle is moved to the tool changing location, and the tool is changed. The Monarch VMC-75 vertical milling center holds 40 tools in its changer, and any of these may be used. After the new tool is inserted, the origin is restored to its previous location. For any of the 15 metal cutting operations the flood coolant is turned back on. For the three zero-setting operations it is left off. Also for the metal cutting operations, the tool change command may include a move to a new xy location after the change is completed.

V. AUTOMATIC NC-CODING

1. OVERVIEW

In the VWS2 system numerical control code (NC-code) is generated automatically by the Data Execution Module. The code that is generated is executable by the GE2000 machine tool controller that controls the Monarch VMC-75 Vertical Machining Center. This code is not a standard language, but it is very similar to many other NC-code languages.

The description of NC-code given in section 2 of this chapter is intended to give the reader an understanding of basic concepts and how they are applied in the VWS2 system. For details of the exact meaning of the codes, refer to [MONA].

2. NC-CODE

2.1. Introduction

NC-code is a series of lines of alphanumeric characters. The lines are interpreted line-by-line by the controller. On each line there are one or more entries separated by spaces. Each entry is usually a letter followed by a number, possibly with a plus or minus sign in between. In some cases (as on the second line in the example below), the number may be replaced by an expression in parentheses. This expression will be evaluated at the appropriate time by the controller, and the value will be a number.

The order of the entries on a line should be according to the following alphabetical sequence: n g x y r z w f s t d m. There may be two "g" entries on a line and three "m" entries, but there should not be more than one of any other type of entry.

The meanings of NC-codes used in the VWS2 system are given in Table 6. The table covers all NC-codes used except for the codes used on the first and last lines of a program, which are unique. There are many other NC-codes available for use on the Monarch which are not used in the VWS2 system.

The controller is smart enough to execute the codes on a single line in a sensible order, which may be different from the order on the line. However, all of one line is executed before the controller proceeds to the next line.

The controller keeps track of the current position of the tip of the tool which is in the spindle. Thus, to control the tool, it is sufficient to tell the controller where to move the tip of the tool next.

Table 6. Numerical Control Codes

CODE	WHAT IT MEANS	WHERE USED
d	Use the tool offset value of tool in the following slot.	many functions
f	Set the feed rate to the following value.	many functions
goto	Go to the line number indicated.	depth_loop, version 2
gsub	Run the following subroutine.	3 set0's
g0	Ignore feed rate and move at traverse speed (fast).	many functions
g1	Move in a straight line to the specified point.	many functions
g2	Make a clockwise arc of less than a semicircle to the point given by x and y, using the radius given by r. If z-value is given, move linearly in z, making a helix.	many functions
g3	Like g2, except counterclockwise.	many functions
g53	Cancel the x and y zero settings.	many functions
g56	Set x and y zero.	change_tool, 3 set0's
g81	Traverse to r-plane, feed to z-value, retract.	center_drill_nc
g82	Traverse to r-plane, feed to z-value, hesitate, retract.	cbore_hole_nc csink_hole_nc
g83	Traverse to r-plane, peck feed to z-value, retract.	hole_nc
g84	Run an air-pressure driven tapping cycle.	tap_hole_nc
g90	Interpret x, y, and z values as coordinates measured from the current origin.	many functions
if	Check the truth of an expression. Do something if true	depth_loop, version 2
m3	Start the spindle clockwise	change_tool
m5	Stop the spindle.	close_nc, 3 set0's
m6	Retract the spindle.	many functions
m8	Turn flood coolant on.	many functions
m9	Turn flood coolant off.	many functions
m28	Unknown - not documented.	2 set0's
m67	Unknown - not documented	2 set0's
m72	Lock the quill against motion in the z-direction.	pocket_chunk
m950	Set the z-axis offset parameter.	close_nc, init_nc, set0z
n	Starts line number, has no effect except as goto label.	print_nc_line
p	Denotes a parameter. Used in a variety of ways.	depth_loop, version 2 3 set0's
r	If used with g2 or g3, the following number is a radius. Otherwise, following number is a z-value above part.	many functions
s	Set the spindle speed to the following number.	many functions
t	Change to the tool whose slot number follows.	change_tool
w	The following number is a w-axis value.	init_nc, close_nc
x	The following number is an x-value.	many functions
y	The following number is a y-value.	many functions
z	The following number is a z-value.	many functions
!	What follows on this line is only a comment.	many functions

2.2. An Example

To understand how NC-code is interpreted, it is useful to look at an example. Here are seven typical lines taken from a program written by the VWS2 system.

```
n0025 g90 g0 s3437 t2 d2 m3 m6
n0026 g56 g90 x(p66) y(p67)
n0027 m8
n0028 x+0.3816 y-0.26
n0029 g90 g0 z+0.1
n0030 g1 z+0.0 f5
n0031 g2 x+0.0473 y+0.2094 r+0.3538 z-0.0121
```

By referring to Table 6, we can interpret the seven lines of code as follows.

To begin with, the first entry on each line is simply a line number. The alphabetic part of the line number is "n", and the numeric parts are in numerical order.

The entries on line n0025 mean:

- g90 = use absolute positioning with respect to the current origin
- g0 = move at traverse rate when a move is indicated
- s3437 = set the spindle speed to 3437 rpm
- t2 = put the tool in changer slot 2 into the spindle
- d2 = use the tool offset value stored for the tool in slot 2
- m3 = start the spindle clockwise at the new spindle speed
- m6 = retract the spindle

In executing this line, the controller retracts the spindle first, then stops it if it is turning (even though there is no m5 command), finds tool 2, puts it into the spindle, and restarts the spindle at 3437 rpm. Notice that the g90 and g0 commands have no effect on this line. These commands continue in effect until countermanded by some other g code and may have an effect on following lines.

The entries on line n0026 mean:

- g56 = Set the x and y values of the origin at the values given on this line (in absolute coordinates)
- g90 = use absolute positioning with respect to the current origin
- x(p66) = the x value is the value of parameter 66 (which must have been set earlier)
- y(p67) = the y value is the value of parameter 67 (which must have been set earlier)

The machine does not move when this line is executed, it just changes internal variables.

