ABSTRACT

COIMBATORE ANNADORAI, LAKSHMAN NARESH. A Just-In-Time Compiler for Intelligent Manufacturing. (Under the direction of Dr. Xipeng Shen.)

Conventional Computerized Numerical Control (CNC) machines are operated using G-Code part programs. These machines require constant monitoring and human interaction during machining process to detect and rectify abnormalities. Making changes to the machining process is time consuming and a tedious task because the CNC machine operator has to have prior knowledge of G-Code programming language. G-Code is a low-level and machine specific programming language. So, G-Code programs do not contain most of the information from CAD/CAM phase of the manufacturing process. And also a G-code program generated for a specific machine cannot be used in a machine manufactured by a different CNC vendor.

This work presents a framework that uses a machine independent programming language as input, does not require CNC operators to have prior knowledge of G-Code programming language, does not require human interaction during machining process, enables real-time remote monitoring of the machining process, performs optimization on the input to a CNC machine based on the results from the previous runs, allows users to perform their exploration on historical feedback data set, and allows users of the framework to write third-party applications and optimization algorithms that perform optimizations on the input.

A Just-In-Time Compiler for Intelligent Manufacturing

by
Lakshman Naresh Coimbatore Annadorai

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2016

APPROVED BY:

Dr. Vincent W. Freeh

Dr. Binil Starly

Dr. Xipeng Shen
Chair of Advisory Committee

DEDICATION

To my parents and brother.

BIOGRAPHY

The author was born in Coimbatore, Tamil Nadu, India. He received a Bachelor's degree in Information Technology from Sri Ramakrishna Engineering College, Coimbatore, India in 2011. He joined HCL Technologies, India as a Software Developer and worked there for three years. He was admitted to NC State in the fall of 2014 as a Master's student to the Department of Computer Science. He worked with Dr. Vincent W. Freeh from January 2015 till May 2015 and Dr. Xipeng Shen as a Research Assistant from September 2015.

ACKNOWLEDGEMENTS

First and foremost I would like to thank my advisor Dr. Xipeng Shen for his guidance in my thesis work and his amazing lectures in Compiler Construction which have inspired me to continue my graduate studies in that area of expertise. His humble attitude and helpful nature made my first steps in academic research a fulfilling and exciting experience. I am lucky to have had the chance to work with Dr. Shen. He is also one of the best instructors that I have had, he enjoys teaching as well as research and I have learned a lot from his courses.

I would also like to thank Dr. Vincent W. Freeh and Dr. Binil Starly for being part of my graduate committee. Dr. Vincent W. Freeh is an incredible instructor and blogger. His course on Operating Systems Principles impressed me by his expertise in the field and taught me a lot. Dr. Binil Starly and Dr. Yuan-Shin Lee provided a lot of insights on the manufacturing domain and helped whenever I got stuck. Without their help it would not have been possible to complete my research work.

I would also like to thank Atin Angrish. Atin Angrish helped me a lot with the project and manufacturing domain. I would like to thank Dr. Shen's research team for the inspiration and support they provided.

Finally, I would like to thank my family and friends who have supported me in all my endeavors and encouraged me to pursue graduate studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

The first wave of Industrial Revolution began in the late 18th century. Currently, Industrial Revolution is in its fourth wave, characterized by a range of new technologies that are integrating the physical, digital and biological worlds, and impacting all disciplines, economies and industries. Fourth Industrial Revolution is also known as an era of **cyber-physical systems** (CPS). A CPS is a system composed of physical entities such as mechanisms controlled or monitored by computer-based algorithms. Today, a precursor generation of cyber-physical systems can be found in areas as diverse as aerospace, automotive, chemical processes, civil infrastructure, energy, healthcare, manufacturing, entertainment, consumer appliances, and transportation.

Cybermanufacturing is a concept derived from cyber-physical systems. Basically, it refers to a modern manufacturing system that offers an information-transparent environment to facilitate asset management, provides reconfigurability, and maintains productivity. The back-bone of cyber-manufacturing is claimed to be the intelligent machines. The conventional approach to making a machine *cyber-enabled* is to outfit the machine with an array of multi-modal sensors which are then integrated to the network and enterprise above it. This approach will make development work cumbersome. Alternate approach is to use the feedback from the machines. Almost all industrial machine vendors have closed hardware and software architecture which leads to development work remaining in-house. Vendor lock-in mode makes it difficult for outside software applications to be

written to obtain the feedback data, potentially limiting fast-paced innovation in the cyber-physical manufacturing realm. Hence, this work presents a framework that acts as a middleware between the manufacturing system and the user or operator. This is similar to Android - a common middleware software platform for many of the leading smart-phone manufacturers.

Most of today's Computerized Numerical Control (CNC) machines are operated using G-code programming language because G-code is supported by most of the CNC vendors. The controller in the CNC machines are closed; the internal operating method, current machine state, and sensor data are not exposed to the users of the CNC machines. Vendor specific controllers make G-code the universally accepted input for CNC machines. And also these controllers do not allow users to read the machine data or create their optimization model based on their exploration, and hinder the ability to remotely monitor the machining process, the sensor data, or the tool condition.

Hypothesis

This thesis hypothesize that it is possible to build an open source CNC controller that drives a CNC using a high-level programming language, performs optimization on the input based on the results from previous runs, and exposes APIs, which enable users to monitor the machining process, to perform exploration on the historical feedback data set, and to optimize input using third-party optimization algorithms.

Contributions

Just-In-Time (JIT) Compiler for Intelligent Manufacturing is a system that overcomes the aforementioned shortcomings of a controller in a vendor specific CNC. This system accepts a high-level language as input and a guideline file that specifies the optimizations to be performed on the input, uses an open-source controller to drive a CNC machine, performs optimizations on the input using the machining feedback data and sensor values from the previous runs, and exposes application programming interfaces (APIs) for machining feedback data, sensor values, and the guideline file. JIT Compiler for Intelligent Manufacturing enables users to remotely monitor the machining process by writing third-party applications using the APIs for the machining feedback data and sensor values, optimize the input by writing third-party optimization algorithms using the APIs for the guideline file, and explore the historical feedback data set using the APIs for the machining feedback data and sensor values.

CHAPTER

2

RELATED WORK

Traditional Just-In-Time Compiler in JVM

The Just-In-Time Compiler for Java Virtual Machine aims to reduce the execution time by performing feedback-directed optimization at runtime and limiting the overhead of runtime optimization. There are two types of feedback-directed optimizations: online and offline feedback-directed optimization. Offline feedback-directed optimizations are optimizations in which the application behavior is captured from a prior execution of the program. This approach fails in scenarios where 1) it is impractical to collect a profile prior to execution, or 2) the application's behavior differs from its behavior during the profiling run[Arn02]. Online feedback-directed optimization avoids the drawbacks of offline profiling by performing optimizations based on the profiles collected during runtime. JIT compiler for JVM uses online profiling to identify hot methods and perform optimizations on the basic blocks in the methods or in-lines frequently executed methods to expose opportunities for additional optimizations. JIT Compiler for JVM achieves this by compiling a java source code into byte-codes using a java compiler and then interprets the bytecode to machine instructions at runtime. The bytecode provides the opportunity to perform optimizations during runtime because bytecode contains high-level information that can be used to perform optimizations.

Optimizations performed on a java program are much more complex than the optimizations

performed on a STEP-NC file because java programs have branches, loops, different control flows, and data flows. The control path taken by a java program varies based on the input to the program so the offline profile collected for a java program might not provide the tailored optimization for the subsequent execution environment. In case of STEP-NC programs, the execution environment varies as the machining conditions varies. So the offline profile collected from an execution is valid for the subsequent run in the same machine until the tool wears off or the workpiece material changes. The online feedback-directed optimization of a STEP-NC program is the real-time optimization of the machining operation and it has to be done quick enough to propagate the changes before the next set of machining instructions are executed. So, it has a hard deadline on the optimization time. The type of optimizations performed on a java program varies from the optimizations performed on a STEP-NC program. The optimizations performed on a java program aims to reduce the execution time and space. But the optimizations performed on a STEP-NC program focuses to minimize machining time (Time-Critical) or optimize surface quality (Quality-Critical). Time-Critical machining operations are often for roughing purposes whereby increasing the material removal rate is one of the main goals and Quality-Critical machining operations are often used for finishing purposes where surface quality is of the main concern[RX13].

STEP-NC enabled machine condition monitoring system

Over the past years several research works have been carried out to use STEP-NC to drive CNCs, perform online inspection, and optimize the machining parameters. The authors of the research work [RX13] propose a STEP-NC enabled Machine Condition Monitoring (MCM) system that accepts a STEP-NC ISO 14649 data model file as input, performs initial feed-rate optimization, converts the STEP-NC program in to a Canonical Machining Commands (CMC) part program using an interpreter that is capable of handling optimization data, drives a CNC using a customized machine tool controller that accepts the CMC part program, monitors and records the machining parameters over the network using MTConnect, visualizes and evaluates the machining parameters to obtain another set of optimum parameters, calculates appropriate feed-rates using the optimum parameters for subsequent machining operations, and updates the STEP-NC file with the calculated feed-rate value.

Figure 2.1 shows the system architecture of the STEP-NC enabled Machine Condition Monitoring system. The initial feed-rate optimization on the STEP-NC file is carried out by optiSTEP-NC subsystem. This subsystem performs an offline feed-rate optimization on the STEP-NC file based on two criteria, minimum machining time and optimal surface quality, and generates a CMC part program from the STEP-NC file. The CNC is controlled by the AECopt subsystem that provides three

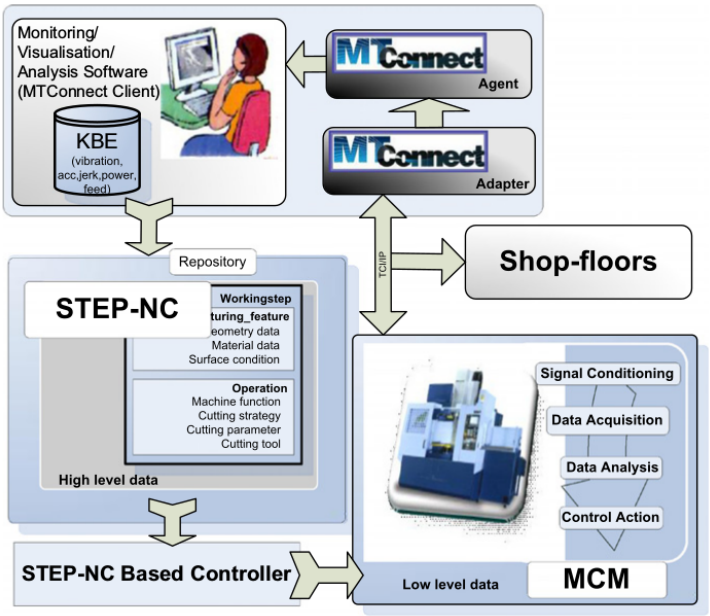


Figure 2.1 STEP-NC enabled MCM system architecture[RX13].

functions: (i) understanding of machine tool behavior and capabilities through MCM, (ii) adaptive control of optimized machine parameters and (iii) execution of a CMC part program. MTConnect is an open protocol and XML-based standard for data integration which can act as an enabler for higher level standards [Vij08]. Its architecture can be easily deployed and retrofitted to the existing machines, hence providing flexibility and portability functions for various types of machining environments. KBE based-MTConnect is responsible for obtaining machining know-how. Optimization is performed before, during or after machining operations, based on the data collected and monitored such as machining vibration, acceleration and jerk, cutting power and feed-rate[RX13]. This research work proposes a system that is capable of optimizing only the feed-rate based on the evaluation proposed by the work. This system requires human intervention during the machining process and does not allow users to remotely monitor the machining process, perform profiling, or build third-party optimization algorithms customized to the manufactured product.

SPAIM

The quality of a machined part is affected by the capability of each machine tool component, from the cutting tool to NC controller. The main errors can be classified into tool deflection, machine tool

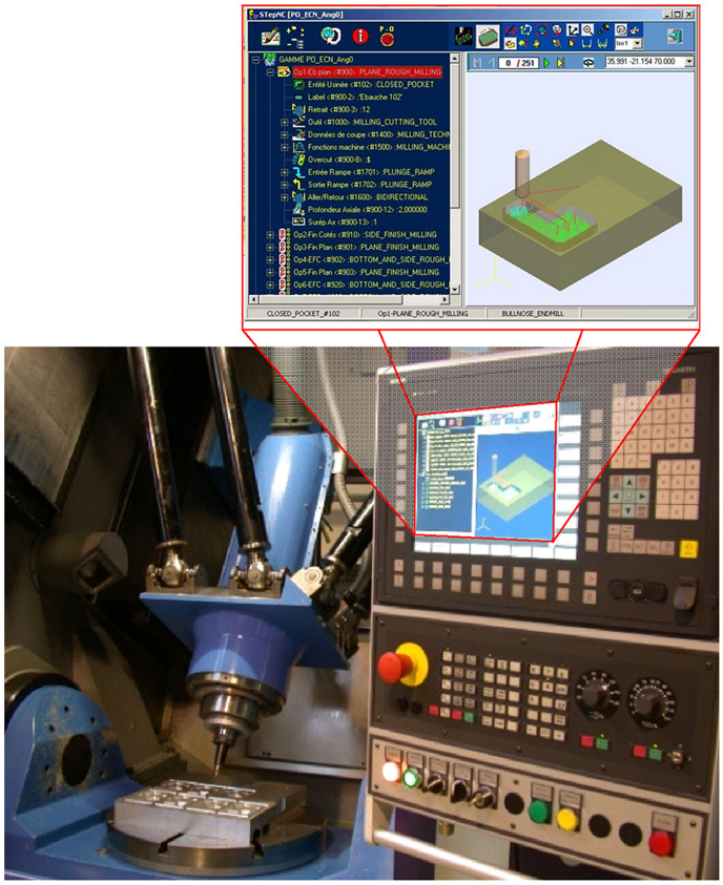


Figure 2.2 SPAIM platform at IRCCyN[Rau12].

and spindle vibrations, geometric errors, thermal effects, chatters, control errors, etc. Machining forces play a very important role in the machining process as well, by reducing productivity to avoid tool breakage, tool and part deflection, power consuming, etc. Several works have been done to improve the knowledge to control the phenomenon linked to these errors [Rau12]. One such work [Rau12] developed and implemented a NC controller that estimates online process data and optimizes machining parameters and tool-paths to eliminate the causes of these errors or to compensate the results. This NC controller is called SPAIM: a STEP-NC platform for advanced and intelligent manufacturing. SPAIM is made of following parts: SPAIM Human/Machine interface (HMI), Delphi master and execution modules, Tool-path generation module, CAD reconstruction module, and Tool and machine tool functional model.

Figure 2.2 shows SPAIM platform at IRCCyN. The HMI is displayed on the NC controller screen,

a 3D visualization of the manufacturing data (CAD geometry features, tool-paths, cutting tools, etc.) and a tree view of the STEP-NC file. This interface makes the link between the user and computation modules of SPAIM. At every stage, the user fully supervises and controls the operations and can modify every undesired parameter value directly within the tree view. Before any machining action, a tool-path visualization and user validation is needed for safety reasons.

The Delphi master module resides in the PC part of the CNC controller and the execution module is located on an external PC linked via an intranet connection. These two modules can be seen as a client and a server. The execution module reads and analyses STEP-NC file to extract feature and process data. It builds a tree for geometry and machining parameters and sends the requested information for processing in the tool-paths generation module and in the simulation module. Tool-paths are generated on the external PC using a tool-path generation module from Delcam PowerMILL software. The execution module is linked with the master module for direct execution on the CNC controller after user visualization and validation. CAD reconstruction module is a Delphi-based automatic tool that rebuilds the CAD geometry from the feature description in the STEP-NC file. Manufacturing features data are extracted from the STEPNC file. After the analysis, the corresponding commands are sent to the Delcam PowerSHAPE CAD software to reconstruct the geometry. The obtained CAD model is not only used by PowerMILL software in the tool-path generation module but also by the HMI to provide feedback from the STEP-NC file to the CAD model for 3D visualization.

The STEP-NC file contains information concerning manufacturing tools. For each manufacturing operation, it is possible to describe the selected tool and all its components. If the corresponding tool is not available on the shop floor, quick modifications can be done on the HMI. The related tool-paths are automatically regenerated[Rau12] and it is validated by the user before machining the product. This work focuses on optimization of the tool-path based on the tools available in a machine before machining to avoid tracking errors or tool deflection. This approach requires human intervention to validate the generated tool-path.

IMSRAC

An in-process multiple-regression-based surface roughness adaptive control (IMSRAC) system in end milling operations was researched and developed by the authors of the research work [ZC07]. A multiple regression algorithm was employed to establish two subsystems: the in-process multiple-regression-based surface roughness evaluation (IMSRE) subsystem and the in-process multiple-regression-based adaptive parameter control (IMAPC) subsystem. These systems include not only machine cutting parameters such as feed rate, spindle speed, and depth of cut, but also cutting

force signals detected by a dynamometer sensor. When the finish milling process starts, the IMRSE subsystem predicts the actual surface roughness based on the cutting parameters and cutting force signals using a multiple-regression model. This model was developed using experimental data from multiple runs of the product under different combinations of cutting parameters. The IMAPC subsystem calculates the delta between the predicted surface roughness and the desired surface roughness, if the predicted surface roughness is greater than the desired surface roughness. The calculated delta in the surface roughness is used in a multiple-regression model to predict the change in the feed-rate. The predicted change in the feed-rate is applied to the machining operation in the subsequent run to meet the surface roughness requirements. The multiple-regression model to predict the surface roughness from this work is used in the thesis to build a part of the optimization module of the JIT Compiler for Intelligent Manufacturing prototype. The delta between the predicted surface roughness and the desired surface roughness is used to calculate the required percentage of change in the force exerted by the tool. The percentage of change in the force is used to calculate the required change in depth of cut, which is updated in the guideline file to reflect the optimization information.