The entry on line n0027 means:

- m8 = turn on flood coolant

The entries on line n0028 mean:
x+0.3816 = move to where x equals 0.3816
y-0.26 = move to where y equals -0.26

The spindle moves in a straight line at traverse speed (as set on line n0025) to the point whose x and y values are given on this line. The x and y values are measured as coordinates with respect to the new origin set on line n0026.

The entries on line n0029 mean:
g90 = use absolute positioning with respect to the current origin
g0 = move at traverse rate
z+0.1 = move to where z equals 0.1

Neither the g90 nor the g0 is really essential on this line, since both are already in force, but they do no harm, either. There are many instances in the VWS2 system of reiterating g codes that are already in force. This programming practice is recommended in the programming manual [MONA]. Codes other than g codes are not reiterated.

The entries on line n0030 mean:
g1 = move in a straight line at the current feed rate
z+0.0 = move to where z equals 0.0
f5 = set the feed rate to 5 inches per minute

The feed rate will be reset before the move starts.

The entries on line n0031 mean:
g2 = make a clockwise arc in the xy-plane
x+0.0473 = the arc should end where x equals 0.0473
y+0.2094 = the arc should end where y equals 0.2094
r+0.3538 = the radius of the arc should be 0.3538
z-0.0121 = at the same time, move the tool to where z equals -0.0121

The z move is linear, so that the actual tool path is a portion of a helix. The feed rate which was set on the preceding line is still in effect, as is the g90 command on the line before that. Two clockwise arcs are possible. The one that is less than a semicircle is used. If the given radius is too small, the controller will come to a halt and post an error message.

2.3. First and Last Lines of a Program

The first line of an NC program always has the following format:

n0001 (ID,PROG,lok1nc,locking clevis first cut,1)

The term "lok1nc" on the above line is the program identifier, and may be replaced by any other sequence of not more than six alphanumeric characters. The term "locking clevis first cut" on the above line is a brief description of the program and may be replaced by any other sequence of not more than 30 alphanumeric characters and spaces. No other spaces are allowed within the parentheses.

The last line always has the format:

n0160 (END,PROG)

where "0160" may be replaced by any other four digits.

3. GENERAL APPROACH TO CODE-WRITING

3.1. Introduction

As noted earlier, if the NC-code writing option of the Data Execution module is on, a block of code is written for each step in the enhanced process plan. Since there are twenty-one different work elements in the VWS2 system, and two of them (face_mill and fly_cut) share an NC-coding function, there are twenty coding functions which may be called. Five of these (init_nc, close_nc, and the three zero-setting functions) do not write code for cutting metal; the other 15 do.

The 15 coders for metal cutting always check first if a tool change is required, and make a change if needed. In order to do this, the module keeps track at all times of which tool is in the spindle. The module also keeps track of the current spindle speed, and changes it only when necessary. Earlier versions of the system kept track of other items, as well, but the benefit of keeping track (in shorter code and reduced machining time) was slight and not worth the cost of complicating the NC-coding system.

At the end of a cut, the tool is always left down in the material, and it is left to the next operation to withdraw the tool.

The NC-code is written as pseudocode by the system and stored in reverse order from the way it will ultimately appear in the file. At the end of the module’s operation, the pseudocode is printed out as real NC-code. Also at the end of the module’s operation, if the drawing option is on, a copy of the pseudocode (in the correct order) is given to the graphics system in case the user wants to see the tool path. If the module aborts during operation, as it may if some feature or operation fails verification, the pseudocode is scrapped, and nothing is printed out.

3.2. Pseudocode and Print Routine

The use of pseudocode was adopted principally to simplify the job of writing NC-coding functions, but it has proved to have several other advantages. The biggest side advantage is that pseudocode is easily used by the drawing system to draw tool paths. This is because its format is native to LISP and because conceptually separate items (such as the name of a coordinate and its value) are still separate. Drawing from real code would require that a parsing routine be written to separate groups of characters into conceptually distinct bunches.

An earlier version of the system wrote NC-code line-by-line to a file, opening and closing a port repeatedly. The system could not handle the frequent opening and closing of the port correctly, and was dropping lines of code. The use of pseudocode cleared the problem up since the port is opened only once and closed once.

Pseudocode is stored in the LISP environment as a list of sublists. Each sublist represents one line of code. The pseudocode differs from the final code in the following respects (in addition to the obvious difference of the pseudocode being in the environment only and the real code being in a file only).

A. In pseudocode floating point numbers are kept with all the significant figures LISP provides (17 or so). In real code, floating point numbers are rounded off to 4 decimal places, and terminal zeros are suppressed in the second through fourth places.

B. In pseudocode some numbers may be either fixed or floating point and may or may not have a sign; namely numbers that represent values of r, x, y, or z. Such numbers are converted to floating point numbers by the print routine and are always printed with a sign.

C. In pseudocode there are no line numbers; they are not needed since lists are inherently ordered. Line numbers appear in the real code. The numbers are generated by the print routine, which uses a counter to keep track. The absence of line numbers in pseudocode is a big help when copies of several lines are to be made, as when repeating code at several different depths.

D. Pseudocode may contain strings. Real code contains no strings. Strings may be used to hold bits of code that will be welded together by the print routine.

E. The print routine automatically deletes spaces following some characters in the pseudocode, unless instructed otherwise.

F. Pseudocode may contain three terms which are not printed, but are interpreted specially by the print routine. These are:

Term	*Interpretation*
nil	Do not print anything. Using nil simplifies the writing of NC-coding functions.
sign	Print the sign of the following number.
no_space	Do not put any space between the previous and following items.

The pseudocode that results in the seven lines of code given in the example of section 2.2. is as follows:

```
(g2 x 0.04729792843330796 y 0.2093696684367282 r 0.35375 z -0.0121017118049899)
(g1 z 0 f 5)
(g90 g0 z 0.1)
(x 0.3816198612624016 y -0.26)
(m8)
(g56 g90 "x(p66)" "y(p67)")
(g90 g0 s 3437 t 2 d 2 m3 m6)
```

An example of line made up in bits is the following:

```
("p66=(p97" sign 0.0 no_space ")" "p67=(p98" sign 0.0 no_space ")")
```

If this happens to be line 18, it prints out as follows:

```
n0018 p66=(p97+0.0) p67=(p98+0.0)
```

3.3. Comments

Whenever a tool change occurs, a comment line is put into the NC-code, describing the new tool. Most of the 15 metal cutters insert a comment describing the operation or the feature being machined. The comment may appear on a separate line, or at the end of an effective line. Comments are denoted by an exclamation mark (!), and anything following the mark on a line of code is interpreted as a comment, no matter how it would ordinarily be interpreted.

3.4. Machine Capabilities

3.4.1. Introduction

The NC-codes selected to be used in the VWS2 system were generally ones which stand for capabilities which are common to most numerically controlled milling machines. This was done so that the system could be adapted easily to other machines. The system has been adapted for a different milling machine at the University of Maryland, and parts have been cut using NC-code written by the adapted system.

By not using any of the zero-setting routines, and using version 1 of the depth_loop function, only the common capabilities listed in the next subsection are needed, except that init_nc, close_nc, and change_tool all would have to be rewritten for a milling machine which does not have parameter capability, since those three functions use parameters.

3.4.2. Common Capabilities

The following capabilities used in the VWS2 system we believe to be common to most numerically controlled machine tools. They are given here in the order they appear in Table 6.

1. use a tool offset value
2. set feed rate
3. move at traverse speed
4. mill in a straight line in three dimensions
5. make a circular arc in two dimensions or a helical arc in three
6. run canned cycles like those represented by g81, g82, g83, and g84.
7. interpret x, y, and z values as coordinates
   (the alternative is to interpret x, y, and z values as distances from the current location).
8. start and stop the spindle
9. retract the spindle
10. turn coolant on and off
11. lock the quill
12. use NC-code with line numbers in it
13. set spindle speed
14. use radii in making arcs and traverse to a given z in canned cycles
15. change a tool
16. move the w-axis
17. use NC-code with comments in it

The air pressure driven tapping cycle used to implement the g84 code on the Monarch VMC-75 is not a common capability, but a canned tapping cycle of some sort is common and could be substituted.

3.4.3. Less Common Capabilities

The least common capability of a milling machine used in the VWS2 system is the Monarch's probing capability. This capability is employed in the three zero-setting operations. Probing is used in the system principally so that we can deal with slight variations in the location of parts which have been loaded automatically by a robot into the vise. The g53, g56, m27, m28, and m950 codes and subroutine calls are used only in connection with zero-setting.

A second less common capability is the use of parameters and the evaluation of expressions including parameters. Parameters are used in the three zero-setting operations, init_nc, close_nc, change_tool, and the second version of depth_loop.

The last two uncommon capabilities are used only in the second version of depth_loop, namely jumping to some other line of code than the next one (a "goto" statement) and using the conditional "if" to trigger the jump. The depth_loop function writes all the NC-code needed to repeat a series of lines of code at increasing depths. The first version of depth_loop prints new lines at each new depth. The second version reuses the lines by using a parameter to represent depth, using "if" and "goto" to loop back as many times as

necessary, and increasing the value of the depth parameter on each loop. The use of the second version of depth_loop may save several hundred lines in a 1000 line program, depending upon the nature of the cuts, of course.

4. SIMPLE ALGORITHMS

4.1. Drilling

The drill is located over the center of the hole, brought down quickly to 0.1 inch above the material, and fed vertically downwards to make the hole. During feeding the drill is retracted for a moment and then fed back into the hole each time it goes another pass depth deeper. This is called pecking. Once in the right xy location, the entire algorithm is carried out by a g83 canned cycle on one line of NC-code.

4.2. Tapping

The tap is located over the center of the hole, brought down quickly to 0.1 inch above the material, and pushed vertically downwards by air pressure. The tap screws itself into the material to the given depth. The spindle reverses and unscrews the tap from the hole. Once in the right xy location, the entire algorithm is carried out by a g84 canned cycle on one line of NC-code.

4.3. Countersinking

The countersink is located over the center of the hole, brought down quickly to 0.1 inch above the material, and fed vertically downwards to the necessary depth. At the final depth the tool hesitates (dwells) for half a second to make a clean cut, and then it is withdrawn. Once in the right xy location, the entire algorithm is carried out by a g82 canned cycle on one line of NC-code.

4.4. Milling a Straight_Groove

A straight groove is milled by ramping the tool back and forth into the material. On each ramp the tool is angled downwards at the minimum of:

A. the maximum ramping angle,
B. an angle that will make the vertical depth of the cut at the far end of the groove be one pass_depth,
C. an angle that will reach the bottom of the groove at the far end.

After the last ramp is done, the tool is returned to the starting point at constant depth to complete the work.

If the length of the tool_path for a straight_groove is less than 0.1 inches, then the groove is too short to ramp. In this case, an error message is issued, and the groove is milled by plunging at a low feed rate into the material at one end of the groove and milling to the other end. The experienced user, having been notified, can then decide whether or not it is safe to use this code.

4.5. Milling a Groove

The set of corners for the groove is converted into a set of contour corners, and the contour groove milling algorithm is used.

4.6. Chamfering

A chamfer tool is moved on a path that is a rectangle with (possibly) rounded corners. Recall that a chamfering operation breaks an edge where a vertical wall meets a horizontal surface of the part. The tool meets the vertical wall at exactly half a tool radius from the axis of the tool and cuts with only the upper half of the tool.

4.7. Center Drilling

A center drill is brought to the correct xy location, brought down quickly to 0.1 inch above the material, and fed vertically downwards to the necessary depth. Once in the right xy location, the entire algorithm is carried out by a g81 canned cycle on one line of NC-code.

4.8. Counterboring

The counterboring algorithm is identical to the countersinking algorithm, except that the dwell time is the amount of time it takes for the spindle to make two full turns, or a quarter of a second, whichever is greater.

5. NON-TRIVIAL BUT EASY ALGORITHMS

5.1. Face Milling

The face milling algorithm mills away a rectangular area of the top of the part to a fixed depth. Normally the rectangle includes the whole top surface of the part, but this is not required.

If the rectangle is longer in the x-direction than the y-direction, the face mill is fed back and forth across the part parallel to the x-axis, removing a strip of material one stepover wide on each horizontal move. After each move parallel to the x-axis, the tool is rapidly moved one stepover in the y-direction while it is outside the rectangle. The last move parallel to the x-axis is adjusted so that the face mill extends beyond the rectangle by 0.1 inch in the positive y-direction (less if the last strip is nearly as wide as the tool).