CHAPTER

3

BACKGROUND

CAD/CAM

CAD/CAM is a term which means computer-aided design and computer-aided manufacturing. It is the technology concerned with the use of digital computers to perform certain functions in design and production. Computer-aided design (CAD) can be defined as the use of software to assist in the creation, modification, analysis, or optimization of a design [Lal08]. CAD involves creating computer models defined by geometrical parameters. These models typically appear on a computer monitor as a three-dimensional representation of a part or a system of parts, which can be readily altered by changing relevant parameters. CAD systems enable designers to view objects under a wide variety of representations and to test these objects by simulating real-world conditions [Inc13]. CAD output typically includes precise dimensions, tolerances, and even material requirements; CAD is frequently integrated with computer-aided engineering (CAE).

Computer-aided manufacturing (CAM) is the use of software to control machine tools and related ones in the manufacturing of workpieces. CAM may also refer to the use of a computer to assist in all operations of a manufacturing plant, including planning, management, transportation and storage. Its primary purpose is to create a faster production process and components, and tooling with more precise dimensions and material consistency, which in some cases, uses only the

required amount of raw material (thus minimizing waste), while simultaneously reducing energy consumption. CAM is a subsequent computer-aided process after computer-aided design (CAD) and sometimes computer-aided engineering (CAE), as the model generated in CAD and verified in CAE can be input into CAM software, which then controls the machine tool[wik16a].

Computerized Numerical Control

Numerical control (NC) is the automation of machine tools that are operated by precisely programmed commands encoded on a storage medium, as opposed to controlled manually by hand wheels or levers. Most NC today is computer (or computerized) numerical control (CNC), in which computers play an integral part of the control[wik16c].

In modern CNC systems, end-to-end component design is highly automated using computer-aided design (CAD) and computer-aided manufacturing (CAM) programs. The programs produce a computer file that is interpreted to extract the commands needed to operate a particular machine by use of a post processor, and then loaded into the CNCs for production. Since any particular component in a product might require the use of a number of different tools - drills, saws, etc., modern machines often combine multiple tools into a single "cell"[wik16c]. In a CNC, motion is controlled along multiple axes, normally at least two (X and Y), and a tool spindle that moves in the Z (depth). The position of the tool is driven by direct-drive stepper motor or servo motors in order to provide highly accurate movements, or in older designs, motors through a series of step down gears. The workpiece, the cutting tool, or both rotate rapidly, and the overall motion of the workpiece or the cutting tool is precisely controlled by stepper motors or other servo-type mechanisms, which are themselves controlled by the numerical control commands.

Elements of a CNC System

A CNC system consists of the following 6 major elements[Moh09]:

1. Input Device
2. Machine Control Unit
3. Machine Tool
4. Driving System
5. Feedback Devices

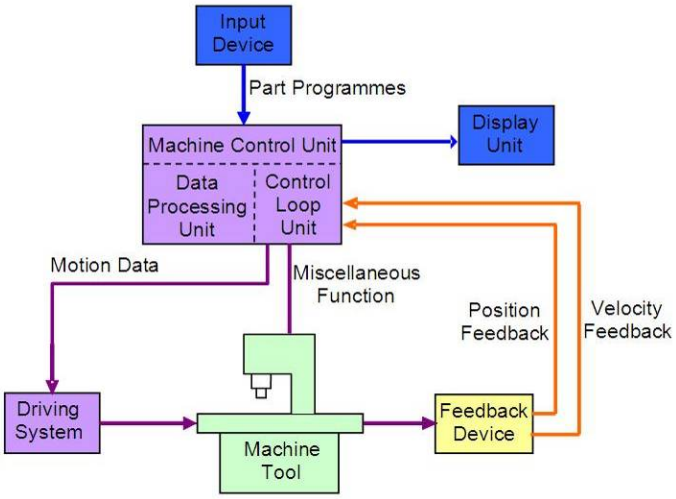


Figure 3.1 Working principles of CNCs[Moh09].

6. Display Unit

Input device can be a floppy disk drive, USB flash drive, serial communication, ethernet communication, or conversational programming that can be input to the controller via the keyboard. Input devices let the operator to input the numerical control or part programs to the CNC system. A part program is a set of instructions that control the operations of a CNC. The machine control unit is the heart of the CNC system. There are two sub-units in the machine control unit: Data Processing Unit (DPU) and the Control Loop Unit(CLU). On receiving a part program, the DPU firstly interprets and encodes the part program into internal machine codes. The interpolator of the DPU then calculates the intermediate positions of the motion in terms of BLU (basic length unit) which is the smallest unit length that can be handled by the controller. The calculated data are passed to CLU for further action. The data from the DPU are converted into electrical signals in the CLU to control the driving system to perform the required motions. Other functions such as machine spindle ON/OFF, coolant ON/OFF, tool clamp ON/OFF are also controlled by this unit according to the internal machine codes [Moh09].

Machine tool can be any type of machine tool or equipment. Driving system responds to the programmed instructions and uses electric motors that is coupled either directly or through a gear box to the machine leadscrew that moves the machine slide or the spindle. The electric motor can be a DC servo motor, AC servo motor, stepping motor, or linear motor. Feedback device updates the positional values and speed of the axes. This help CNC to operate accurately. Two types of feed back

devices are normally used, positional feed back device and velocity feed back device. The Display Unit serves as an interactive device between the machine and the operator. When the machine is running, the Display Unit displays the present status such as the position of the machine slide, the spindle RPM, the feed rate, the part programs, etc. In an advanced CNC, the Display Unit can show the graphics simulation of the tool path so that part programs can be verified before the actual machining. Other important information about the CNC system are also displayed for maintenance and installation work such as machine parameters, logic diagram of the programmer controller, error massages and diagnostic data [Moh09].

G-code Programming Language

G-code programming language is the conventional programming language of numerical controlled machine tools. ISO standard for G-code is ISO 6938. These codes were developed more than 50 years ago with little, if any, intelligence. The initial design of the codes was to hold a set of low-level data that are mostly step-by-step instructions to drive the earliest models of machine tools. This sequential programming language contains simple commands for single movement and switching operations but cannot support more complex geometries or logical structures. Since only limited control of the program execution is allowed during machining, it is difficult to change the program on the shop-floor. Information flow in G-codes was designed unidirectionally, i.e., from CAD to the shop-floor, and does not enable feedback of know-how from the shop-floor to the designer. As a result, this conventional way of NC programming is considered a bottleneck for achieving an intelligent machining environment[RX13].

There are several variations of G-codes and they are specific to a CNC vendor. These variants are generated by a CAM software using the specific post-processor for the variant. G-code is a machine specific and low-level programming language. G-code program generated for a specific CNC cannot be used in a CNC manufactured by a different CNC vendor. G-code is limited to axis motion, tool selection, machine spindle ON/OFF, coolant ON/OFF, and tool clamp ON/OFF commands. It does not contain most of the high-level information from CAD/CAM software and enough information about the part, material and stock. It is almost impossible to back feed the information from a CNC to the CAM/CAD software [KK09]. G-code has more than three thousand variations and post-processors.

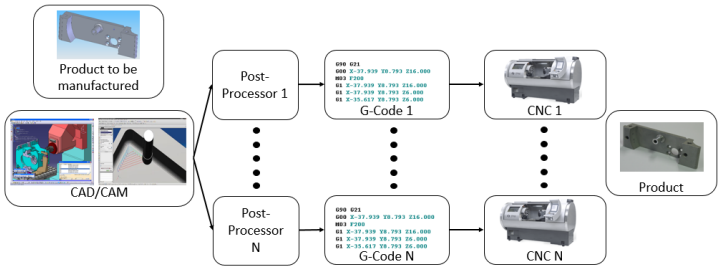


Figure 3.2 Conventional manufacturing process.

Conventional Manufacturing Process

Figure 3.2 depicts the conventional manufacturing process. A product is designed in a CAD software and detailed in CAM software. The output of the CAM software is sent to a post-processor to generate G-code part program for a specific CNC. The generated G-code part program is sent as input to a CNC, which reads the G-code part program and manufactures the product. The problem with this approach is that the G-code part programs have to be generated for each different type of CNCs on the shop-floor. If something were to go wrong during a manufacturing operation, the machine has to be stopped and the G-code part program has to be edited manually by the operator to reflect the correct manufacturing operation. Since all the machines on the shop-floor are manufacturing the same product and the G-codes are not the same for CNCs manufactured by different CNC vendors, operators have to stop the machines and manually edit the G-codes every time the product is manufactured. This is a cumbersome work and it will increase the production time. And also it requires the operators to have prior knowledge of the G-code programming language and the specific type of variants required for the CNCs on the shop-floor.

LinuxCNC

LinuxCNC is a free, open-source GNU/Linux software system that implements numerical control capability using general purpose computers to control CNCs. It can control up to 9 axes of a CNC using G-code as input. It has several GUIs suited to specific kinds of usage (touch screen, interactive development). Due to the need of fine grained, precise real time control of machines in motion, LinuxCNC requires a platform with real-time computing capabilities. It uses linux kernel with real time extensions (RTAI) or with RT-PREEMPT kernel using LinuxCNC's 'uspace' flavor of RTAPI[wik16b].

LinuxCNC Architecture Overview

LinuxCNC consist of four components: a motion controller (EMCMOT), a discrete IO controller (EMCIO), a task executor which coordinates them (EMCTASK) and several text-mode and graphical User Interfaces. At the coarsest level, LinuxCNC is a hierarchy of three controllers: the task level command handler and program interpreter, the motion controller, and the discrete I/O controller. The discrete I/O controller is implemented as a hierarchy of controllers for spindle, coolant, and auxiliary (e.g., estop, lube) subsystems. The task controller coordinates the actions of the motion and discrete I/O controllers. Their actions are programmed in conventional numerical control "G and M code" programs, which are interpreted by the task controller into Neutral Message Language (NML) messages and sent to either the motion or discrete I/O controllers at the appropriate times. The motion controller receives commands from user space modules via a shared memory buffer, and executes those commands in real-time. The status of the controller is made available to the user space modules through the same shared memory area. The motion controller interacts with the motors and other hardware using the Hardware Abstraction Layer (HAL)[lin16a].

STEP-NC

STEP-NC (Standard for the Exchange of Product data for Numerical Control) is an ISO standard machine tool control language. It provides an opportunity to overcome the obstacles of G-code especially in realizing intelligent machining operations. The main characteristic of STEP-NC is its high-level and object-oriented data structure. Unlike G-code where a part program is written to describe simple tool movements and functions, the STEP-NC interface is able to work with rich information such as manufacturing features, multiple operations such as finishing and roughing processes, machine tool capability, motor drive power, mechanical efficiency, machining strategy, cutting tool information, and workpiece properties. Since STEP-NC data model describes rich information, quality knowledge and data can be utilized on the shop-floor, which enables advanced optimization analysis to be conducted. Modifications on the shop-floor are possible and machining know-how can be preserved for designers and process planners, thus improving the communication link between design and manufacturing departments. By providing a complete and structured data model, no information is lost. Post-processors for machine-specific adaptations of NC programs are no longer needed. In addition, this rich information content results in higher flexibility enabling last-minute changes or the correction of technological values within the part program [RX13].

In STEP-NC data formats, the technological steps to manufacture a part are defined as a sequence of material removing operations. Each operation represents removing a chunk of material

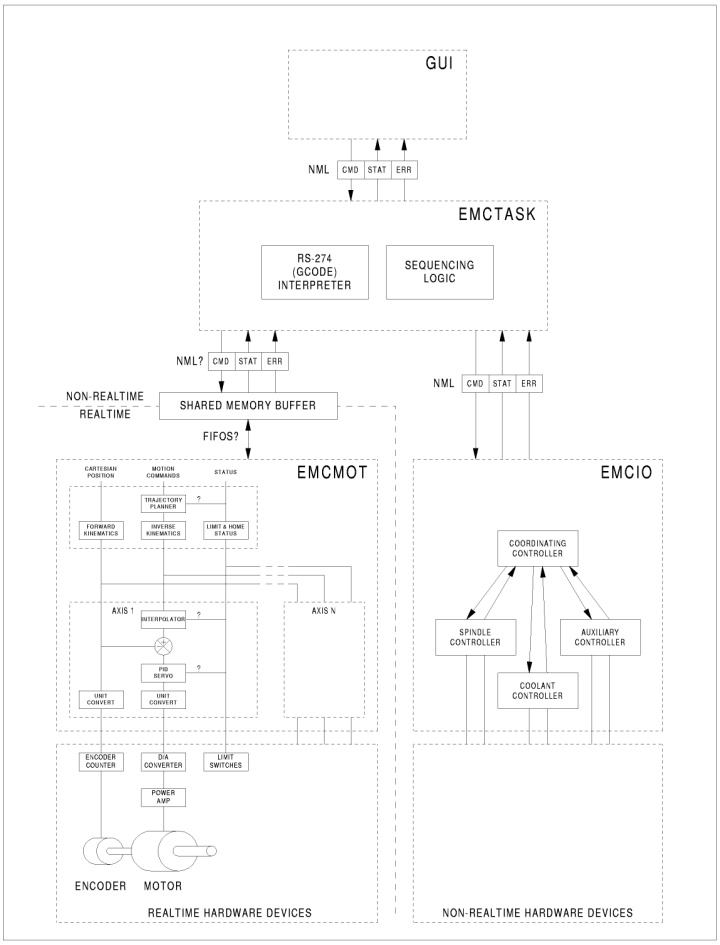


Figure 3.3 LinuxCNC Architecture [lin16a].

of the regular geometric shape. The standard supports all 2D geometric features (holes, pockets, grooves) and also 3D geometry (geometry confined by 3D surfaces). Each removal operation adds its geometric features, tolerances, type and size of a tool, etc. [KK09].

STEP-NC Formats

STEP-NC has two standards that cover the field of STEP-NC: ISO 14649 and ISO 10303-238. Each of them covers its own portion of data exchange between different steps in product development and manufacturing. ISO 14649 is intended for applications where CAM software has total access to all the data from the production, whereas ISO 10303-238 is intended for total integration of CAD, CAM and manufacturing [KK09].

ISO 14649

Its main purpose is to replace the ISO 6938 as the standard for NC program data formatting. The deficiencies of ISO 6938 are compensated by describing working process with different working steps which are needed to produce the geometry instead of tool movement trajectory. These working steps represent higher level geometry elements combined with all the process parameters needed in order to manufacture them. NC controller is responsible for translating these working steps into actual tool movement. This data structure also allows using the same STEP-NC program on any type of machine without the use of post-processors. ISO 14649 data format is very different from the old NC-code. It is well structured and object oriented [KK09].

Its main elements are [KK09]:

1. header
2. geometrical features of the part
3. technological working steps for manufacturing (each working step is composed by many feature/technology operations)

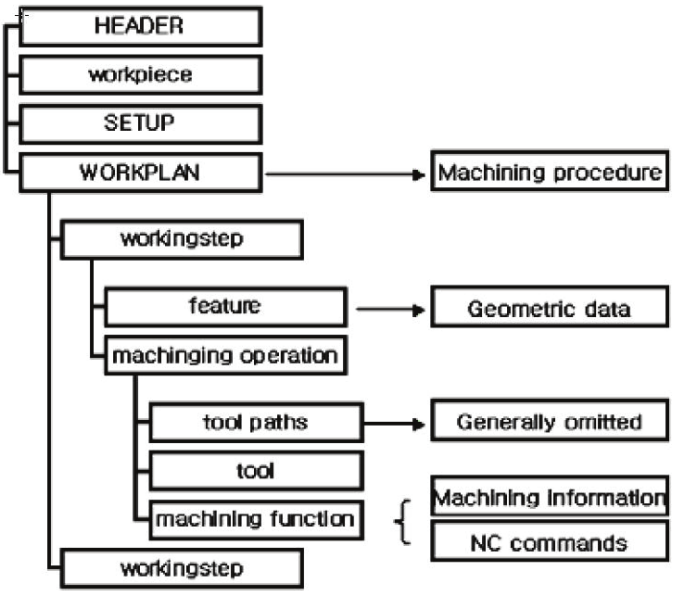


Figure 3.4 Graphical overview of ISO14649 data organization [KK09].

Table 3.1 A simple example of ISO 14649 data structure [KK09].

ISO 14694 data format	Explanation
Header	
#1=Project(WorkPlan#10);	Select Workplan at #10
#10=WorkPlan(#20,#35,#71,.....);	Workplan is consisted of: WorkStep1 at #20, WorkStep2 at #35,...
.	
#20=WorkStep1(,#21(Feature), #22(Manufacturing))	WorkStep1: manufacture geometry feature #21 using technological parameters #22
#21=Hole('Tap M6',,,,,,,,,);	geometry feature #21
#22=Drilling(#..(Tool),,,#..(Technology),#..;	technological parameters #22
.	
#35= WorkStep2 (.....);	WorkStep2
END-ISO-10303-21;	End of program

Table 3.1 shows how all the data in the file are written in plain text using special coding [KK09].

ISO 10303-238

ISO 10303-238 data model is also know as AP238 data model. The main difference in ISO 10303-238 when compared to ISO 14649 is that this data model also includes a 3D CAD model of the product. It is basically an upgrade of an ISO 14649 standard. The table 3.2 clearly shows that a large portion of AP238's data structure implementation was copied from previously released ISO 14649 [KK09].

The data model of AP238 is structured into three main parts [KK09]:

- part 3D representation
- working plan for manufacturing
- technological parameters for manufacturing

A part 3D representation uses the same data model as other STEP application models. A 3D part is described by using ISO 10303 standard data model, also all the information about owners and creation dates are included. This data can be created by any STEP compatible CAD program [KK09].

Table 3.2 Origins of different elements of ISO 10303-238 data model [KK09].

Element of AP238 data model	Element origin
Part 3D geometry	All STEP application protocols share the same data model
Dimensions and tolerances	STEP GT&D
Measures Project data Stock data Geometry features Working steps sequence tool-paths	ISO 14649 Part10
Milling operations Milling tools	ISO 14649 Part11 ISO 14649 Part111
Lathe operations Lathe tools	ISO 14649 Part12 ISO 14649 Part121

Besides the 3D representation the standard foresees that the part is assembled by using interconnected features. Each feature is a distinct geometric shape of a special type: 2.5D element (hole, pocket, groove...), transitional element (radius, chamfer...), and element of repeating (mirror, copy...), region element (3D surfaces), lathe element (grooves...) and others. Some of 2.5D features are shown in the figure 3.5 [KK09].

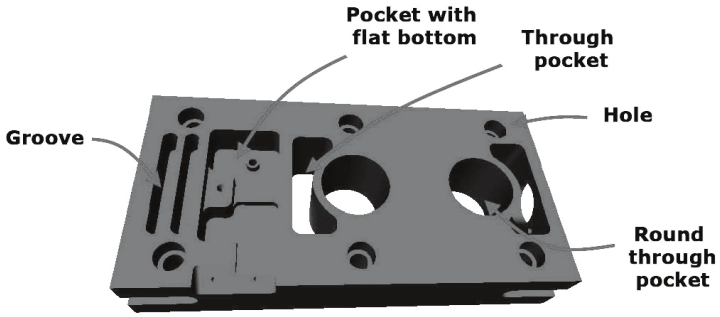


Figure 3.5 A part with 2.5D features [KK09].

Working plan for manufacturing is "the heart" of the AP238 application protocol. It contains information about manufacturing steps which is needed to manufacture the part. Figure 3.6 shows the plan, which is divided into several working steps. They are instructions that designate which strategies and parameters are used to manufacture each feature.

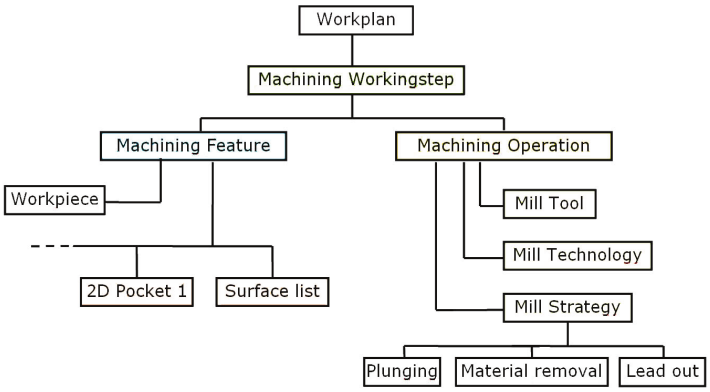


Figure 3.6 Manufacturing working plan [KK09].

STEP-NC DLL

STEP-NC DLL is a library developed by STEP Tools Inc. It connects design and manufacturing systems. Applications that have been implemented using the DLL include just in time simulation and verification, integrated machining and inspection, five axis cutter compensation, and supply chain traceability [Har06]. But this library exposes limited APIs that allows applications to read and validate the data. It is closed sourced and cannot be used to build the Just-In-Time Compiler for Intelligent Manufacturing framework.

ISO14649 toolkit

ISO14649 toolkit is built at NIST for programming with ISO 14649 Parts 10, 11, and 111. These parts are data models to be used for controlling a machining center. The ISO14649 toolkit has a compiler for 3-axis machining. It reads STEP Part 21 files corresponding to the data model described in Parts 10, 11 and 111 of ISO 14649 and uses a Lex-Yacc parser to generate a parse tree. Later this parse tree is loaded in an internal data structure that represents the contents of the STEP-NC file. It uses the information from the data structure to determine the working steps, geometry of a working step,

and operations to be performed for a working step. It uses an internal data structure to monitor the current tool and its dimension, tool position based on the previous commands generated, coolant status, spindle speed, feed-rate, etc.,. Finally, the toolkit generates canonical machining commands using the information from these data structures. The compiler implements only a modest portion of Parts 10, 11, and 111. The compiler includes tool-path generators for rough and finish plane milling, rough and finish rectangular pocket milling, drilling, multistep drilling, reaming, center drilling, countersinking, and counterboring [NIS09].

The object-oriented representation of machining process and the geometry of the working steps in a STEP-NC file enables the ISO14649 toolkit to optimize the tool-path during runtime of the framework. The authors of the ISO14649 toolkit describe it as an interpreter but it actually is a compiler because the toolkit generates an IR, provides the opportunity to optimize the input during runtime, and does not execute the input program instead it generates a different form of representation of the program. The canonical machining commands are atomic commands. Each command produces a single tool motion or a single logical action. There is no standard for canonical machining commands and it might vary based on the implementation. LinuxCNC internally uses canonical machining commands to communicate the operation to be performed in the CNC [Pro97].

CHAPTER

4

DESIGN AND IMPLEMENTATION

Just-In-Time Compiler for Intelligent Manufacturing Framework

Today's CNCs have closed source in-house controllers and use G-code part programs as input. These limitations restrict the users of CNCs to use G-code part programs as input. And also the users can not use the feedback data from a vendor specific CNC. Just-In-Time Compiler for Intelligent Manufacturing framework is designed to enable users to use a high-level language as input, use the built-in optimization module to optimize the input, use the machining feedback data and sensor values from a CNC to write their applications to optimize the input, remotely monitor the machining process, and perform exploration on historical feedback data set.

Figure 4.1 shows the JIT Compiler for Intelligent Manufacturing framework. The input to the framework is a high-level language program that contains most of the product information from CAD/CAM software, enables optimization on the input, does not require post-processor, and allows users to use the same program across CNCs manufactured by different CNC vendors. Controller is an open-source software that acts as an interface between the user and a CNC. Controller accepts the input, performs optimizations on the input based on the optimization information from either the optimization module or a third-party application, simulates tool-path or controls a CNC, reads the sensor values, stores the feedback machining data and sensor values in a database, and generates

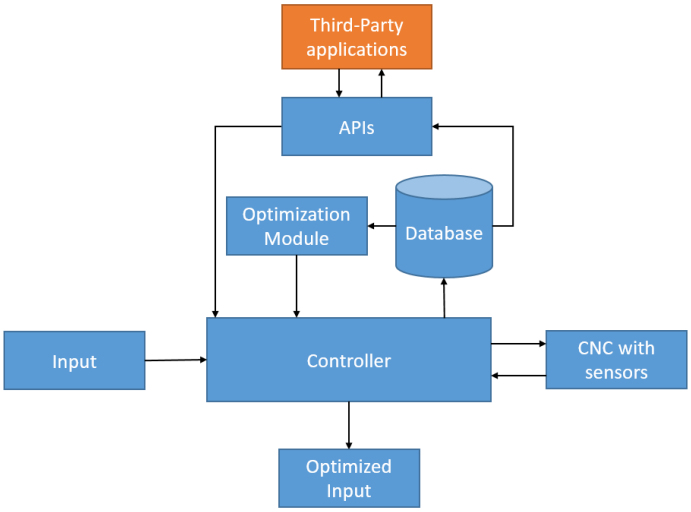


Figure 4.1 JIT Compiler for Intelligent Manufacturing Framework.

optimized input file that can be used in the subsequent runs of the same product in the same or different CNC. Optimization module reads the machining feedback data and sensor values, determines the optimization to be performed on the input, and sends this information to the controller. Application Programming Interfaces (APIs) enable users to write their applications which use the machining feedback data and sensor data to remotely monitor the machining process, to build an optimization module that sends the optimization information to the controller, and to perform exploration on the historical feedback data set.

Prototype Design

This work presents a prototype of the Just-In-Time Compiler for Intelligent Manufacturing framework. Major parts of the proposed Just-In-Time Compiler for Intelligent Manufacturing framework are designed and implemented in the prototype. Figure 4.2 depicts the system architecture of the prototype. The product to be manufactured is designed in a CAD software where the geometry of the product, features in the product, geometry of the features, working steps in each feature, and operations in a working step are modeled. The output of the CAD software is passed as input to a CAM software. In the CAM software, the tool-path and cutting parameters are modeled and the output is exported as a STEP-NC file. The STEP-NC file generated by the CAM software is the input to the framework. The input is fed to the ISO14649 toolkit along with a guideline file and a tools file by the user. The guideline file is specific to a product and a machine; each product

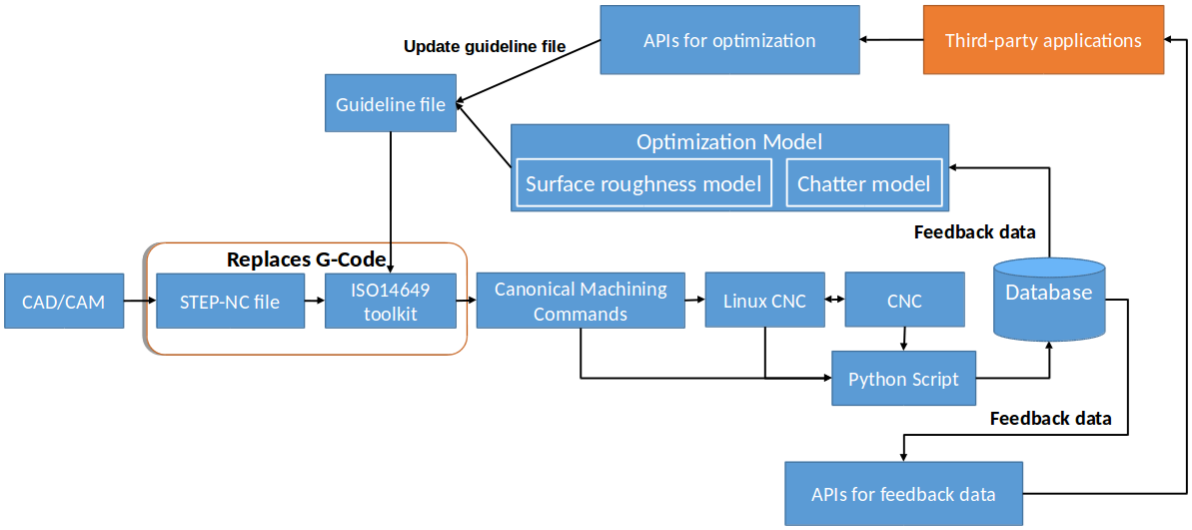


Figure 4.2 System architecture of JIT Compiler for Intelligent Manufacturing prototype.

machined in a CNC will have its guideline file specific for that machine. The guideline file contains optimization information for the product from either the optimization module or a third-party application. ISO14649 toolkit [NIS09] processes the STEP-NC file, guideline file, and tools file to generate appropriate canonical machining commands. The canonical machining commands file is loaded in to LinuxCNC [lin16b] by the user. LinuxCNC reads the canonical machining commands file one line at a time and executes the commands. The output of the execution is displayed in the simulation window of the LinuxCNC software or is used to drive a CNC. The ISO14649 toolkit and LinuxCNC together acts as the controller of the framework.

LinuxCNC generates a lot of real-time machining feedback data that provide real-time information about the machining operation currently being performed. These data are captured by a python script that maps the feedback data and sensor values to a working step and stores the data in real-time in a database. Thus the feedback from the CNC is store in a database in real-time. The prototype exposes application programming interfaces (APIs) that enable users to query the feedback data during machining process and after the product has been manufactured. These APIs provide the opportunity to remotely monitor the machining process in real-time and to perform exploration on the historical feedback data set. And also the prototype has a built-in optimization module that updates the guideline file with optimization information and has APIs to access the guideline file. User can use the built-in optimization module to perform optimizations on the STEP-NC file. This requires the user to input the sensor values and working step information to the built-in

optimization module. The optimization module executes an optimization algorithm on the input and updates the guideline file with optimization information for the working step. User can also build a third-party optimization module, which reads the feedback data using the APIs exposed for the feedback data, executes an optimization algorithm, and updates the guideline file using the APIs exposed to access the guideline file. In both the cases, the guideline file is updated with optimization information. Thus optimization information either from the optimization module or from a third-party application is communicated to the controller via the guideline file. This file is used in the subsequent executions of the same product by the ISO14649 toolkit to generate optimized canonical machining commands.

Assumption

The STEP-NC data model accepted by the prototype is ISO 14649 data model. This data model is not supported by CAD/CAM software because it is superseded by STEP-NC AP238 standard. The prototype is designed to accept ISO 14649 data model because there is no other tool similar to ISO14649 toolkit to generate canonical machining commands from STEP-NC AP238 standard. And also the STEP-NC AP238 standard is very complex to be used to design and implement a compiler that accepts this standard and generates canonical machining commands. The STEP-NC standard is evolving and contains most of the high level details from CAD/CAM software. The framework is flexible enough to replace the ISO14649 toolkit with a compiler that accepts the latest STEP-NC standard and generates an intermediate representation that is accepted by LinuxCNC.

Prototype Implementation

Controller

ISO14649 toolkit

ISO14649 toolkit [NIS09] reads a STEP-NC file and tools file, and generates canonical machining commands. But it cannot perform optimization on the input to generate optimized canonical machining commands. To generate optimized canonical machining commands, ISO14649 toolkit is modified to read a guideline file that specifies the optimizations to be performed. The guideline file can be empty or can contain the below structure(s). The structure is defined one per working step and specifies the optimization to be performed for a working step. The structure has two fields, first field is an id field that specifies the working step id and second field is a reduce_by field that represents the value by which the existing depth of cut should be divided. Depth of cut is the depth

by which a tool cuts the work piece. Working steps will have different depth of cuts and they are defined in the CAD/CAM phase of the product manufacturing. The actual depth of a working step is a multiple of the depth of cut for the working step. During a pass the tool removes the material in the specified tool path at a given depth of cut. So, the number of passes required to complete the working step is the multiplier by which depth of cut is multiplied to get the actual depth of the working step. For instance, if the reduce_by value is 2 for a working step, the existing depth of cut for the working step will be reduced by half. A working step has one or more operations and an operation has multiple passes. If the depth of cut is reduced by half, the number of passes for the operation increases by 100%, the number of passes for the working step increases by 100% and the number of canonical machining commands for the working step increases to reflect the number of passes. Thus the amount of material removed in a single pass is reduced by half, the force exerted on the tool is reduced, and the production time is increased. ISO14649 toolkit is modified to change the depth of cut for working steps based on the optimization information from the guideline file. So, the number of canonical machining commands generated for a working step varies based on the reduce_by value for the working step in the guideline file.

```
AXIAL_CUTTER_DEPTH_REDUCTION
{
  id = ""
  reduce_by = 2
}
```

The guideline file is read by a parser and a scanner customized to read the structure. The scanner generates an intermediate representation that contains the contents of the guideline file. The IR is passed to a LL(1) grammar that is designed for the structure. The grammar loads the optimization information in to an internal data structure. When the ISO14649 toolkit processes a working step from a STEP-NC file to generate canonical machining commands, the internal data structure containing the optimization information is checked for the working step id. If the internal data structure contains the working step id, the reduce_by value from the data structure for the working step is used to determine the current depth of cut for the working step. Based on the depth of cut, the number of passes required to complete the working step is determined and the canonical machining commands are generated to reflect the current depth of cut.

The canonical machining commands file generated by ISO14649 toolkit does not contain working step information. These information are useful to map a line in the canonical machining commands file to a working step. So, ISO14649 toolkit is modified to generate working step information along with the canonical machining commands. These information provide details about where a working

step begins. They are generated as comments in the canonical machining commands file because LinuxCNC does not require the working step information and it ignores comments. But these comments are read by a python script to map feedback data to a working step id.

LinuxCNC

LinuxCNC [lin16b] accepts only G-code part programs. But internally LinuxCNC converts a G-code into a canonical machining command to simulate tool-path or control a CNC. And also it has a module called canterp that allows the user to load canonical machining command program for a 9-axis CNC. However, this module is not fully developed to simulate tool-path or control a CNC. But the canonical machining command program generated by ISO14649 is for a 2.5-axis CNC. To enable LinuxCNC to accept canonical machining command program for a 2.5-axis CNC and simulate tool-path or control a CNC, the canterp module is modified.

Database

LinuxCNC has python APIs to read the machining feedback data generated in real-time during a machining operation. These data provide information about the machining operation currently being performed. Python script module in the prototype uses these APIs to fetch the machining feedback data at specific time intervals and uses arduino APIs to read the sensor values from the sensors attached to the system. And also it reads the working step information from the canonical machining commands file generated by the ISO14649 toolkit and uses these information to map a line number in the canonical machining commands file to a working step. The machining feedback data from LinuxCNC contains the current line number being executed. Using the working step and line number mapping from the canonical machining commands file and the line number information from the machining feedback data, the python script module maps the machining feedback data and current sensor value to a working step. This mapping is useful for working step specific optimizations. The data collected by the python script module at specific time intervals are stored in a database on a server. Thus the current machining feedback data and sensor values are stored in the database in real-time. At the end of the machining process, the python script module generates a new summary record or updates the existing summary record for the product with the information from the current machining process.