If the total depth to be milled is greater than the pass depth, the same moves are repeated until the final depth is reached.

If the rectangle is longer in the y-direction than the x-direction, the back and forth motion is parallel to the y-axis and the stepover is made in the x-direction.

The back and forth motion results in an alternation between conventional cutting and climb cutting. This has not proved to be a problem.

5.2. Fly Cutting

The fly cutting algorithm is identical to the face milling algorithm. In fact, the same NC-coding function is used for both. Of course speeds, feed rates, pass depths, and stepovers are different for the two operations, but these are arguments to the NC-coding function.

6. SOPHISTICATED ALGORITHMS

6.1. Pocket Milling

6.1.1. Introduction

The normal pocket algorithm is non-trivial but easy. However, the overall algorithm is complicated by the variety of situations it is designed to handle. The complicating factors are: very small pockets, small pockets, and making the initial slot. Figure 6 shows the top view of the cutter path for the normal situation and the three complications in milling a pocket.

6.1.2. Normal Pocket Algorithm

As shown in Figure 6(A), the normal situation is that a slot is milled across the center of the pocket to full depth. Then several passes are made at successively increasing depths to mill away the bulk of the material inside the pocket to within 0.01 inch of the final sides of the pocket. Finally, a finish pass around the perimeter of the pocket is done at the full depth to even up the sides of the pocket.

The bulk removal passes are identical except for the depth of the cut. On each pass the tool is inserted in the initial slot, moved downwards and to the right to start the cut, and then passed around the expanding periphery of uncut material in a clockwise direction, moving downwards and to the right again after each full circuit. On the first few circuits, the tool path is rectangular, but as milling proceeds, the corners of the rectangle are rounded concentrically with the corners of the pocket.

6.1.3. Very Small Pockets

A very small pocket is one whose length and width are both less than 0.02 inch greater than the diameter of the end mill being used to make the pocket. A very small pocket is made by plunging the end mill straight into the material to full depth, and then making a finish cut (unless the tool is the same size as the pocket, in which case no finish cut is needed). This is shown in Figure 6(B). Because plunge cutting may be unsafe, a message is sent to the user if a plunge cut is to be made.

6.1.4. Small Pockets

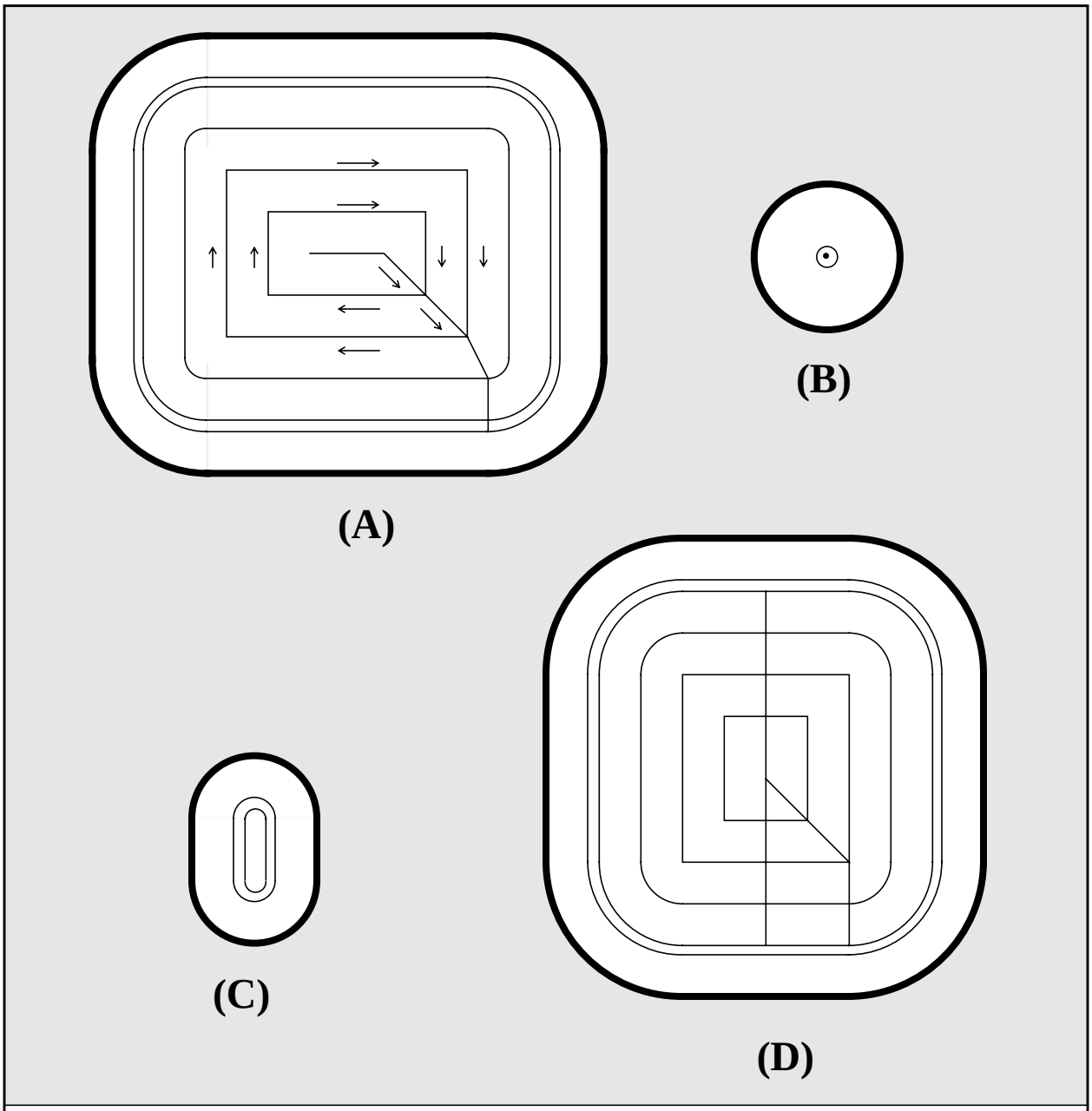
If both the length and the width of the pocket are smaller than 1.5 times the tool diameter plus 0.02 inch, so that there is not enough room to mill a slot, but the pocket is not very small (as defined above), bulk material removal is accomplished by generating a set of contour corners, and a contour groove is milled 0.01 inch (or less, if the pocket is very narrow) inside in final walls. Then a finish pass is made. This is shown in Figure 6(C). Because the pocket is small, making a contour groove will not leave an island inside the pocket.

6.1.5. Making the Initial Slot

In the normal situation, the length of the initial slot is the difference between the length and width of the pocket. If the pocket is longer in the x-direction, the slot is horizontal. If it is longer in the y-direction, the slot is vertical. If the normal size slot is long enough (more than one tool radius long), it is made by ramping down into it.

If the pocket is square, or close to it, the normal slot length will be too small for ramping. In this case, as shown in Figure 6(D), the ends of the slot are extended to near the walls of the pocket to allow room for ramping. This will provide a suitable line unless the pocket is small.

Figure 6. Pocket Cutting Tool Paths



In this figure tool paths are shown for the four variations of the pocket making algorithm.
Heavy outlines show the pockets. Light lines show the tool paths.
The distance between the outermost pair of tool path lines, 0.01, is exaggerated in each case.
(A) the normal situation. First a slot is made, then material removed, then a finish cut made.
(B) a very small pocket. First a hole is plunge cut, then a finish cut made.
(C) a small pocket. First a contour groove is made and then a finish cut made.
(D) length and width are almost equal. The initial slot goes across the pocket, rest is like A.

6.2. Text Milling

Text milling is accomplished by milling each character separately. The first character is milled at the starting location of the text. The routine which generates NC-code for one character also returns the x-value of the location where the next character should start.

The VWS2 methods for dealing with text are described in detail in sections 2.15.7 and 2.16 of chapter II of the design protocol paper [K&J2].

Each character in each font has a template for making the character stored in the fonts database. The template for making a character consists of two lists: nc_points and nc_path. In the plain font the letter R, for example, has the nc_points and nc_path shown in Figure 7.

Each entry on the nc_points list (except the last one) is a pair of numbers which represent the x and y coordinates of a point on the character. These coordinates are scaled according to the height of the text and are translated to the proper xy-location.

The last entry on the nc_points list is the point to go to when the character is finished. The x-value of the last point is always larger than the x-value of the lower right-hand corner of an imaginary parallelogram that just fits around the character by an amount which is the spacing for the particular font. The y-value of the last entry is always zero.

Each entry on the nc_path is a pair in which the first item represents a type of path to mill to the next point, and the second item tells which point on the nc_points list is the next point.

The letter codes found on the nc_path list mean the following:

s = straight line
w = clockwise arc
ccw = counterclockwise arc
j = jump without milling

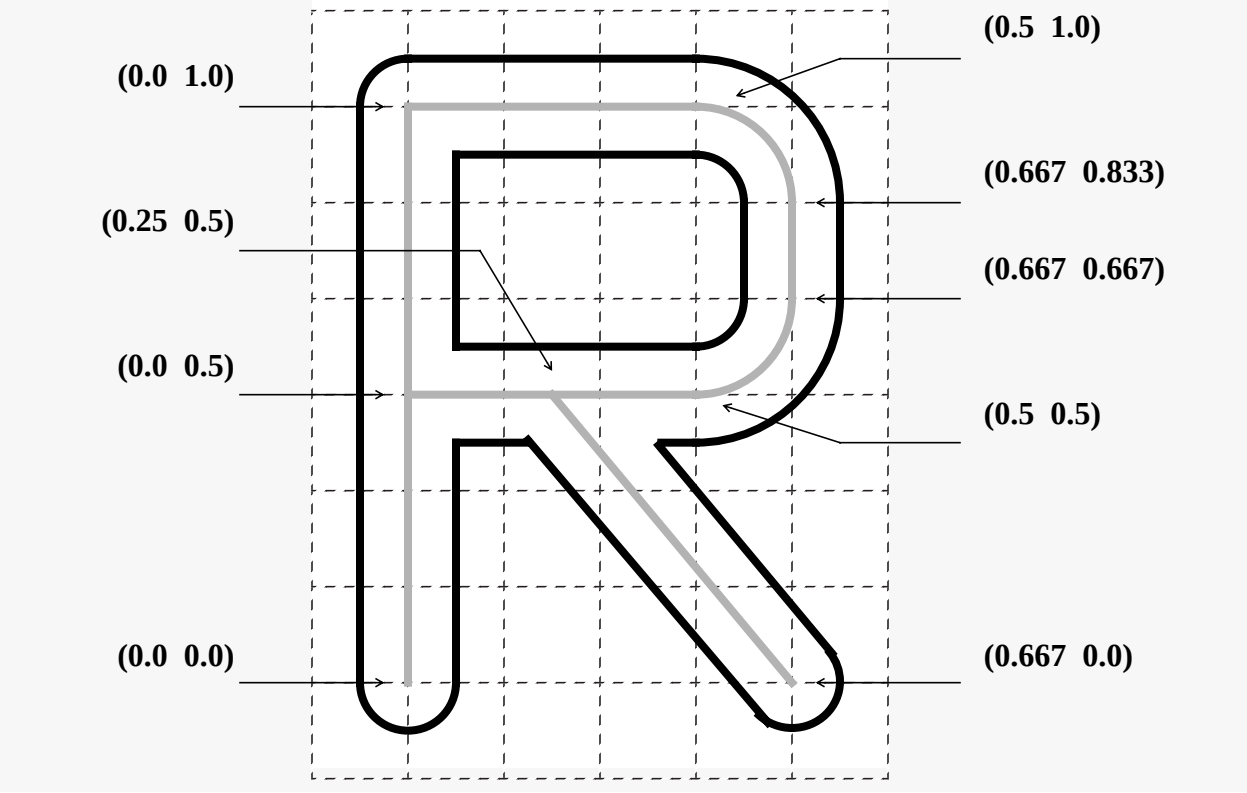
The use of nc_points and nc_path to mill the letter R is explained in Figure 7.

If text is deep, it will be cut in several passes.

The initial entry of the tool into a text character is by plunge cutting. While this is not ideal, since text is almost always quite shallow, entry by plunge cutting is almost always safe.

For most fonts, the radius used for making arcs is of such a size that arcs and straight lines join together smoothly. However, one of the options of the font maker is to flatten out arcs by making the radius larger. The "angular" font uses this option. No change in any other part of the system is required.