Optimization Module

Just-In-Time Compiler for Intelligent Manufacturing prototype has a built-in optimization module that enables users to perform optimization on the input. User has to input the sensor values and the

working step information to the built-in optimization module. For each working step in a product the optimization module is executed once. Optimization module calculates the required change in depth of cut based on the sensor values and the working step information, and updates the guideline file with the reduce_by value for a working step. Optimization module has two optimization models, one for finish milling operation and the other for rough milling operation. Optimization module reads a configuration file that is specific to a product; a configuration file is required for each product manufactured using the prototype. The configuration file contains the below structure(s). The structure is defined one per working step and specifies the details required to calculate the change in depth of cut based on the optimization model.

```
WORKINGSTEP_CONFIG
{
  id = "FINISH␣PLANAR␣FACE1"
  type = "FINISH"
  intercept_coeff=23.6407
  speed_coeff=−0.0142
  feed_coeff=3.5537
  speed_avg=2125
  feed_avg=13
  speed_feed_coeff=−0.0021
  force_coeff=0.3214
  Ra=42
}

WORKINGSTEP_CONFIG
{
  id = "ROUGH␣POCKET1"
  type = "ROUGH"
  Chatter_Threshold = 100
}
```

The optimization model for the finish milling operation uses a multiple linear regression model [ZC07] to predict the surface roughness of the feature based on average resultant peak force, cutting speed, and feed-rate. The multiple linear regression coefficients are specific to a working step and are specified in the configuration file for a product. The structure for a finish milling operation contains the working step id, working step type, multiple linear regression coefficients, constants,

and expected surface roughness (Ra). The type field in the structure for a finish milling operation is assigned a string "FINISH". The predicted surface roughness for the finish milling operation is compared with expected surface roughness specified in the configuration file. If the predicted surface roughness is not the same as the expected surface roughness, the required percentage of change in the average resultant peak force to match the expected surface roughness is calculated and is used to calculate the reduce_by value for the working step. The calculated reduce_by value for the working step is updated in the guideline file. The optimization model for the rough milling operation calculates the change in depth of cut based on the average sensor value of the vibration sensor. The structure for a rough milling operation contains the working step id, working step type, and chatter threshold. The type field in the structure for a rough milling operation is assigned a string "ROUGH". Chatter threshold field specifies the threshold for the vibration sensor value. If the average vibration sensor value for a working step is greater than the chatter threshold, the reduce_by value for the working step is calculated based on the required percentage of decrease in the average vibration sensor value. The calculated reduce_by value for the working step is updated in the guideline file.

The configuration file is read by a parser and a scanner customized to read the structure. The scanner generates an intermediate representation that contains the contents of the configuration file. The IR is passed to a LL(1) grammar that is designed for the structure. The grammar loads the details in the configuration structure in to an internal data structure. The user is required to input the working step id and the machining feedback data to calculate the required change in depth of cut. If the working step is a finish milling operation, user has to input average resultant peak force, cutting speed, and feed-rate. If the working step is a rough milling operation, user has to input average vibration sensor value. Based on the input provided by the user and the configuration specified for the working step, the optimization module calculates the required change in depth of cut and updates the guideline file.

Other Optimizations

The prototype supports only depth of cut optimization but the framework is capable of supporting other optimizations. Following are some sample optimizations that can be performed using the framework: coolant usage optimization, change of tool if the current tool is predicted to break or the required tool is not available on the shop-floor, and feed-rate/spindle speed override. Coolant plays a major role in machining because it helps to clear chips, lubricate materials that are sticky, and carry heat away from the cut. [CNC16] Chips are formed when the workpiece is cut by a tool. Spraying a liquid at the cut helps move the chips out of the way of the tool so it doesn't have to re-cut

the chips or use valuable chip clearance on chips that are not part of the current cut. Re-cutting chips destroys surface finish and dulls tools much more quickly. In the worst case, a tool down in a slot or hole can get clogged with chips and get much hotter or even break. Lubricant is required to make the surface of a sticky workpiece slippery so the chips are less likely to adhere. Cooling liquids such as water-soluble coolants, are capable of carrying heat away from the cut much more efficiently than air. Based on the machine condition coolant can be switched on/off by inserting commands at appropriate locations in the generated canonical machining command program.

Change of tool optimization can be performed in two scenarios. First scenario is when a tool is not available on the shop-floor, the framework can use a different tool to achieve the same operation by generating a new tool-path using a tool-path generation module similar to the one implemented in the research work SPAIM [Rau12]. This avoids the need to change the tool-path manually outside the framework prior to machining. And also avoids the production delay caused by the unavailability of the tool. The framework should be capable of validating the geometry of the product without human intervention. This can be achieved by developing a module to compare the geometry of the new tool-path and the original tool-path. This validation is required to check for any violation of the product geometry. Next scenario is when an optimization algorithm predicts tool breakage after few runs of the product, the module to change the tool and tool-path can be used to change the current tool with a new tool.

The prototype can be extended to perform optimizations on feed-rate and spindle speed in real-time because LinuxCNC is capable of overriding feed-rate and spindle speed in real-time. So, third-party optimization modules can use these variables to control the cutting parameters of the tool.

Application Programming Interface

JIT Compiler for Intelligent Manufacturing prototype exposes APIs that enable users to read feedback data and to access the guideline file. These APIs can be used by third-party applications to remotely monitor the machining process, explore historical feedback data set, or perform optimization using the feedback data and update the guideline file. The APIs are written in C++ and compiled as a library. This library can be exported to any remote machine along with the .cshrc file that contains the path to the guideline file and the connection details to the database server. Third-party applications that monitor the feedback data or access the historical feedback data set can use this library from any remote machine. But third-party applications that access the guideline file to perform optimization have to reside in the local machine because the guideline file cannot be accessed over the network. A sample third-party application is implemented to access the exposed APIs and fetch the feedback

data. And also the third-party application updates the guideline file with optimized reduce_by value for working steps in a product. The APIs exposed by the prototype are provided in the appendix for reference.

CHAPTER

5

EVALUATION

Just-In-Time Compiler for Intelligent Manufacturing prototype was tested in an Intel Core i7-4510U platform. Experiments were performed to simulate tool-path using sample STEP-NC files, simulate optimized tool-path that is generated using the feedback from the previous run, experiment the functionalities of the APIs by executing a sample third-party application, and evaluate the impact of optimization on the production process. The original and optimized tool-paths were simulated in LinuxCNC and compared to demonstrate the optimization performed by the prototype using the optimization module and a third-party application.

Simulation

Two sample STEP-NC files are used to simulate the tool-path in LinuxCNC. One of the sample files is used to evaluate the optimization performed using the built-in optimization module and the other is used to evaluate the optimization performed using a third-party application. The sample STEP-NC files used in the evaluation of the prototype are ex1_comment.stp and face1.stp and are provided in the appendix.

face1.stp file is used in the first experimentation. face1.stp file, a tools file and a guideline file are loaded in the ISO14649 toolkit by the user. The user interface of the ISO14649 toolkit [NIS09]

loaded with these files is shown in the Figure 5.1. The tools file "iso14649.tool_ex1" is specific to a CNC and is the same for all the products manufactured in a CNC until a tool is removed or added to the CNC. The input file is the face1.stp STEP-NC file, the oracle file is the guideline file for the product face1 and is empty for the first run of the product, and the output file contains the generated canonical machining commands. The option "continue interpreting after error" allows the ISO14649 toolkit to generate canonical machining commands even when an error occurs during generation of canonical machining commands. The option "print stack on error" enables the ISO14649 toolkit to print the execution stack at the time of error. ISO14649 toolkit generates canonical machining commands file after the user selects the option "start interpreting". At the end of the execution of ISO14649 toolkit, the output file contains the generated canonical machining commands for the product face1. The generated canonical machining commands file is provided in the appendix for reference.

```
CURRENT SETTINGS
tool file name: ../data/iso14649.tool_ex1
input file name: ../data/data/face1.stp
output file name: ../output/face1.can
oracle file name: ../data/oracle_files/face1_oracle.txt
continue interpreting after error: NO
print stack on error: YES
enter a number:
1 = start interpreting
2 = change tool file
3 = change input file
4 = change output file
5 = change oracle file
6 = continue on error
7 = do not print stack on error
8 = quit
enter choice => 1
```

Figure 5.1 User interface of ISO14649 toolkit loaded with face1.

The generated canonical machining commands file is loaded in LinuxCNC [lin16b] by the user. Figure 5.2 shows the user interface of LinuxCNC loaded with the canonical machining commands file generated for the product face1. The preview window on the right in the figure shows the tool-path to be simulated using the canonical machining command program. The manual control window allows the user to change xyz axis settings of the tool before the machining process. Once the user starts the machining process by clicking the "begin executing current file" button, the tool in the simulation window will trace its path along the tool-path specified by the canonical machining commands. Figure 5.3 shows the final tool-path simulated for the product face1. This is the original

tool-path generated for the product without any optimization on the input.

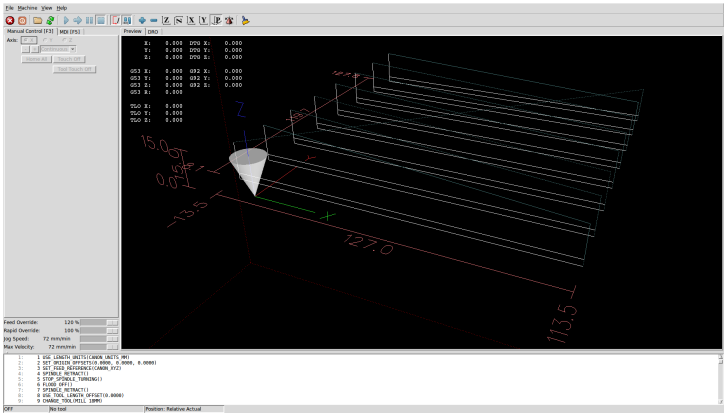


Figure 5.2 LinuxCNC loaded with the product face1.

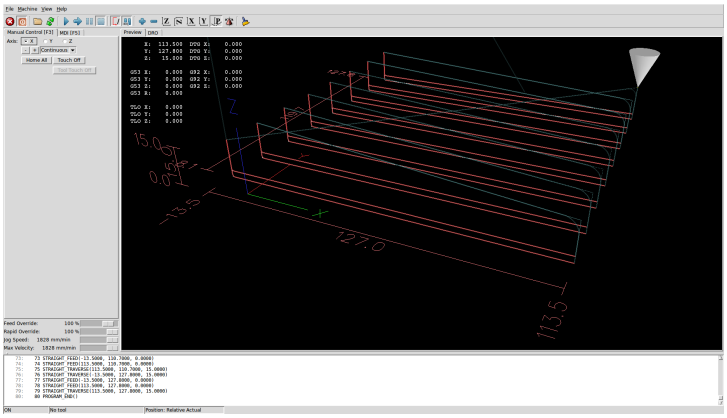


Figure 5.3 Simulation of face1 product in LinuxCNC.

The second experimentation is carried out with ex1_comment product. The STEP-NC file and guideline file of ex1_comment product are loaded in ISO14649 toolkit by the user. Figure 5.4 shows user interface of the ISO14649 toolkit [NIS09] loaded with the STEP-NC file and guideline file of the product ex1_comment. The guideline file of ex1_comment product is empty for the first run. The tools file is the same file used for face1 product because tools file is specific to a CNC. After the user

selects the option to "start interpreting", the ISO14649 toolkit reads the STEP-NC file, guideline file, and tools file to generate canonical machining commands. The canonical machining commands file for ex1_comment product is generated at the end of execution of ISO14649 toolkit.

```
CURRENT SETTINGS
tool file name: ../data/iso14649.tool_ex1
input file name: ../data/ex1_comment.stp
output file name: ../output/ex1_comment.can
oracle file name: ../data/oracle_files/ex1_comment_oracle.txt
continue interpreting after error: NO
print stack on error: YES
enter a number:
1 = start interpreting
2 = change tool file
3 = change input file
4 = change output file
5 = change oracle file
6 = continue on error
7 = do not print stack on error
8 = quit
enter choice => 1
```

Figure 5.4 User interface of ISO14649 toolkit loaded with ex1_comment.

The canonical machining commands generated for ex1_comment product is loaded in LinuxCNC [lin16b] by the user and the simulation starts when the user clicks the "begin executing current file" button. LinuxCNC simulation output for ex1_comment product is shown in the figure 5.5.

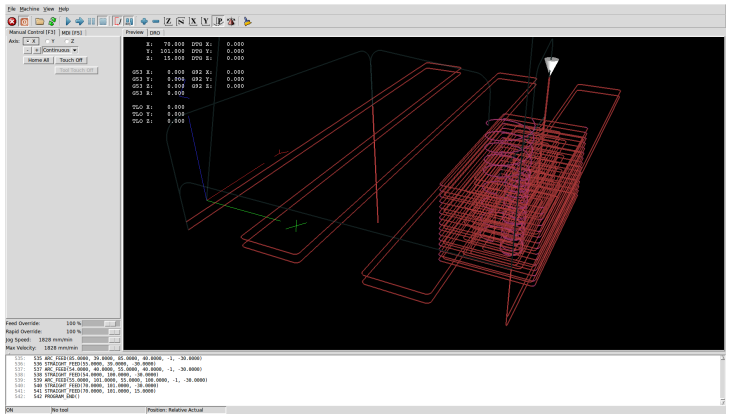


Figure 5.5 Simulation of ex1_comment product in LinuxCNC.

Optimization module

Just-In-Time Compiler for Intelligent Manufacturing framework supports a built-in optimization module that accepts the sensor values and updates the guideline file with the optimization information by executing an optimization algorithm on the sensor values. The built-in optimization module has two optimization models; one optimization model is for finish milling [ZC07] and the other optimization model is for rough milling. These two optimization models are experimented using the values from a random number generation function. They are used to demonstrate the functionality of the optimization module.

```
CURRENT SETTINGS
Config file name: ../data/config.txt
Workingstep ID: FINISH PLANAR FACE1
Speed: 900
Feed: 4
Average resultant peak force: 40
Chatter Average: 100
Enter a number:
1 = calculate change in depth of cut
2 = change config file
3 = change workingstep id
4 = change speed
5 = change feed
6 = change average resultant peak force
7 = change chatter average
8 = quit
Enter choice => 1
```

Figure 5.6 User interface of optimization module loaded with feedback data of FINISH PLANAR FACE1 working step.

Figure 5.6, 5.7, and 5.8 shows the user interface for the optimization module loaded with the sensor values for the working steps FINISH PLANAR FACE1, FINISH POCKET1, and ROUGH POCKET1 respectively. Config file name field is the path to the configuration file for the product ex1_comment. Working step id, speed, feed, average resultant peak force, and chatter average fields are the working step id for which the optimization module is executed, cutting speed, feed rate, average resultant peak force value, and the average vibration sensor value for the working step respectively. FINISH PLANAR FACE1 and FINISH POCKET1 working steps are finish milling working steps. These working steps are optimized using the finish milling optimization model. This model uses the cutting speed,

```
CURRENT SETTINGS
Config file name: ../data/config.txt
Workingstep ID: FINISH POCKET1
Speed: 1750
Feed: 7
Average resultant peak force: 30
Chatter Average: 100
Enter a number:
1 = calculate change in depth of cut
2 = change config file
3 = change workingstep id
4 = change speed
5 = change feed
6 = change average resultant peak force
7 = change chatter average
8 = quit
Enter choice => 1
```

Figure 5.7 User interface of optimization module loaded with feedback data of FINISH POCKET1 working step.

```
CURRENT SETTINGS
Config file name: ../data/config.txt
Workingstep ID: ROUGH POCKET1
Speed: 1750
Feed: 7
Average resultant peak force: 89
Chatter Average: 150
Enter a number:
1 = calculate change in depth of cut
2 = change config file
3 = change workingstep id
4 = change speed
5 = change feed
6 = change average resultant peak force
7 = change chatter average
8 = quit
Enter choice => 1
```

Figure 5.8 User interface of optimization module loaded with feedback data of ROUGH POCKET1 working step.

feed rate, and average resultant peak force to calculate the change in depth of cut. User inputs these values along with the working step id and the path to the configuration file. The optimization module calculates the change in depth of cut for the working step and updates the guideline file

with the optimization information for the working step. ROUGH POCKET1 working step is a rough milling working step and is optimized using the rough milling optimization model. This model uses the average vibration sensor value for the working step to calculate the required change in depth of cut for the working step. The updated guideline file is shown in the figure 5.9. The value of the reduce_by field for the working step FINISH PLANAR FACE1 is 2.078747. This reduces the depth of cut for the working step by half. So, the number of passes for the working step increases by 100%. The number of passes for the working step ROUGH POCKET1 increases by 80% because the value of the reduce_by field is 1.875001. But for the working step FINISH POCKET1, the number of passes is decreased by 53.33% because the reduce_by field has a value of 0.418156, which increases the depth of cut by 114.29%.

```
AXIAL_CUTTER_DEPTH_REDUCTION
{
        id = "ROUGH POCKET1"
        reduce_by = 1.875001
}
AXIAL_CUTTER_DEPTH_REDUCTION
{
        id = "FINISH POCKET1"
        reduce_by = 0.418156
}
AXIAL_CUTTER_DEPTH_REDUCTION
{
        id = "FINISH PLANAR FACE1"
        reduce_by = 2.078747
}
```

Figure 5.9 Updated guideline file with the optimization details for ex1_comment product.

The updated guideline file and the STEP-NC file for the product ex1_comment is used to generate the canonical machining commands for the next run of the product. The ISO14649 toolkit [NIS09] generates the optimized canonical machining commands. The number of canonical machining commands in the unoptimized canonical machining command program and optimized canonical machining command program is shown in the table 5.1. The decrease in the number of passes in the working step FINISH POCKET1 has significant impact on the decrease in the number of canonical machining commands than the increase in the number of passes in the working steps

Table 5.1 Number of canonical machining commands in the unoptimized and optimized canonical machining command program of the product ex1_comment.

Canonical machining command program	Number of lines
Unoptimized	542
Optimized	504

FINISH PLANAR FACE1 and ROUGH POCKET1. This is because the number of commands required to change the tool direction in the working step FINISH POCKET1 is higher than that of other two working steps. The generated optimized canonical machining commands file is loaded and simulated in LinuxCNC [lin16b] by the user. Figure 5.10 shows the simulated optimized tool-path for the product ex1_comment.