Figure 7. Text Tool Path



This figure shows the tool path for milling the letter R in plain font. The outline of the R is shown with a heavy black line. The tool path is shown with a heavy grey line.

Two sets of data are needed. Nc_points is a list of pairs of numbers representing coordinates:
((0 0) (0 0.5) (0 1) (0.5 1) (0.667 0.833) (0.667 0.667) (0.5 0.5) (0.25 0.5) (0.667 0) (1 0))

Nc_path is a list of pairs. The first element of each pair is a letter code, and the second element is an integer standing for an element of the nc_points:
((s 3) (s 4) (w 5) (s 6) (w 7) (s 2) (j 8) (s 9)).

The machining of a character always starts at the first point on the nc_points list. Thus, milling begins by moving to (0 0) and inserting the tool. Then the nc_path is followed in order:
(s 3) = mill straight to point 3 at (0 1).
(s 4) = mill straight to point 4 at (0.5 1).
(w 5) = mill in a clockwise arc to point 5 at (0.667 0.833).
(s 6) = mill straight to point 6 at (0.667 0.667).
(w 7) = mill in a clockwise arc to point 7 at (0.5 0.5).
(s 2) = mill straight to point 2 at (0 0.5).
(j 8) = pull the tool up, jump to point 8 at (0.25 0.5), and reinsert the tool.
(s 9) = mill straight to point 9 at (0.667 0).
The last point on the nc_points list, (1 0), shows where the next character should start.

For NC-coding, the nc_points list is scaled and translated appropriately before being used.

6.3. Milling a Contour Groove

A contour groove is specified by a set of corners. The notion of a set of contour corners (a contour outline) is discussed in detail in the design protocol paper [K&J2], Chapter II, section 2.15.8. Included in the description of each corner are its starting and ending points, the radius of the arc, and the angle it subtends. If the first corner has a nil radius, then the groove is open. If the radius for the first corner has a value, then the groove is closed, and can be milled by ramping downward in circuits around the groove. If the groove is open, then the groove must be milled back and forth between the ends.

A separate routine is used for each type of contour groove. For a closed groove, the ramp angle is set according to the type of material (5 degrees for steel and monel, 15 degrees for aluminum and brass). The tool, which is an end mill for a flat-bottomed groove and a ball-nosed-end-mill for a round-bottomed groove, is ramped into the material at the ramp angle along the contour outline until (i) one pass depth is reached, or (ii) the bottom of the cut is reached, or (iii) a complete circuit around the outline is made. In case (i), the tool path is levelled off for the rest of the circuit. In case (ii), the path is levelled and one more complete circuit is made to the place where levelling started.

For an open groove, the method is slightly different. The ramping depth is found as the depth, the pass_depth, or the length of the groove times the tangent of the maximum ramping angle, whichever is smallest. The change in depth for each segment is then the fraction of the groove length covered by the segment times the ramping depth. This is repeated, with NC code written for each segment, until the end of the groove is reached. The groove is then milled in the reverse direction at a constant depth. This process is repeated as many times as necessary to reach the proper depth.

6.4. Milling a Contour Pocket

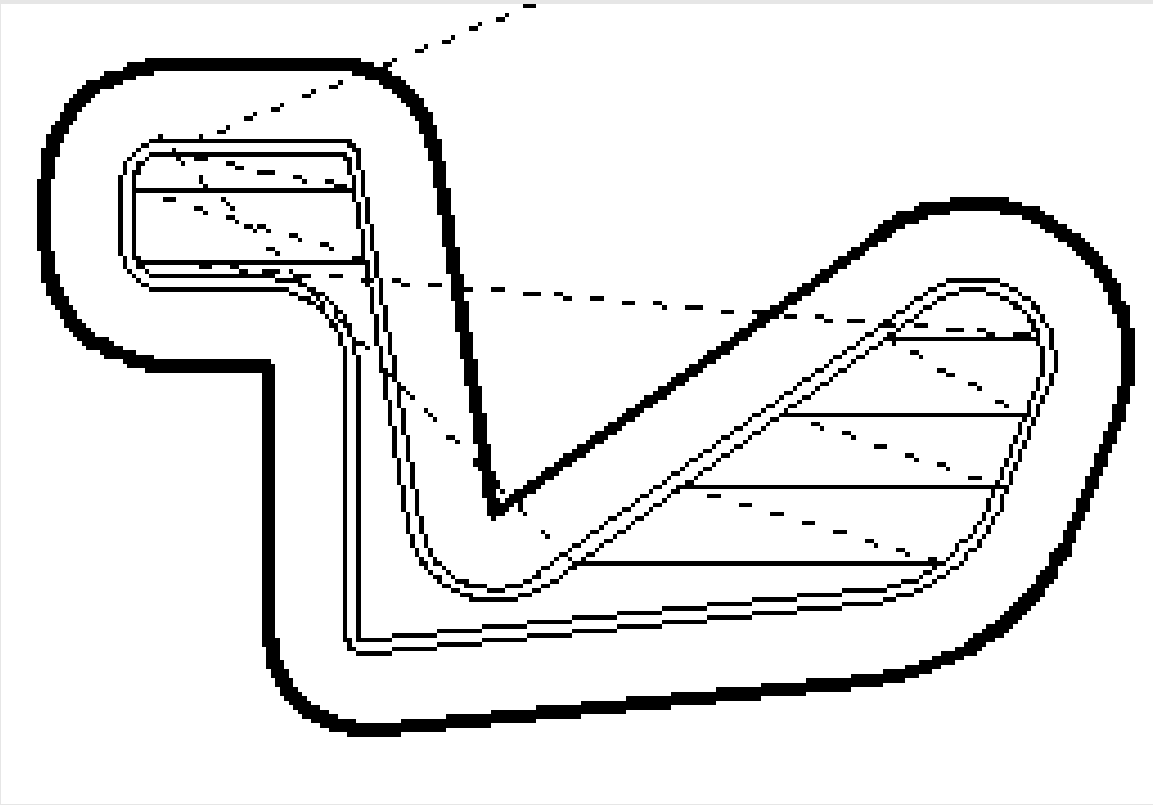
The algorithm for milling a contour pocket is shown in Figure 8. The algorithm is described on the figure, as well. The initial cut is made by the routine described in section 6.3 for ramping into closed contour grooves. The final cut is made by a simpler routine which inserts the tool to full depth first and then follows the contour.