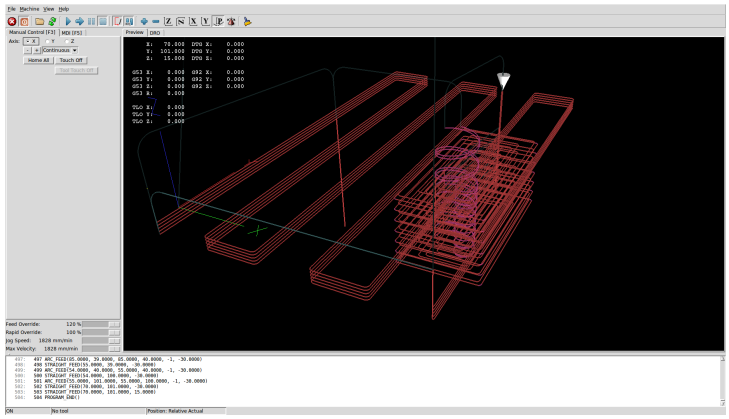


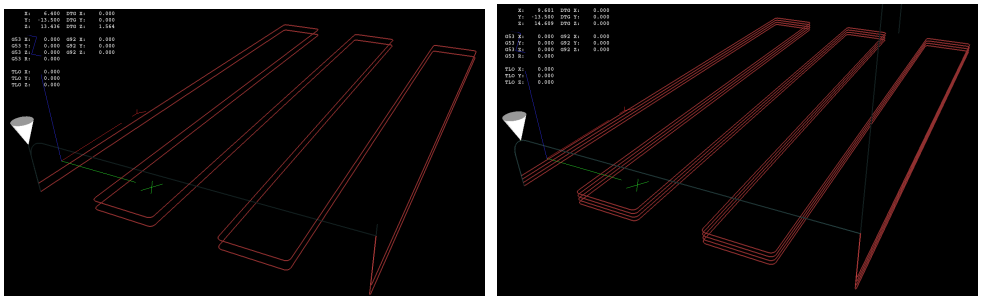
Figure 5.10 Simulation of optimized tool-path for ex1_comment product.

Figure 5.11, 5.12, and 5.13 shows the tool-paths generated for the unoptimized and optimized canonical machining command programs for the working steps FINISH PLANAR FACE1, FINISH POCKET1, and ROUGH POCKET1. It is clear from these figures that the number of passes for the working steps FINISH PLANAR FACE1 and ROUGH POCKET1 have increased, and the number of passes for the working step FINISH POCKET1 has decreased.

The percentage of increase in the number of passes for the working steps FINISH PLANAR FACE1 and ROUGH POCKET1 are 100% and 80% respectively and the percentage of decrease in the number

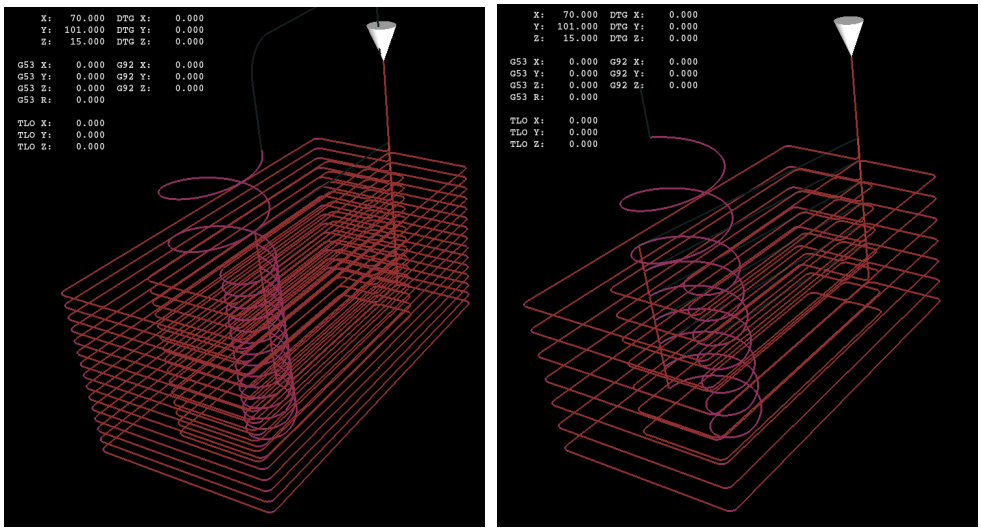
of passes for working step FINISH POCKET1 is 53.33%. The percentage change in the number of passes for these working steps have significant impact on the machining time of these working steps. The time taken for machining the working steps FINISH PLANAR FACE1 and ROUGH POCKET1 have increased by 126.24% and 67.433% respectively and the time taken for machining the working step FINISH POCKET1 has decreased by 49.333%. These details are shown in the tables 5.4 and 5.6.

Table 5.2 shows the time taken for generation of canonical machining commands and total machining time for unoptimized and optimized versions of the product ex1_comment. The time taken for generation of canonical machining commands and total machining time has increased by 4.894% and 16.316% respectively. The time taken for generation of canonical machining commands is in the magnitude of microseconds and the overhead in the generation time is 4.894%. This does not have significant impact on the production time. The increase in the total machining time is attributed to the increase in the number of passes for the working steps FINISH PLANAR FACE1 and ROUGH POCKET1. The increase in the time taken for machining these working steps has less impact on the increase in the total machining time because these working steps contribute only 20.34% and 24.06% respectively towards the total unoptimized machining time of the product ex1_-comment. But the working step FINISH POCKET1 has a contribution of 52.35%. So, the decrease in the machining time caused by this working step has compensated certain percentage of increase in the production time by the other two working steps. Thus the overhead caused by the framework is very less when compared to the time taken to machine the working steps. And also the increase in the machining time of the working steps is proportional to the increase in the number of canonical machining commands for the working steps. The machining time specified in the evaluation are not actual time taken to machine the part. But they are the time taken to simulate the part in LinuxCNC.



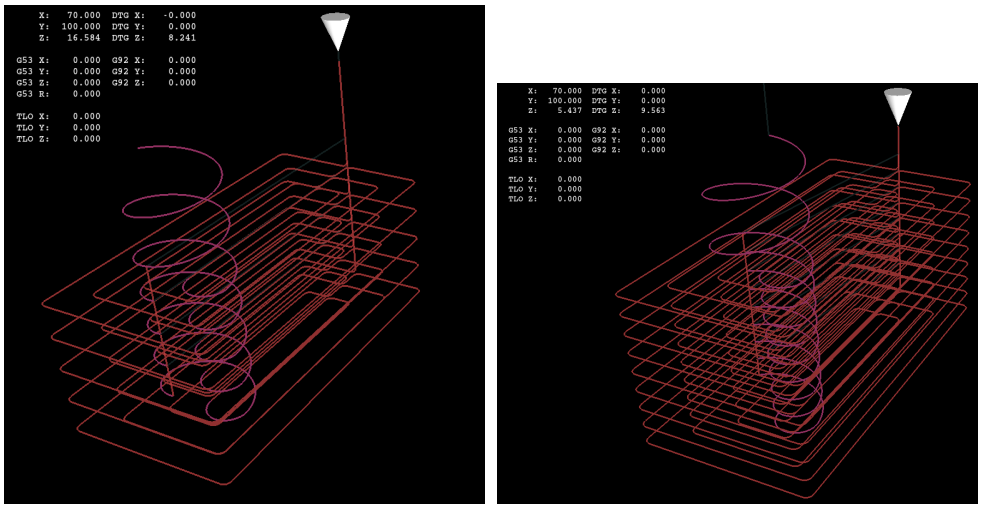
(a) **(b)**

Figure 5.11 Unoptimized (a) and Optimized (b) tool-path for the working step FINISH PLANAR FACE1



(a) **(b)**

Figure 5.12 Unoptimized (a) and Optimized (b) tool-path for the working step FINISH POCKET1



(a) **(b)**

Figure 5.13 Unoptimized (a) and Optimized (b) tool-path for the working step ROUGH POCKET1

Table 5.2 Time taken for generating canonical machining commands and machining the product ex1_-comment.

Version	Canonical Machining Commands generation time	Machining Time
Unoptimized	2350µs	403.015s
Optimized	2465µs	468.771s

Table 5.3 Time taken for each working step during unoptimized and optimized machining of the product ex1_comment.

Version	FINISH PLANAR FACE1	ROUGH POCKET1	FINISH POCKET1
Unoptimized	82.483s	97.84s	211.928s
Optimized	186.126s	163.817s	107.378s

Table 5.4 Percentage change in the number of passes for each working step in the product ex1_comment.

Product Name	FINISH PLANAR FACE1	ROUGH POCKET1	FINISH POCKET1
ex1_comment	100% Increase	80% Increase	53.33% Decrease

Table 5.5 Percentage change in time taken for optimized Canonical Machining Commands generation, optimized Machining Time of the product ex1_comment.

Product Name	CMC generation time	Machining Time
ex1_comment	4.894% Increase	16.316% Increase

Table 5.6 Percentage change in time taken for each working step in the product ex1_comment during optimized machining.

Product Name	FINISH PLANAR FACE1	ROUGH POCKET1	FINISH POCKET1
ex1_comment	126.24% Increase	67.433% Increase	49.333% Decrease

Third-Party application

Application programming interfaces exposed by the Just-In-Time Compiler for Intelligent Manufacturing prototype can be used to write third-party applications to monitor the machining process, explore the historical feedback data set, and perform optimization on the input to the framework. A sample third-party application is implemented to experiment the functionalities of the APIs. This application uses the APIs to fetch all the products that has been simulated by the LinuxCNC, fetch all the working steps of the product ex1_comment, fetch all the sensors used in manufacturing the product ex1_comment, get the manufacturing summary of the product ex1_comment, fetch the sensor 1 values for the working step FINISH PLANAR FACE1 during the first run of the product ex1_comment, fetch the sensor 2 values during the first run of the product ex1_comment, and access the guideline file to update the value of the reduce_by field for the working step FINISH PLANAR FACE1 of the product face1. Figure 5.14 shows the output of the third party application and the figure 5.15 shows the updated guideline file. The value of the reduce_by field for the working step FINISH PLANAR FACE1 is 2.000000. So, the depth of cut for the working step is reduced by half and the number of passes for the working step is doubled.

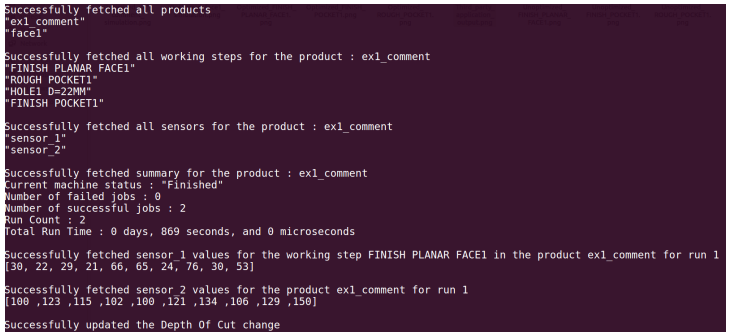


Figure 5.14 Output of the third party application.

The updated guideline file is used by the ISO14649 toolkit [NIS09] in the next run of the product face1. Table 5.7 shows the number of canonical machining commands in the unoptimized and optimized canonical machining command programs. The number of canonical machining commands in the optimized canonical machining command program is greater than that of unoptimized canonical machining command program because the number of passes in the product face1 has doubled. The updated guideline file is loaded in LinuxCNC to simulate the optimized tool-path.

```
AXIAL_CUTTER_DEPTH_REDUCTION
{
        id = "FINISH PLANAR FACE1"
        reduce_by = 2.000000
}
```

Figure 5.15 Updated guideline file of the product face1.

Table 5.7 Number of canonical machining commands in the unoptimized and optimized canonical machining commands file of the product face1.

Canonical machining commands file	Number of lines
Unoptimized file	80
Optimized file	143

Figure 5.16 shows the optimized tool-path simulated in LinuxCNC [lin16b]. Figure 5.3 and 5.16 show that the number of passes in the optimized tool-path has increased because of the optimization performed by the ISO14649 toolkit using the guideline file. The percentage of increase in the number of passes for the working step FINISH PLANAR FACE1 in the optimized tool-path is 100% and it has a significant impact on the machining time of the working step. The machining time of the working step has increased by 96.36%. The increase in the machining time of the working step has increased the total machining time of the product by 96.49%. Thus the increase in the number of passes for the working step FINISH PLANAR FACE1 has significant impact on the total machining time of the product face1. The overhead caused by the generation of optimized canonical machining commands for the product face1 is 5.49%. The canonical machining commands generation time is in the magnitude of microseconds and the percentage change in the canonical machining commands generation time does not have significant impact on the production time of the product face1. Thus the overhead caused by the framework is very less when compared to the time taken to machine the working step. And also the increase in the machining time of the working step is proportional to the increase in the number of canonical machining commands for the working step. The machining time specified in the evaluation are not actual time taken to machine the part. But they are the time taken to simulate the part in LinuxCNC.

Table 5.8 Time taken for generating canonical machining commands and machining the product face1.

Version	Canonical Machining Commands generation time	Machining Time
Unoptimized	1419µs	156.280s
Optimized	1497µs	307.0824s

Table 5.9 Time taken for the working step FINISH PLANAR FACE1 during unoptimized and optimized machining of the product face1.

Version	FINISH PLANAR FACE1
Unoptimized	156.277s
Optimized	306.877s

Table 5.10 Percentage change in the number of passes for the working step FINISH PLANAR FACE1 in the product face1.

Product Name	FINISH PLANAR FACE1
face1	100% Increase

Table 5.11 Percentage change in time taken for optimized Canonical Machining Commands generation, optimized Machining Time of the product face1.

Product Name	CMC generation time	Machining Time
face1	5.49% Increase	96.49% Increase

Table 5.12 Percentage change in time taken for the working step FINISH PLANAR FACE1 in the product face1 during optimized machining.

Product Name	FINISH PLANAR FACE1
face1	96.36% Increase

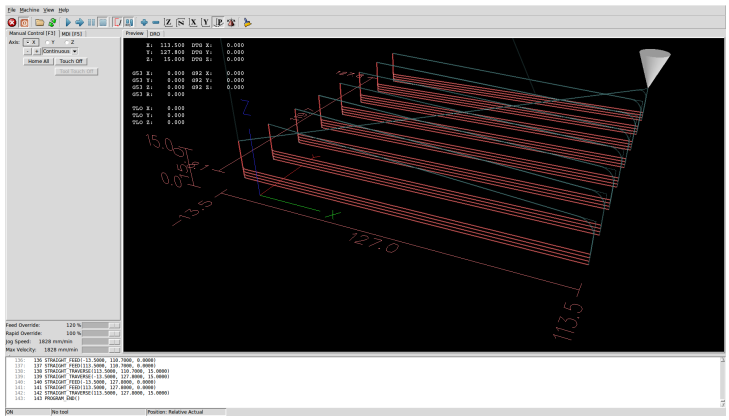


Figure 5.16 Simulation of optimized tool-path for face1 product.

Benefits of the framework

JIT Compiler for Intelligent Manufacturing framework can be used on the shop floor to monitor the machining process remotely, explore historical feedback data set and build customized optimization algorithms to optimize the machining process on the fly. This section presents three scenarios in which the JIT Compiler for Intelligent Manufacturing framework can be used on the shop-floor.

Scenario A

If the CNCs or the cutting tools used in the CNCs on the shop-floor are newer than expected and the depth of cut is optimized to an old tool, the framework can detect this using the surface roughness based optimization model. The surface roughness of the product is predicted to be very less than the desired surface roughness and the model suggests an increase in the depth of cut such that the surface roughness is slightly less than the desired surface roughness. The increase in depth of cut decreases the number of passes required to complete a working step. As shown in the evaluation of the optimization module, the decrease in the number of passes has significant impact on the production time of the product. If there are several working steps in a product and all of them use the same tool, the optimization model decreases the number of passes for all these working steps.

If a product has m working steps and the time taken to machine each working step is y minutes, the total time taken to machine the product is represented as

$$x = y \times m \tag{5.1}$$

minutes. If n out of m working steps use a relatively new tool and the framework reduces the number

of passes for the n working steps by z%, the total time taken to machine the product using optimized depth of cut is

$$x' = y \times n \times (1 - \frac{z}{100}) + y \times (m - n) \tag{5.2}$$

minutes and the percentage of decrease in the machining time is

$$\%of decrease = \frac{(y \times m) - (y \times n \times (1 - \frac{z}{100}) + y \times (m - n))}{y \times m} \times 100 \tag{5.3}$$

If there are 10 working steps in a product and each working step takes 15 minutes to complete, the total time taken to machine the product is 2 hours and 30 minutes. If 7 out of 10 working steps use a relatively new tool and the percentage of decrease in the number of passes for these working step is 60%, the time taken to machine the optimized tool-path is 1 hour and 27 minutes. The production time is decreased by 42%. If there are 100 machines with a relatively new tool and all these machines manufacture the same product, the production time saved is 105 hours, which allows the shop floor to manufacture 72 more pieces of the same product. Users of the framework can build optimization algorithms customized to the tool condition, tool-path and workpiece material to efficiently detect working steps whose depth of cut can be increased. This will lead to a much more efficient production of the product.

Scenario B

The optimization module in the framework detects tool wear by monitoring the chatter level and surface roughness. If the tool used in a CNC is old and is wearing off, the finish of the surface is defective. The optimization module detects this and suggests a decrease in the depth of cut for the working steps those use the old tool. The decrease in depth of cut decreases the force exerted on the tool. So, the number of defective pieces manufactured is decreased. But this increases the number of passes for the working steps.

If the defective pieces are manually identified on the shop floor after n cycles of manufacturing a product and there are m machines with an old tool manufacturing the same product,

$$n \times m \tag{5.4}$$

defective pieces of the product are manufactured. If each machine takes x minutes to manufacture the product and requires y minutes to change the cutting parameters manually, there is a production delay of

$$(x \times n \times m) + (m \times y) \tag{5.5}$$

minutes. But if the framework is used to detect the tool wear by monitoring the surface roughness and chatter levels, the depth of cut is decreased in the next cycle. So, the force exerted on the material and tool is reduced and there is no damage to the finish of the part. The number of defective pieces manufactured, if the framework is used is

$$1 \times m \tag{5.6}$$

pieces. The production delay caused is

$$(x \times m) \tag{5.7}$$

minutes. The percentage of decrease in the number of defective parts manufactured is

$$\%of decrease = \frac{(n \times m) - (1 \times m)}{n \times m} \times 100 = \frac{n-1}{n} \times 100 \tag{5.8}$$

and the percentage of decrease in the production delay is

$$\%of decrease = \frac{(x \times n \times m) + (m \times y) - (x \times m)}{(x \times n \times m) + (m \times y)} \times 100 \tag{5.9}$$

If 10 machines manufacture the same product and the defective pieces are detected after 3 cycles of manufacturing the product, 30 defective pieces are manufactured. If the time taken to manufacture a piece of the product is 1 hour and the average time taken to modify the cutting parameters in a machine is 5 minutes, there is a production delay of 30 hours and 50 minutes. If the framework is used to detect the tool condition and modify the depth of cut, the production delay is reduced to 10 hours and the number of defective pieces manufactured is reduced to 10. The percentage of decrease in the production delay and number of defective pieces manufactured are 67.5% and 66.7% respectively. Users of the framework can build an optimization algorithm that predicts the tool wear and changes the cutting parameters even before the production of first cycle of defective products. This would avoid production of defective products and the production delay caused due to manufacturing defective products.

Scenario C

The material of the workpiece plays a major role in deciding the correct cutting parameters. The cutting parameters are feed-rate, spindle speed, and depth of cut. If the material of the workpiece were to change during a manufacturing process, the optimized cutting parameters have to be fed in to the machine by the operator. Operator can control only the feed-rate and spindle speed because a change in depth of cut causes the tool-path to change and it is not possible to edit the tool-path during machining process. And also there is a delay in the production process caused due to human

intervention because the operator has to identify the optimal cutting parameters either by trial and error mechanism or using prior knowledge about the workpiece material. If the hardness of the workpiece material is stored in the database of the framework and historical data sets for the cutting parameters and hardness combinations are available, an algorithm can be devised to identify the optimal cutting parameters based on the hardness of the workpiece material. These parameters can be used to generate canonical machining commands for optimized machining of the product. This does not require the operator to have prior knowledge about the workpiece material and does not require human intervention during manufacturing the product. Thus the production delay caused due to human intervention is avoided.

CHAPTER

6

CONCLUSION AND FUTURE WORK

Evaluation of the JIT Compiler for Intelligent Manufacturing prototype shows that the overhead of the framework is less and has opened the possibility of using the framework on the shop-floor for efficient production. The hypothesis stands valid as the prototype accepts STEP-NC ISO 14649 data model as input, uses an open-source controller as the CNC controller, stores the feedback data in real-time in a database, performs optimizations on the input, exposes APIs to read feedback data and to suggest optimization information to the controller, and enables users to remotely monitor the machining process, build third-party optimization algorithms and explore historical dataset. The STEP-NC ISO 14649 data model contains high-level machining information that allows optimization of the machining process and can be used across CNCs manufactured by different CNC vendors. The framework is flexible enough to replace the input with the latest STEP-NC ISO data model and to control CNCs that are compatible with LinuxCNC. And also the framework allows users to efficiently machine a product using customized third-party optimization algorithms customized for the product.

JIT Compiler for Intelligent Manufacturing prototype performs optimization on the input between runs; optimization is performed based on the feedback from the previous runs. The next step is to modify the prototype to enable real-time optimization of the machining process. For instance, if there are two similar working steps, the prototype can monitor the feedback data from

the first working step and optimize the machining process for the second working step based on the optimization performed using the feedback data from the first working step. The feedback data from the machining process can be used to build a real-time visualization tool that displays the real-time tool-path, tool position, and operation performed in a CNC. The prototype supports only optimization based on depth of cut. It can be extended to support different optimizations such as coolant usage based on the machining condition, change of tool based on the tool condition and tool availability, change in feed-rate, and change in spindle speed. Currently, geometry of the product after generation of optimized tool-path is not validated against the geometry of the product with the original tool-path. This validation is required to check for any violation of the product geometry. A module has to be implemented to do this validation because changes to the tool-path should not affect the geometry of the product. The ISO14649 toolkit in the prototype should be replaced with a compiler that accepts the latest STEP-NC data model or AP238 data model and converts it into an intermediate representation that can be used to drive LinuxCNC. The new compiler can be easily modified to perform optimizations on a STEP-NC file because STEP-NC data models have an object oriented representation of the machining process and the geometry of the product. These modifications will enable the framework to accept STEP-NC data model generated by any CAD/CAM software. And also the framework can be used on the shop-floor and the users can utilize the full potential of the framework.

BIBLIOGRAPHY

[Arn02] Arnold, M. et al. "Online Feedback-Directed Optimization of Java". *OOPSLA '02 Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* **37**.11 (2002), pp. 111–129.

[CNC16] CNCCookbook, I. *Coolant and Chip Clearing*. 2016. URL: `http://www.cnccookbook.com/CCCNCMillFeedsSpeedsCoolant.htm` (visited on 06/03/2016).

[Har06] Hardwick, M. *Manufacturing Integration using the STEP-NC DLL*. 2006. URL: `http://www.steptools.com/support/stepnc_docs/stepncdll/Library_white.pdf` (visited on 05/24/2016).

[Inc13] Inc.com. *Computer-Aided Design (CAD) and Computer-Aided Manufacturing (CAM)*. 2013. URL: `http://www.inc.com/encyclopedia/computer-aided-design-cad-and-computer-aided-cam.html` (visited on 05/24/2016).

[KK09] Kržič and. P Stoic, A. & Kopač, J. "STEP-NC: A New Programming Code for the CNC Machines". *Strojniški vestnik - Journal of Mechanical Engineering* **55**.6 (2009), pp. 406–417.

[Lal08] Lalitnarayan, K. *Computer Aided Design and Manufacturing*. New Delhi: Prentice Hall of India, 2008.

[lin16a] linuxcnc.org. *LinuxCNC Developer Documentation*. 2016. URL: `http://www.linuxcnc.org/docs/2.5/html/code/Code_Notes.html` (visited on 05/24/2016).

[lin16b] linuxcnc.org. *LinuxCNC SourceCode*. `git://git.linuxcnc.org/git/linuxcnc.git`. 2016.

[Moh09] Mohsen, S. *Reading Materials for IC Training Modules, Computer Numerical Control*. INDUSTRIAL CENTRE, THE HONG KONG POLYTECHNIC UNIVERSITY, 2009.

[NIS09] NIST. *A toolkit for ISO 14649 data model*. `https://github.com/ArcEye/iso-14649-toolkit`. 2009.

[Pro97] Proctor, F. et al. *Canonical Machining Commands*. 1997. URL: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.9814&rep=rep1&type=pdf` (visited on 05/24/2016).

[Rau12] Rauch, M. et al. "An advanced STEP-NC controller for intelligent machining processes". *Robotics and Computer-Integrated Manufacturing* **28** (2012), pp. 375–384.

[RX13] Ridwan, F. & Xu, X. "Advanced CNC system with in-process feed-rate optimisation". *Robotics and Computer-Integrated Manufacturing* **29** (2013), pp. 12–20.

[Vij08] Vijayaraghavan, A. et al. "Improving machine tool interoperability using standardized interface protocols: MTConnect". *Proceedings of ISFA, International Symposium on Flexible Automation* (2008).

[wik16a] wikipedia.org. *Computer-aided manufacturing*. 2016. URL: `https://en.wikipedia.org/wiki/Computer-aided_manufacturing` (visited on 05/24/2016).

[wik16b] wikipedia.org. *LinuxCNC*. 2016. URL: `https://en.wikipedia.org/wiki/LinuxCNC` (visited on 05/24/2016).

[wik16c] wikipedia.org. *Numerical control*. 2016. URL: `https://en.wikipedia.org/wiki/Numerical_control` (visited on 05/24/2016).

[ZC07] Zhang, J. Z. & Chen, J. C. "The development of an in-process surface roughness adaptive control system in end milling operations". *The International Journal of Advanced Manufacturing Technology* **31** (2007), pp. 877–887.

APPENDICES

APPENDIX

A

APPLICATION PROGRAMMING INTERFACES

Below is the list of Application Programming Interfaces exposed by the Just-In-Time Compiler for Intelligent Manufacturing framework.

APIs	Description
reduceDOC(productName, workingStepID, reduceBy)	Changes the reduce_by value in the guideline file for a working step in a product. This API accepts the product name, working step id, and reduce_by value that has to be updated in the guideline file, validates the product name and working step id using the database where the feedback data from LinuxCNC is stored, finds the guideline file using the product name, updates the guideline file with the reduce_by value for the working step, and returns whether or not the update was successful.
getProducts()	Returns all the products manufactured in the CNC machine.
getWSIDs(productName)	Accepts a product name, validates the product using the database, and returns all the working steps in the product.

getSensors(productName)	Accepts a product name, validates the product using the database, and returns all the sensors used in machining the product.
getSensorValues(productName, run, workingStepID, sensorID)	Accepts a productName, run number, working step id and sensor id, validates these details using the database, and returns the sensor values of the sensor with the given sensor id for the given run number and the given working step in the specified product.
getSensorValues(productName, run, sensorID)	Accepts a productName, run number and sensor id, validates these details using the database, and returns the sensor values of the sensor with the given sensor id for the given run number of the specified product.
getSummary(productName)	Accepts a product name, validates the product using the database, and returns the summary of the product. Summary contains the current machining status for the product, number of failed jobs, number of successful jobs, number of times the product has been manufactured using the CNC, and the total machining time spent on the product. The current machining status can be finished, incomplete, and running. They represent whether the last machining of the product was completed successfully, was aborted, or is still in process.
getRunTime(productName, run)	Accepts a productName and run number, validates these details using the database, and returns the machining time for the run.(*)
getRunTime(productName, run, workingStepID)	Accepts a productName, run number and working step id, validates these details using the database, and returns the machining time for the working step in the run.(*)
setFeedRate(productName, workingStepID, feed-rate)	Accepts a productName, working step id and feed-rate , validates these details using the database, and sets the feed-rate for the working step in the product.(*)
setSpindleSpeed(productName, workingStepID, spindle-speed)	Accepts a productName, working step id and spindle speed, validates these details using the database, and sets the spindle speed for the working step in the product.(*)

getToolLocation(productName, run)	Accepts productName and run number, validates these details using the database, and returns a list of tool locations for the run of the product. This can be used to plot and simulate the tool-path in a 3D plane.(*)
getToolLocation(productName, run, workingStepID)	Accepts productName, run number, and working step id, validates these details using the database, and returns a list of tool locations for working step in the run of the product. This can be used to plot and simulate the tool-path in a 3D plane.(*)
setCoolantStatus(productName, workingStepID, status)	Accepts a productName, working step id and status, validates these details using the database, and sets the coolant to ON/OFF depending the status value for the working step in the product.(*)

Table A.1 APIs exposed by the Just-In-Time Compiler for Intelligent Manufacturing framework.

(*) - APIs that are yet to be implemented.

APPENDIX

B

STEP-NC FILES

ex1_comment product

Below is the STEP-NC file for ex1_comment product.

```
ISO-10303-21;

HEADER;
FILE_DESCRIPTION(('ISO␣14649-11␣EXAMPLE␣1',
            'SIMPLE␣PROGRAM␣WITH␣A␣PLANAR_FACE,␣A␣POCKET,␣AND␣A␣ROUND_HOLE'),
            '1');
FILE_NAME('EXAMPLE1.STP',
            '2002-02-02',
            ('YONG␣TAK␣HYUN','JOCHEN␣WOLF'),
            ('WZL,␣RWTH-AACHEN'),
            '$',
            'ISO␣14649',
            '$');
FILE_SCHEMA(('MACHINING_SCHEMA','MILLING_SCHEMA'));
```

```
ENDSEC;

DATA;
#1= PROJECT('EXECUTE␣EXAMPLE1',#2,(#4),$,$,$);
#2= WORKPLAN('MAIN␣WORKPLAN',(#10,#11,#12,#13,#14),$,#8,$);
#4= WORKPIECE('SIMPLE␣WORKPIECE',#6,0.010,$,$,$,(#66,#67,#68,#69));
#6= MATERIAL('ST-50','STEEL',(#7));
#7= DESCRIPTIVE_PARAMETER('E=200000N/M2','mild');
#8= SETUP('SETUP1',#71,#62,(#9));
#9= WORKPIECE_SETUP(#4,#74,$,$,());
#10= MACHINING_WORKINGSTEP('WS␣FINISH␣PLANAR␣FACE1',#62,#16,#19,$);
#11= MACHINING_WORKINGSTEP('WS␣DRILL␣HOLE1',#62,#17,#20,$);
#12= MACHINING_WORKINGSTEP('WS␣REAM␣HOLE1',#62,#17,#21,$);
#13= MACHINING_WORKINGSTEP('WS␣ROUGH␣POCKET1',#62,#18,#22,$);
#14= MACHINING_WORKINGSTEP('WS␣FINISH␣POCKET1',#62,#18,#23,$);
#16= PLANAR_FACE('PLANAR␣FACE1',#4,(#19),#77,#63,#24,#25,$,());
#17= ROUND_HOLE('HOLE1␣D=22MM',#4,(#20,#21),#81,#64,#58,$,#26);
#18= CLOSED_POCKET('POCKET1',#4,(#22,#23),#84,#65,(),$,#27,$,#37,#28);
#19= PLANE_FINISH_MILLING($,$,'FINISH␣PLANAR␣FACE1',10.000,$,#39,#40,
     #41,$,#60,#61,#42,2.500,$);
#20= DRILLING($,$,'DRILL␣HOLE1',10.000,$,#44,#45,#41,$,$,$,$,$,#46);
#21= REAMING($,$,'REAM␣HOLE1',10.000,$,#47,#48,#41,$,$,$,$,$,#49,.T.,$,$);
#22= BOTTOM_AND_SIDE_ROUGH_MILLING($,$,'ROUGH␣POCKET1',15.000,$,#39,#50,
     #41,$,$,$,#51,6.500,5.000,1.000,0.500);
#23= BOTTOM_AND_SIDE_FINISH_MILLING($,$,'FINISH␣POCKET1',15.000,$,#39,
     #52,#41,$,$,$,#53,2.000,10.000,$,$);
#24= LINEAR_PATH($,#54,#55);
#25= LINEAR_PROFILE($,#57);
#26= THROUGH_BOTTOM_CONDITION();
#27= PLANAR_POCKET_BOTTOM_CONDITION();
#28= RECTANGULAR_CLOSED_PROFILE($, #122, #121);
#29= TAPERED_ENDMILL(#30,4,.RIGHT.,.F.,$,$);
#30= MILLING_TOOL_DIMENSION(18.000, $,$, 29.0, 0.0, $,$);
#31= TWIST_DRILL(#32,2,.RIGHT.,.F.,0.840);
#32= MILLING_TOOL_DIMENSION(20.000,31.000,0.100,45.000,2.000,
```

```
    5.000,8.000);
#33= TAPERED_REAMER(#34,6,.RIGHT.,.F.,$,$);
#34= MILLING_TOOL_DIMENSION(22.000,$,$,40.0,$,$,$);
#37= TOLERANCED_LENGTH_MEASURE(10.000,#38);
#38= PLUS_MINUS_VALUE(0.100,0.100,3);
#39= MILLING_CUTTING_TOOL('MILL␣18MM',#29,(#125),80.000,$,$);
#40= MILLING_TECHNOLOGY(0.040,.TCP.,$,-12.000,$,.F.,.F.,.F.,$);
#41= MILLING_MACHINE_FUNCTIONS(.T.,$,$,.F.,$,(),.T.,$,$,());
#42= BIDIRECTIONAL_MILLING(0.05, .T., #43, .LEFT., $);
#43= DIRECTION('STRATEGY␣PLANAR␣FACE1:␣1.DIRECTION',(0.000,1.000,0.000));
#44= MILLING_CUTTING_TOOL('SPIRAL_DRILL_20MM',#31,(#126),90.000,$,$);
#45= MILLING_TECHNOLOGY(0.030,.TCP.,$,-16.000,$,.F.,.F.,.F.,$);
#46= DRILLING_TYPE_STRATEGY(0.75, 0.50, 2.000, 0.50,0.75, 8.000);
#47= MILLING_CUTTING_TOOL('REAMER_22MM',#33,(#127),100.000,$,$);
#48= MILLING_TECHNOLOGY(0.030,.TCP.,$,-18.000,$,.F.,.F.,.F.,$);
#49= DRILLING_TYPE_STRATEGY($,$,$,$,$,$);
#50= MILLING_TECHNOLOGY(0.040,.TCP.,$,-20.000,$,.F.,.F.,.F.,$);
#51= CONTOUR_PARALLEL($,$,.CW.,.CONVENTIONAL.);
#52= MILLING_TECHNOLOGY(0.040,.TCP.,$,-20.000,$,.F.,.F.,.F.,$);
#53= CONTOUR_PARALLEL(0.05, .T., .CW., .CONVENTIONAL.);
#54= TOLERANCED_LENGTH_MEASURE(120.000,#56);
#55= DIRECTION('COURSE␣OF␣TRAVEL␣DIRECTION',(0.000,1.000,0.000));
#56= PLUS_MINUS_VALUE(0.300,0.300,3);
#57= NUMERIC_PARAMETER('PROFILE␣LENGTH',100.000,'MM');
#58= TOLERANCED_LENGTH_MEASURE(22.000,#56);
#60= PLUNGE_TOOLAXIS($);
#61= PLUNGE_TOOLAXIS($);
#62= PLANE('SECURITY␣PLANE',#73);
#63= PLANE('PLANAR␣FACE1-DEPTH␣PLANE',#80);
#64= PLANE('DEPTH␣SURFACE␣FOR␣ROUND␣HOLE1',#83);
#65= PLANE('DEPTH␣SURFACE␣FOR␣POCKET1',#94);
#66= CARTESIAN_POINT('CLAMPING_POSITION1',(0.000,20.000,25.000));
#67= CARTESIAN_POINT('CLAMPING_POSITION2',(100.000,20.000,25.000));
#68= CARTESIAN_POINT('CLAMPING_POSITION3',(0.000,100.000,25.000));
#69= CARTESIAN_POINT('CLAMPING_POSITION4',(100.000,100.000,25.000));
```

```
#71= AXIS2_PLACEMENT_3D('SETUP1',#95,#96,#97);
#73= AXIS2_PLACEMENT_3D('PLANE1',#98,#99,#100);
#74= AXIS2_PLACEMENT_3D('WORKPIECE',#101,#102,#103);
#77= AXIS2_PLACEMENT_3D('PLANAR␣FACE1',#104,#105,#106);
#80= AXIS2_PLACEMENT_3D('PLANAR␣FACE1',#107,#108,#109);
#81= AXIS2_PLACEMENT_3D('HOLE1',#110,#111,#114);
#83= AXIS2_PLACEMENT_3D('HOLE1',#112,#113,#114);
#84= AXIS2_PLACEMENT_3D('POCKET1',#115,#116,#117);
#94= AXIS2_PLACEMENT_3D('POCKET1',#118,#119,#120);
#95= CARTESIAN_POINT('SETUP1:␣LOCATION␣',(0.000,0.000,0.000));
#96= DIRECTION('␣AXIS␣',(0.000,0.000,1.000));
#97= DIRECTION('␣REF_DIRECTION',(1.000,0.000,0.000));
#98= CARTESIAN_POINT('SECPLANE1:␣LOCATION␣',(0.000,0.000,30.000));
#99= DIRECTION('␣AXIS␣',(0.000,0.000,1.000));
#100= DIRECTION('␣REF_DIRECTION',(1.000,0.000,0.000));
#101= CARTESIAN_POINT('WORKPIECE1:LOCATION␣',(0.000,0.000,0.000));
#102= DIRECTION('␣AXIS␣',(0.000,0.000,1.000));
#103= DIRECTION('␣REF_DIRECTION',(1.000,0.000,0.000));
#104= CARTESIAN_POINT('PLANAR␣FACE1:LOCATION␣',(0.000,0.000,5.000));
#105= DIRECTION('␣AXIS␣',(0.000,0.000,1.000));
#106= DIRECTION('␣REF_DIRECTION',(1.000,0.000,0.000));
#107= CARTESIAN_POINT('PLANAR␣FACE1:DEPTH␣',(0.000,0.000,-5.000));
#108= DIRECTION('␣AXIS␣',(0.000,0.000,1.000));
#109= DIRECTION('␣REF_DIRECTION',(1.000,0.000,0.000));
#110= CARTESIAN_POINT('HOLE1:␣LOCATION␣',(20.000,60.000,0.000));
#111= DIRECTION('␣AXIS␣',(0.000,0.000,1.000));
#112= CARTESIAN_POINT('HOLE1:␣DEPTH␣',(0.000,0.000,-30.000));
#113= DIRECTION('␣AXIS␣',(0.000,0.000,1.000));
#114= DIRECTION('␣REF_DIRECTION',(1.000,0.000,0.000));
#115= CARTESIAN_POINT('POCKET1:␣LOCATION␣',(70.000, 70.000, 0.000));
#116= DIRECTION('␣AXIS␣',(0.000,0.000,1.000));
#117= DIRECTION('␣REF_DIRECTION',(0.000, 1.000, 0.000));
#118= CARTESIAN_POINT('POCKET1:␣DEPTH␣',(0.000,0.000,-30.000));
#119= DIRECTION('␣AXIS␣',(0.000,0.000,1.000));
#120= DIRECTION('␣REF_DIRECTION',(1.000,0.000,0.000));
```

```
#121= TOLERANCED_LENGTH_MEASURE(80.000,#123);
#122= TOLERANCED_LENGTH_MEASURE(50.000,#124);
#123= PLUS_MINUS_VALUE(0.100,0.100,3);
#124= PLUS_MINUS_VALUE(0.100,0.100,3);
#125= CUTTING_COMPONENT(80.000,$,$,$,$);
#126= CUTTING_COMPONENT(90.000,$,$,$,$);
#127= CUTTING_COMPONENT(100.000,$,$,$,$);
ENDSEC;

END-ISO-10303-21;
```