A second version of this algorithm, written earlier, replaces the middle portion of the algorithm (bulk material removal) with a zigzag pattern. This second algorithm is more difficult. It was discarded because it alternates between conventional cutting on the zig and climb cutting on the zag. Using the algorithm would have required that a much smaller stepover be used for steel to avoid chatter during climb cutting. This would have lengthened machining time considerably.

6.5. Milling a Side Contour

The algorithm for milling a side contour is shown in Figure 9. The algorithm is described on the figure, as well. As with a contour pocket, the initial cut is made by the routine described in section 6.3 for ramping into closed contour grooves, and the final cut is made by the simpler routine.

Figure 8. Contour Pocket Tool Path



This tool path drawing, which is a bit image taken from the screen, shows the tool path used to cut a particular contour pocket. The heavy line marks the outline of the contour pocket. A light solid line is drawn when the tool is cutting. A light dotted line is drawn when the tool is in the air above the part.

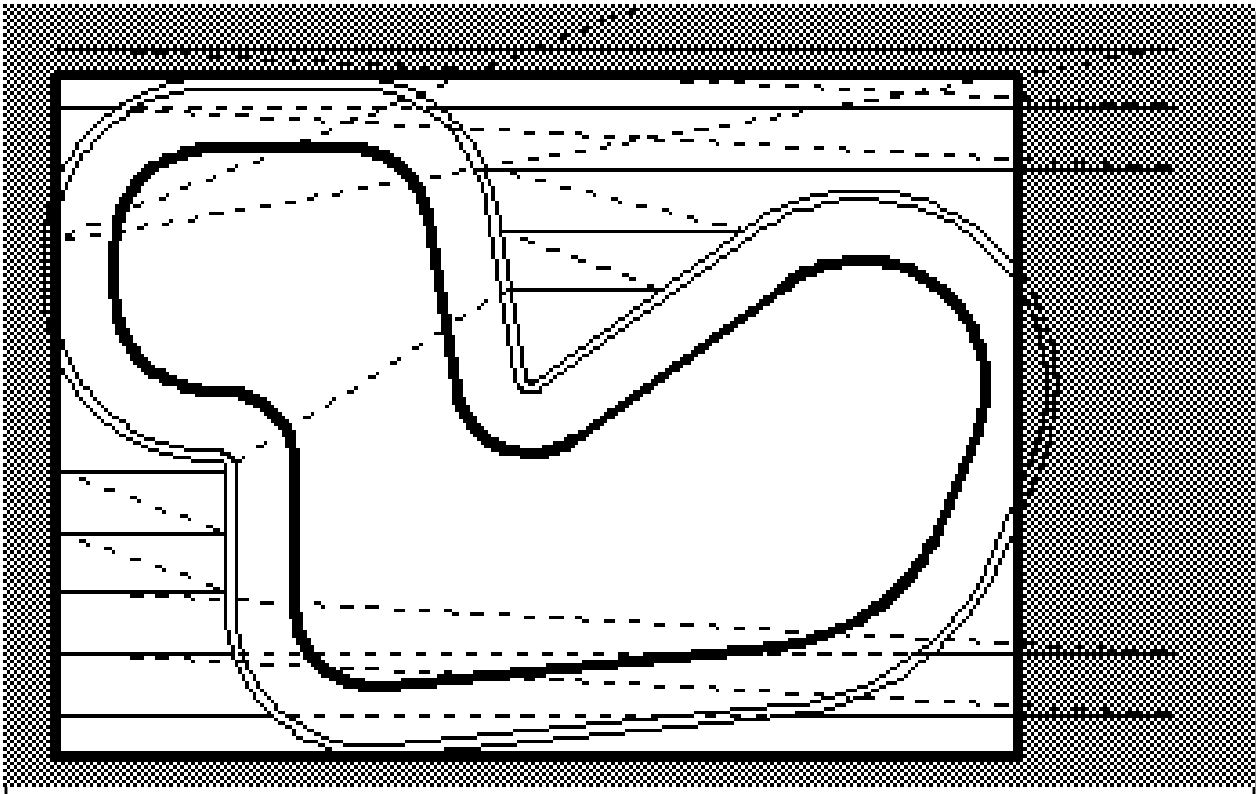
The cut starts at the top of the picture. The inner contour path is followed first. It is made by ramping the tool down into the part to full depth while following the path.

The horizontal cuts are made next to remove the material on the inside of the contour. Each cut spans the inner contour and is made from right to left so that conventional milling is done. The order in which the cuts are made is from top to bottom. After each cut the tool is withdrawn, moved to the beginning of the next cut and inserted rapidly. Note that there are no horizontal cuts in the neck of the figure. This is because cuts whose length is less than a tool radius have been suppressed; the material they would cut has already been removed.

If the pocket is deep, the entire pattern of horizontal cuts is repeated several times at increasing depths to reach the final depth.

The last step is to make a finish pass at full depth around the outer contour outline.

Figure 9. Side Contour Tool Path



This tool path drawing, which is a bit image taken from the screen, shows the tool path used to cut a particular side contour. The rectangle is the part, and the heavy line marks the outline of the side contour. A light solid line is is drawn when the tool is cutting. A light dotted line is drawn when the tool is in the air above the part.

The cut starts at the left of the picture, slightly below the top. The outer contour path is followed first. It is made by ramping the tool down into the part to full depth while following the contour path.

The horizontal cuts are made next to remove the material outside the outer contour. Each cut is made from right to left so that conventional milling is done. The order in which the cuts are made is from top to bottom. After each cut the tool is withdrawn, moved to the beginning of the next cut, and inserted rapidly. At the right, the tool is inserted outside the part by a tool radius plus 0.1 inch. At the left the tool is withdrawn just at the edge of the part. Cuts whose length inside the material is less than a tool radius have been suppressed; the material they would cut has already been removed. If the side contour is deep, the entire pattern of horizontal cuts is repeated several times at increasing depths to reach the final depth.

The last step is to make a finish pass at full depth around the inner contour outline.

VI. DATA EXECUTION MODULE SOFTWARE

1. INTRODUCTION

The software for the Data Execution module is all written in Franz Lisp, as is nearly all the rest of the VWS2 software. It is compilable, but is normally run interpreted rather than compiled since changes have been frequent, and, if bugs appear they are much easier to find and eliminate in uncompiled code.