face1 product

Below is the STEP-NC file for face1 product.

```
ISO-10303-21;

HEADER;
FILE_DESCRIPTION(('ISO␣14649-11␣EXAMPLE␣1',
            'PLANAR_FACE'),
            '1');
FILE_NAME('EXAMPLE1.STP',
            '2003-09-25',
            ('T.␣Kramer'),
            ('NIST'),
            '$',
            'ISO␣14649',
            '$');
FILE_SCHEMA(('MACHINING_SCHEMA','MILLING_SCHEMA'));
ENDSEC;

DATA;
#1= PROJECT('DO␣FACE',#2,(#4),$,$,$);
#2= WORKPLAN('MAIN␣WORKPLAN',(#10),$,#8,$);
#4= WORKPIECE('SIMPLE␣WORKPIECE',#6,0.010,$,$,$,());
```

```
#6= MATERIAL('ST-50','STEEL',(#7));
#7= DESCRIPTIVE_PARAMETER('E=200000N/M2','mild');
#8= SETUP('SETUP1',#71,#62,(#9));
#9= WORKPIECE_SETUP(#4,#74,$,$,());
#10= MACHINING_WORKINGSTEP('WS␣FINISH␣PLANAR␣FACE1',#62,#16,#19,$);
#16= PLANAR_FACE('PLANAR␣FACE1',#4,(#19),#77,#63,#24,#25,$,());
#19= PLANE_FINISH_MILLING($,$,'FINISH␣PLANAR␣FACE1',10.000,$,#39,#40,
     #41,$,#60,#61,#42,2.500,$);
#24= LINEAR_PATH($,#54,#55);
#25= LINEAR_PROFILE($,#57);
#29= TAPERED_ENDMILL(#30,4,.RIGHT.,.F.,$,$);
#30= MILLING_TOOL_DIMENSION(18.000, $,$, 29.0, 0.0, $,$);
#39= MILLING_CUTTING_TOOL('MILL␣18MM',#29,(#125),80.000,$,$);
#40= MILLING_TECHNOLOGY(0.040,.TCP.,$,-12.000,$,.F.,.F.,.F.,$);
#41= MILLING_MACHINE_FUNCTIONS(.T.,$,$,.F.,$,(),.T.,$,$,());
#42= UNIDIRECTIONAL_MILLING(0.05, .T., #43, .CONVENTIONAL.);
#43= DIRECTION('STRATEGY␣PLANAR␣FACE1:␣1.DIRECTION',(1.000,0.000,0.000));
#54= TOLERANCED_LENGTH_MEASURE(120.000,#56);
#55= DIRECTION('COURSE␣OF␣TRAVEL␣DIRECTION',(0.000,1.000,0.000));
#56= PLUS_MINUS_VALUE(0.300,0.300,3);
#57= NUMERIC_PARAMETER('PROFILE␣LENGTH',100.000,'MM');
#60= PLUNGE_TOOLAXIS($);
#61= PLUNGE_TOOLAXIS($);
#62= PLANE('SECURITY␣PLANE',#73);
#63= PLANE('PLANAR␣FACE1-DEPTH␣PLANE',#80);
#71= AXIS2_PLACEMENT_3D('SETUP1',#95,#96,#97);
#73= AXIS2_PLACEMENT_3D('PLANE1',#98,#99,#100);
#74= AXIS2_PLACEMENT_3D('WORKPIECE',#101,#102,#103);
#77= AXIS2_PLACEMENT_3D('PLANAR␣FACE1',#104,#105,#106);
#80= AXIS2_PLACEMENT_3D('PLANAR␣FACE1',#107,#108,#109);
#95= CARTESIAN_POINT('SETUP1:␣LOCATION␣',(0.000,0.000,0.000));
#96= DIRECTION('␣AXIS␣',(0.000,0.000,1.000));
#97= DIRECTION('␣REF_DIRECTION',(1.000,0.000,0.000));
#98= CARTESIAN_POINT('SECPLANE1:␣LOCATION␣',(0.000,0.000,30.000));
#99= DIRECTION('␣AXIS␣',(0.000,0.000,1.000));
```

```
#100= DIRECTION('␣REF_DIRECTION',(1.000,0.000,0.000));
#101= CARTESIAN_POINT('WORKPIECE1:LOCATION␣',(0.000,0.000,0.000));
#102= DIRECTION('␣AXIS␣',(0.000,0.000,1.000));
#103= DIRECTION('␣REF_DIRECTION',(1.000,0.000,0.000));
#104= CARTESIAN_POINT('PLANAR␣FACE1:LOCATION␣',(0.000,0.000,5.000));
#105= DIRECTION('␣AXIS␣',(0.000,0.000,1.000));
#106= DIRECTION('␣REF_DIRECTION',(1.000,0.000,0.000));
#107= CARTESIAN_POINT('PLANAR␣FACE1:DEPTH␣',(0.000,0.000,-5.000));
#108= DIRECTION('␣AXIS␣',(0.000,0.000,1.000));
#109= DIRECTION('␣REF_DIRECTION',(1.000,0.000,0.000));
#125= CUTTING_COMPONENT(80.000,$,$,$,$);
ENDSEC;

END-ISO-10303-21;
```