In addition to the authors, portions of the code were written by three others. Mr. W. Timothy Strayer, a summer worker at NBS, had a hand in many earlier versions of the NC-coding functions. Mr. Alton Quist, a research associate from General Dynamics who worked at NBS for a year ending in June 1985 wrote most of the original version of the NC-coding system, although only a few lines of it remain. Dr. Edward Magrab, formerly an NBS employee, provided Monarch language versions of the three top-level zero-finding functions. He also wrote the three Monarch language zero-finding subroutines which are called by the top-level functions.

2. LISP FUNCTIONS

When it is run with all options off and the input process plan has already been enhanced, the Data Execution module uses eight core functions plus a few miscellaneous and property list manipulation functions (in addition to the functions provided by Franz LISP). When it is run with all options on and the input process plan is not enhanced, the module uses 19 core functions, 57 NC-coding functions, quite a few miscellaneous and property list manipulation functions and several hundred graphics, verification, and geometry functions. The core functions and the NC-coders are listed in Table 7. Half the core functions are in the exec2 directory and half in the proc2 directory. All of the NC-coders are in the exec2 directory. There are no functions in the exec2 directory which are not listed in Table 7.

Functions not listed in Table 7 are discussed in other papers written for AMRF documentation.

The function at the top of the hierarchy is "execute_plan". The real workhorse, though, is "execute_step", which makes data-driven function calls to over 50 of the top level verification, drawing, and NC-coding functions. The operation of these two functions is described earlier in this paper.

Table 7. Data Execution LISP Functions

CORE FUNCTIONS - EXEC2

add_feature
enhance1_plan
enhance_tool_parms
exec_header
execute_plan
execute_step
init_exec_plan
init_workpiece
precify

CORE FUNCTIONS - PROC2

add_extra_items
cull_steps
delete_step
enhance_step
insert_face_mill
insert_step
order_ops
pass_depth
print_plan
pull_tool_req

TOP LEVEL NC-CODERS - EXEC2

cbore_hole_nc
center_drill_nc
chamfer_in_nc
chamfer_out_nc
close_nc
contour_groove_nc
contour_pocket_nc
csink_hole_nc
face_mill_nc
groove_nc
hole_nc
init_nc
pocket_nc
set0_center_nc
set0_corner_nc
set0_z_nc
side_contour_nc
straight_groove_nc
tap_hole_nc
text_nc

NC-CODING SUBORDINATES - EXEC2

arc_length
cg_chunk
cg_go_backwards
cg_ramp
change_tool
compute_r_plane
cp_chunk
cp_cull
cp_first_pt
cp_int_gen
cp_int_order
cp_length
cp_next_init
cp_next_pt
cp_pick_angle
cp_ramp
cp_traverse
cs_chunk
cs_cull
cs_traverse
depth_loop1
depth_loop2
feed_rate
gets_and_puts
groove_chunk
letter_chunk
mill_letter
nc_hunk
nc_line
pocket_chunk
print_nc_file
print_nc_line
radial_stepover
rev_path_edoc
spindle_speed
straight_groove_chunk
sub_face_mill

REFERENCES

[JUN]
Jun, Jau-Shi; "The Vertical Machining Workstation Systems"; NISTIR 88-3890; National Institute of Standards and Technology; 1988; 65 pages.

[KRA1]
Kramer, Thomas R.; “Process Plan Expression, Generation, and Enhancement for the Vertical Workstation Milling Machine in the Automated Manufacturing Research Facility at the National Bureau of Standards”; NBSIR 87-3678; National Bureau of Standards; 1987; 56 pages.

[KRA2]
Kramer, Thomas R.; "Process Planning for a Milling Machine from a Feature-Based Design"; Proceedings of Manufacturing International Meeting; Atlanta, Georgia; April 1988; ASME; 1988; Vol. III, pp. 179 -189.

[KRA3]
Kramer, Thomas R.; "The Graphics Subsystem of the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards; NBSIR 88-3783; National Bureau of Standards; 1988; 27 pages.

[KRA4]
Kramer, Thomas R.; "Data Handling in the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 88-3763; National Bureau of Standards; 1988; 62 pages.

[KRA5]
Kramer, Thomas R.; "The vws_cadm User Interface in the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 88-3738; National Bureau of Standards; 1988; 110 pages.

[K&J1]
Kramer, Thomas R.; and Jun, Jau-Shi; "Software for an Automated Machining Workstation"; Proceedings of the 1986 International Machine Tool Technical Conference; September 1986; Chicago, Illinois; National Machine Tool Builders Association; 1986; pp. 12-9 through 12-44.

[K&J2]
Kramer, Thomas R.; and Jun, Jau-Shi; "The Design Protocol, Part Design Editor, and Geometry Library of the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 88-3717; National Bureau of Standards; 1988; 101 pages.

[K&S1]
Kramer, Thomas R.; and Strayer, W. Timothy; "Error Prevention in Data Preparation for a Numerically Controlled Milling Machine"; Proceedings of 1987 ASME annual meeting; ASME; 1987; PED-Vol. 25; pp. 195 - 213.

[K&S2]
Kramer, Thomas R.; and Strayer, W. Timothy; "Error Prevention and Detection in Data Preparation for the Vertical Workstation Milling Machine in the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 87-3677; National Bureau of Standards; 1987; 61 pages.

[LOVE]
Lovett, Denver; "The Vertical Workstation's Equipment Controllers"; NBSIR 88-3769; National Bureau of Standards; 1988; 59 pages.

[METC]
Machining Data Handbook, Metcut Research Associates, Inc.

[MONA]
Programming Manual for the Monarch VMC-75 with GE2000MC Controls; Monarch Cortland Publication Number PRG GE2000MC-4; Monarch Cortland; Cortland, NY; undated.

[NA&J]
Nakpalohpo, Ibrahim; and Jun, Jau-Shi; "Automated Equipment Program Generator and Execution System of the AMRF Vertical Workstation"; not yet published; 1987; 17 pages.

[RUDD]
Rudder, Frederick; "Operations Manual for the Automatic Operation of the Vertical Workstation"; NISTIR 89-4031; National Institute of Standards and Technology; 1989; 33 pages.