APPENDIX

C

CANONICAL MACHINING COMMANDS FILES

ex1_comment product

Unoptimized Canonical Machining Commands File

```
USE_LENGTH_UNITS(CANON_UNITS_MM)
SET_ORIGIN_OFFSETS(0.0000, 0.0000, 0.0000)
SET_FEED_REFERENCE(CANON_XYZ)
SPINDLE_RETRACT()
STOP_SPINDLE_TURNING()
USE_TOOL_LENGTH_OFFSET(0.0000)
CHANGE_TOOL(MILL 18MM)
USE_TOOL_LENGTH_OFFSET(50.0000)
FLOOD_ON()
SET_SPINDLE_SPEED(720.0000)
START_SPINDLE_CLOCKWISE()
```

```
SET_FEED_RATE(2400.0000)
COMMENT("WorkingStepID=FINISH␣PLANAR␣FACE1,
␣␣␣OperationName=mill_planar")
STRAIGHT_TRAVERSE(91.9000, -13.5000, 250.0000)
STRAIGHT_TRAVERSE(91.9000, -13.5000, 15.0000)
STRAIGHT_FEED(91.9000, -13.5000, 2.5000)
STRAIGHT_FEED(91.9000, 133.5000, 2.5000)
STRAIGHT_FEED(74.8000, 133.5000, 2.5000)
STRAIGHT_FEED(74.8000, -13.5000, 2.5000)
STRAIGHT_FEED(57.7000, -13.5000, 2.5000)
STRAIGHT_FEED(57.7000, 133.5000, 2.5000)
STRAIGHT_FEED(40.6000, 133.5000, 2.5000)
STRAIGHT_FEED(40.6000, -13.5000, 2.5000)
STRAIGHT_FEED(23.5000, -13.5000, 2.5000)
STRAIGHT_FEED(23.5000, 133.5000, 2.5000)
STRAIGHT_FEED(6.4000, 133.5000, 2.5000)
STRAIGHT_FEED(6.4000, -13.5000, 2.5000)
STRAIGHT_TRAVERSE(6.4000, -13.5000, 15.0000)
STRAIGHT_TRAVERSE(91.9000, -13.5000, 15.0000)
STRAIGHT_FEED(91.9000, -13.5000, 0.0000)
STRAIGHT_FEED(91.9000, 133.5000, 0.0000)
STRAIGHT_FEED(74.8000, 133.5000, 0.0000)
STRAIGHT_FEED(74.8000, -13.5000, 0.0000)
STRAIGHT_FEED(57.7000, -13.5000, 0.0000)
STRAIGHT_FEED(57.7000, 133.5000, 0.0000)
STRAIGHT_FEED(40.6000, 133.5000, 0.0000)
STRAIGHT_FEED(40.6000, -13.5000, 0.0000)
STRAIGHT_FEED(23.5000, -13.5000, 0.0000)
STRAIGHT_FEED(23.5000, 133.5000, 0.0000)
STRAIGHT_FEED(6.4000, 133.5000, 0.0000)
STRAIGHT_FEED(6.4000, -13.5000, 0.0000)
STRAIGHT_TRAVERSE(6.4000, -13.5000, 15.0000)
STOP_SPINDLE_TURNING()
FLOOD_OFF()
SPINDLE_RETRACT()
```

```
46 USE_TOOL_LENGTH_OFFSET(0.0000)
47 CHANGE_TOOL(SPIRAL_DRILL_20MM)
48 USE_TOOL_LENGTH_OFFSET(70.0000)
49 COMMENT("WorkingStepID=HOLE1_D=22MM,OperationName=op_drilling")
50 FLOOD_ON()
51 START_SPINDLE_CLOCKWISE()
52 SET_FEED_RATE(900.0000)
53 STRAIGHT_TRAVERSE(6.4000, -13.5000, 30.0000)
54 STRAIGHT_TRAVERSE(20.0000, 60.0000, 30.0000)
55 STRAIGHT_TRAVERSE(20.0000, 60.0000, 10.0000)
56 STRAIGHT_FEED(20.0000, 60.0000, -9.1378)
57 SET_SPINDLE_SPEED(960.0000)
58 SET_FEED_RATE(1800.0000)
59 STRAIGHT_FEED(20.0000, 60.0000, -15.1378)
60 SET_SPINDLE_SPEED(480.0000)
61 SET_FEED_RATE(1350.0000)
62 STRAIGHT_FEED(20.0000, 60.0000, -37.1378)
63 SET_FEED_RATE(1800.0000)
64 STRAIGHT_FEED(20.0000, 60.0000, 10.0000)
65 STOP_SPINDLE_TURNING()
66 FLOOD_OFF()
67 SPINDLE_RETRACT()
68 USE_TOOL_LENGTH_OFFSET(0.0000)
69 CHANGE_TOOL(REAMER_22MM)
70 USE_TOOL_LENGTH_OFFSET(50.0000)
71 FLOOD_ON()
72 SET_SPINDLE_SPEED(1080.0000)
73 START_SPINDLE_CLOCKWISE()
74 COMMENT("WorkingStepID=HOLE1_D=22MM,OperationName=op_reaming")
75 STRAIGHT_FEED(20.0000, 60.0000, -30.0000)
76 STOP_SPINDLE_TURNING()
77 SET_SPINDLE_SPEED(0.0000)
78 STRAIGHT_FEED(20.0000, 60.0000, 10.0000)
79 FLOOD_OFF()
80 SPINDLE_RETRACT()
```

```
USE_TOOL_LENGTH_OFFSET(0.0000)
CHANGE_TOOL(MILL 18MM)
USE_TOOL_LENGTH_OFFSET(50.0000)
FLOOD_ON()
SET_SPINDLE_SPEED(1200.0000)
START_SPINDLE_CLOCKWISE()
SET_FEED_RATE(2400.0000)
COMMENT("WorkingStepID=ROUGH_POCKET1,
␣␣␣OperationName=mill_rectangular_pocket")
STRAIGHT_TRAVERSE(20.0000, 60.0000, 30.0000)
STRAIGHT_TRAVERSE(64.7540, 50.0685, 30.0000)
STRAIGHT_TRAVERSE(64.7540, 50.0685, 15.0000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -3, -5.9000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -5.9000)
STRAIGHT_FEED(70.0000, 85.0000, -5.9000)
STRAIGHT_FEED(70.0000, 90.0000, -5.9000)
STRAIGHT_FEED(75.0000, 90.0000, -5.9000)
STRAIGHT_FEED(75.0000, 50.0000, -5.9000)
STRAIGHT_FEED(65.0000, 50.0000, -5.9000)
STRAIGHT_FEED(65.0000, 90.0000, -5.9000)
STRAIGHT_FEED(70.0000, 90.0000, -5.9000)
STRAIGHT_FEED(70.0000, 95.0000, -5.9000)
STRAIGHT_FEED(80.0000, 95.0000, -5.9000)
STRAIGHT_FEED(80.0000, 45.0000, -5.9000)
STRAIGHT_FEED(60.0000, 45.0000, -5.9000)
STRAIGHT_FEED(60.0000, 95.0000, -5.9000)
STRAIGHT_FEED(70.0000, 95.0000, -5.9000)
STRAIGHT_FEED(70.0000, 100.0000, -5.9000)
STRAIGHT_FEED(85.0000, 100.0000, -5.9000)
STRAIGHT_FEED(85.0000, 40.0000, -5.9000)
STRAIGHT_FEED(55.0000, 40.0000, -5.9000)
STRAIGHT_FEED(55.0000, 100.0000, -5.9000)
STRAIGHT_FEED(70.0000, 100.0000, -5.9000)
STRAIGHT_FEED(70.0000, 100.0000, 0.0000)
STRAIGHT_TRAVERSE(69.5317, 47.8152, 0.0000)
```

```
STRAIGHT_FEED(69.5317, 47.8152, -5.9000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -11.8000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -11.8000)
STRAIGHT_FEED(70.0000, 85.0000, -11.8000)
STRAIGHT_FEED(70.0000, 90.0000, -11.8000)
STRAIGHT_FEED(75.0000, 90.0000, -11.8000)
STRAIGHT_FEED(75.0000, 50.0000, -11.8000)
STRAIGHT_FEED(65.0000, 50.0000, -11.8000)
STRAIGHT_FEED(65.0000, 90.0000, -11.8000)
STRAIGHT_FEED(70.0000, 90.0000, -11.8000)
STRAIGHT_FEED(70.0000, 95.0000, -11.8000)
STRAIGHT_FEED(80.0000, 95.0000, -11.8000)
STRAIGHT_FEED(80.0000, 45.0000, -11.8000)
STRAIGHT_FEED(60.0000, 45.0000, -11.8000)
STRAIGHT_FEED(60.0000, 95.0000, -11.8000)
STRAIGHT_FEED(70.0000, 95.0000, -11.8000)
STRAIGHT_FEED(70.0000, 100.0000, -11.8000)
STRAIGHT_FEED(85.0000, 100.0000, -11.8000)
STRAIGHT_FEED(85.0000, 40.0000, -11.8000)
STRAIGHT_FEED(55.0000, 40.0000, -11.8000)
STRAIGHT_FEED(55.0000, 100.0000, -11.8000)
STRAIGHT_FEED(70.0000, 100.0000, -11.8000)
STRAIGHT_FEED(70.0000, 100.0000, -5.9000)
STRAIGHT_TRAVERSE(69.5317, 47.8152, -5.9000)
STRAIGHT_FEED(69.5317, 47.8152, -11.8000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -17.7000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -17.7000)
STRAIGHT_FEED(70.0000, 85.0000, -17.7000)
STRAIGHT_FEED(70.0000, 90.0000, -17.7000)
STRAIGHT_FEED(75.0000, 90.0000, -17.7000)
STRAIGHT_FEED(75.0000, 50.0000, -17.7000)
STRAIGHT_FEED(65.0000, 50.0000, -17.7000)
STRAIGHT_FEED(65.0000, 90.0000, -17.7000)
STRAIGHT_FEED(70.0000, 90.0000, -17.7000)
STRAIGHT_FEED(70.0000, 95.0000, -17.7000)
```

```
STRAIGHT_FEED(80.0000, 95.0000, -17.7000)
STRAIGHT_FEED(80.0000, 45.0000, -17.7000)
STRAIGHT_FEED(60.0000, 45.0000, -17.7000)
STRAIGHT_FEED(60.0000, 95.0000, -17.7000)
STRAIGHT_FEED(70.0000, 95.0000, -17.7000)
STRAIGHT_FEED(70.0000, 100.0000, -17.7000)
STRAIGHT_FEED(85.0000, 100.0000, -17.7000)
STRAIGHT_FEED(85.0000, 40.0000, -17.7000)
STRAIGHT_FEED(55.0000, 40.0000, -17.7000)
STRAIGHT_FEED(55.0000, 100.0000, -17.7000)
STRAIGHT_FEED(70.0000, 100.0000, -17.7000)
STRAIGHT_FEED(70.0000, 100.0000, -11.8000)
STRAIGHT_TRAVERSE(69.5317, 47.8152, -11.8000)
STRAIGHT_FEED(69.5317, 47.8152, -17.7000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -23.6000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -23.6000)
STRAIGHT_FEED(70.0000, 85.0000, -23.6000)
STRAIGHT_FEED(70.0000, 90.0000, -23.6000)
STRAIGHT_FEED(75.0000, 90.0000, -23.6000)
STRAIGHT_FEED(75.0000, 50.0000, -23.6000)
STRAIGHT_FEED(65.0000, 50.0000, -23.6000)
STRAIGHT_FEED(65.0000, 90.0000, -23.6000)
STRAIGHT_FEED(70.0000, 90.0000, -23.6000)
STRAIGHT_FEED(70.0000, 95.0000, -23.6000)
STRAIGHT_FEED(80.0000, 95.0000, -23.6000)
STRAIGHT_FEED(80.0000, 45.0000, -23.6000)
STRAIGHT_FEED(60.0000, 45.0000, -23.6000)
STRAIGHT_FEED(60.0000, 95.0000, -23.6000)
STRAIGHT_FEED(70.0000, 95.0000, -23.6000)
STRAIGHT_FEED(70.0000, 100.0000, -23.6000)
STRAIGHT_FEED(85.0000, 100.0000, -23.6000)
STRAIGHT_FEED(85.0000, 40.0000, -23.6000)
STRAIGHT_FEED(55.0000, 40.0000, -23.6000)
STRAIGHT_FEED(55.0000, 100.0000, -23.6000)
STRAIGHT_FEED(70.0000, 100.0000, -23.6000)
```

```
185 STRAIGHT_FEED(70.0000, 100.0000, -17.7000)
186 STRAIGHT_TRAVERSE(69.5317, 47.8152, -17.7000)
187 STRAIGHT_FEED(69.5317, 47.8152, -23.6000)
188 ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -29.5000)
189 ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -29.5000)
190 STRAIGHT_FEED(70.0000, 85.0000, -29.5000)
191 STRAIGHT_FEED(70.0000, 90.0000, -29.5000)
192 STRAIGHT_FEED(75.0000, 90.0000, -29.5000)
193 STRAIGHT_FEED(75.0000, 50.0000, -29.5000)
194 STRAIGHT_FEED(65.0000, 50.0000, -29.5000)
195 STRAIGHT_FEED(65.0000, 90.0000, -29.5000)
196 STRAIGHT_FEED(70.0000, 90.0000, -29.5000)
197 STRAIGHT_FEED(70.0000, 95.0000, -29.5000)
198 STRAIGHT_FEED(80.0000, 95.0000, -29.5000)
199 STRAIGHT_FEED(80.0000, 45.0000, -29.5000)
200 STRAIGHT_FEED(60.0000, 45.0000, -29.5000)
201 STRAIGHT_FEED(60.0000, 95.0000, -29.5000)
202 STRAIGHT_FEED(70.0000, 95.0000, -29.5000)
203 STRAIGHT_FEED(70.0000, 100.0000, -29.5000)
204 STRAIGHT_FEED(85.0000, 100.0000, -29.5000)
205 STRAIGHT_FEED(85.0000, 40.0000, -29.5000)
206 STRAIGHT_FEED(55.0000, 40.0000, -29.5000)
207 STRAIGHT_FEED(55.0000, 100.0000, -29.5000)
208 STRAIGHT_FEED(70.0000, 100.0000, -29.5000)
209 STRAIGHT_FEED(70.0000, 100.0000, 15.0000)
210 COMMENT("WorkingStepID=FINISH␣POCKET1,
␣␣OperationName=mill_rectangular_pocket")
211 STRAIGHT_TRAVERSE(70.0000, 100.0000, 30.0000)
212 STRAIGHT_TRAVERSE(74.8901, 60.2846, 30.0000)
213 STRAIGHT_TRAVERSE(74.8901, 60.2846, 15.0000)
214 ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -3, -2.0000)
215 ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -2.0000)
216 STRAIGHT_FEED(70.0000, 85.0000, -2.0000)
217 STRAIGHT_FEED(70.0000, 93.0000, -2.0000)
218 STRAIGHT_FEED(78.0000, 93.0000, -2.0000)
```

```
STRAIGHT_FEED(78.0000, 47.0000, -2.0000)
STRAIGHT_FEED(62.0000, 47.0000, -2.0000)
STRAIGHT_FEED(62.0000, 93.0000, -2.0000)
STRAIGHT_FEED(70.0000, 93.0000, -2.0000)
STRAIGHT_FEED(70.0000, 101.0000, -2.0000)
STRAIGHT_FEED(85.0000, 101.0000, -2.0000)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -2.0000)
STRAIGHT_FEED(86.0000, 40.0000, -2.0000)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -2.0000)
STRAIGHT_FEED(55.0000, 39.0000, -2.0000)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -2.0000)
STRAIGHT_FEED(54.0000, 100.0000, -2.0000)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -2.0000)
STRAIGHT_FEED(70.0000, 101.0000, -2.0000)
STRAIGHT_FEED(70.0000, 101.0000, 0.0000)
STRAIGHT_TRAVERSE(69.9672, 62.1999, 0.0000)
STRAIGHT_FEED(69.9672, 62.1999, -2.0000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -4.0000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -4.0000)
STRAIGHT_FEED(70.0000, 85.0000, -4.0000)
STRAIGHT_FEED(70.0000, 93.0000, -4.0000)
STRAIGHT_FEED(78.0000, 93.0000, -4.0000)
STRAIGHT_FEED(78.0000, 47.0000, -4.0000)
STRAIGHT_FEED(62.0000, 47.0000, -4.0000)
STRAIGHT_FEED(62.0000, 93.0000, -4.0000)
STRAIGHT_FEED(70.0000, 93.0000, -4.0000)
STRAIGHT_FEED(70.0000, 101.0000, -4.0000)
STRAIGHT_FEED(85.0000, 101.0000, -4.0000)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -4.0000)
STRAIGHT_FEED(86.0000, 40.0000, -4.0000)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -4.0000)
STRAIGHT_FEED(55.0000, 39.0000, -4.0000)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -4.0000)
STRAIGHT_FEED(54.0000, 100.0000, -4.0000)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -4.0000)
```

```
STRAIGHT_FEED(70.0000, 101.0000, -4.0000)
STRAIGHT_FEED(70.0000, 101.0000, -2.0000)
STRAIGHT_TRAVERSE(69.9672, 62.1999, -2.0000)
STRAIGHT_FEED(69.9672, 62.1999, -4.0000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -6.0000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -6.0000)
STRAIGHT_FEED(70.0000, 85.0000, -6.0000)
STRAIGHT_FEED(70.0000, 93.0000, -6.0000)
STRAIGHT_FEED(78.0000, 93.0000, -6.0000)
STRAIGHT_FEED(78.0000, 47.0000, -6.0000)
STRAIGHT_FEED(62.0000, 47.0000, -6.0000)
STRAIGHT_FEED(62.0000, 93.0000, -6.0000)
STRAIGHT_FEED(70.0000, 93.0000, -6.0000)
STRAIGHT_FEED(70.0000, 101.0000, -6.0000)
STRAIGHT_FEED(85.0000, 101.0000, -6.0000)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -6.0000)
STRAIGHT_FEED(86.0000, 40.0000, -6.0000)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -6.0000)
STRAIGHT_FEED(55.0000, 39.0000, -6.0000)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -6.0000)
STRAIGHT_FEED(54.0000, 100.0000, -6.0000)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -6.0000)
STRAIGHT_FEED(70.0000, 101.0000, -6.0000)
STRAIGHT_FEED(70.0000, 101.0000, -4.0000)
STRAIGHT_TRAVERSE(69.9672, 62.1999, -4.0000)
STRAIGHT_FEED(69.9672, 62.1999, -6.0000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -8.0000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -8.0000)
STRAIGHT_FEED(70.0000, 85.0000, -8.0000)
STRAIGHT_FEED(70.0000, 93.0000, -8.0000)
STRAIGHT_FEED(78.0000, 93.0000, -8.0000)
STRAIGHT_FEED(78.0000, 47.0000, -8.0000)
STRAIGHT_FEED(62.0000, 47.0000, -8.0000)
STRAIGHT_FEED(62.0000, 93.0000, -8.0000)
STRAIGHT_FEED(70.0000, 93.0000, -8.0000)
```

```
STRAIGHT_FEED(70.0000, 101.0000, -8.0000)
STRAIGHT_FEED(85.0000, 101.0000, -8.0000)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -8.0000)
STRAIGHT_FEED(86.0000, 40.0000, -8.0000)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -8.0000)
STRAIGHT_FEED(55.0000, 39.0000, -8.0000)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -8.0000)
STRAIGHT_FEED(54.0000, 100.0000, -8.0000)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -8.0000)
STRAIGHT_FEED(70.0000, 101.0000, -8.0000)
STRAIGHT_FEED(70.0000, 101.0000, -6.0000)
STRAIGHT_TRAVERSE(69.9672, 62.1999, -6.0000)
STRAIGHT_FEED(69.9672, 62.1999, -8.0000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -10.0000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -10.0000)
STRAIGHT_FEED(70.0000, 85.0000, -10.0000)
STRAIGHT_FEED(70.0000, 93.0000, -10.0000)
STRAIGHT_FEED(78.0000, 93.0000, -10.0000)
STRAIGHT_FEED(78.0000, 47.0000, -10.0000)
STRAIGHT_FEED(62.0000, 47.0000, -10.0000)
STRAIGHT_FEED(62.0000, 93.0000, -10.0000)
STRAIGHT_FEED(70.0000, 93.0000, -10.0000)
STRAIGHT_FEED(70.0000, 101.0000, -10.0000)
STRAIGHT_FEED(85.0000, 101.0000, -10.0000)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -10.0000)
STRAIGHT_FEED(86.0000, 40.0000, -10.0000)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -10.0000)
STRAIGHT_FEED(55.0000, 39.0000, -10.0000)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -10.0000)
STRAIGHT_FEED(54.0000, 100.0000, -10.0000)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -10.0000)
STRAIGHT_FEED(70.0000, 101.0000, -10.0000)
STRAIGHT_FEED(70.0000, 101.0000, -8.0000)
STRAIGHT_TRAVERSE(69.9672, 62.1999, -8.0000)
STRAIGHT_FEED(69.9672, 62.1999, -10.0000)
```

```
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -12.0000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -12.0000)
STRAIGHT_FEED(70.0000, 85.0000, -12.0000)
STRAIGHT_FEED(70.0000, 93.0000, -12.0000)
STRAIGHT_FEED(78.0000, 93.0000, -12.0000)
STRAIGHT_FEED(78.0000, 47.0000, -12.0000)
STRAIGHT_FEED(62.0000, 47.0000, -12.0000)
STRAIGHT_FEED(62.0000, 93.0000, -12.0000)
STRAIGHT_FEED(70.0000, 93.0000, -12.0000)
STRAIGHT_FEED(70.0000, 101.0000, -12.0000)
STRAIGHT_FEED(85.0000, 101.0000, -12.0000)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -12.0000)
STRAIGHT_FEED(86.0000, 40.0000, -12.0000)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -12.0000)
STRAIGHT_FEED(55.0000, 39.0000, -12.0000)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -12.0000)
STRAIGHT_FEED(54.0000, 100.0000, -12.0000)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -12.0000)
STRAIGHT_FEED(70.0000, 101.0000, -12.0000)
STRAIGHT_FEED(70.0000, 101.0000, -10.0000)
STRAIGHT_TRAVERSE(69.9672, 62.1999, -10.0000)
STRAIGHT_FEED(69.9672, 62.1999, -12.0000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -14.0000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -14.0000)
STRAIGHT_FEED(70.0000, 85.0000, -14.0000)
STRAIGHT_FEED(70.0000, 93.0000, -14.0000)
STRAIGHT_FEED(78.0000, 93.0000, -14.0000)
STRAIGHT_FEED(78.0000, 47.0000, -14.0000)
STRAIGHT_FEED(62.0000, 47.0000, -14.0000)
STRAIGHT_FEED(62.0000, 93.0000, -14.0000)
STRAIGHT_FEED(70.0000, 93.0000, -14.0000)
STRAIGHT_FEED(70.0000, 101.0000, -14.0000)
STRAIGHT_FEED(85.0000, 101.0000, -14.0000)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -14.0000)
STRAIGHT_FEED(86.0000, 40.0000, -14.0000)
```

```
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -14.0000)
STRAIGHT_FEED(55.0000, 39.0000, -14.0000)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -14.0000)
STRAIGHT_FEED(54.0000, 100.0000, -14.0000)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -14.0000)
STRAIGHT_FEED(70.0000, 101.0000, -14.0000)
STRAIGHT_FEED(70.0000, 101.0000, -12.0000)
STRAIGHT_TRAVERSE(69.9672, 62.1999, -12.0000)
STRAIGHT_FEED(69.9672, 62.1999, -14.0000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -16.0000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -16.0000)
STRAIGHT_FEED(70.0000, 85.0000, -16.0000)
STRAIGHT_FEED(70.0000, 93.0000, -16.0000)
STRAIGHT_FEED(78.0000, 93.0000, -16.0000)
STRAIGHT_FEED(78.0000, 47.0000, -16.0000)
STRAIGHT_FEED(62.0000, 47.0000, -16.0000)
STRAIGHT_FEED(62.0000, 93.0000, -16.0000)
STRAIGHT_FEED(70.0000, 93.0000, -16.0000)
STRAIGHT_FEED(70.0000, 101.0000, -16.0000)
STRAIGHT_FEED(85.0000, 101.0000, -16.0000)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -16.0000)
STRAIGHT_FEED(86.0000, 40.0000, -16.0000)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -16.0000)
STRAIGHT_FEED(55.0000, 39.0000, -16.0000)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -16.0000)
STRAIGHT_FEED(54.0000, 100.0000, -16.0000)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -16.0000)
STRAIGHT_FEED(70.0000, 101.0000, -16.0000)
STRAIGHT_FEED(70.0000, 101.0000, -14.0000)
STRAIGHT_TRAVERSE(69.9672, 62.1999, -14.0000)
STRAIGHT_FEED(69.9672, 62.1999, -16.0000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -18.0000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -18.0000)
STRAIGHT_FEED(70.0000, 85.0000, -18.0000)
STRAIGHT_FEED(70.0000, 93.0000, -18.0000)
```

```
STRAIGHT_FEED(78.0000, 93.0000, -18.0000)
STRAIGHT_FEED(78.0000, 47.0000, -18.0000)
STRAIGHT_FEED(62.0000, 47.0000, -18.0000)
STRAIGHT_FEED(62.0000, 93.0000, -18.0000)
STRAIGHT_FEED(70.0000, 93.0000, -18.0000)
STRAIGHT_FEED(70.0000, 101.0000, -18.0000)
STRAIGHT_FEED(85.0000, 101.0000, -18.0000)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -18.0000)
STRAIGHT_FEED(86.0000, 40.0000, -18.0000)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -18.0000)
STRAIGHT_FEED(55.0000, 39.0000, -18.0000)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -18.0000)
STRAIGHT_FEED(54.0000, 100.0000, -18.0000)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -18.0000)
STRAIGHT_FEED(70.0000, 101.0000, -18.0000)
STRAIGHT_FEED(70.0000, 101.0000, -16.0000)
STRAIGHT_TRAVERSE(69.9672, 62.1999, -16.0000)
STRAIGHT_FEED(69.9672, 62.1999, -18.0000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -20.0000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -20.0000)
STRAIGHT_FEED(70.0000, 85.0000, -20.0000)
STRAIGHT_FEED(70.0000, 93.0000, -20.0000)
STRAIGHT_FEED(78.0000, 93.0000, -20.0000)
STRAIGHT_FEED(78.0000, 47.0000, -20.0000)
STRAIGHT_FEED(62.0000, 47.0000, -20.0000)
STRAIGHT_FEED(62.0000, 93.0000, -20.0000)
STRAIGHT_FEED(70.0000, 93.0000, -20.0000)
STRAIGHT_FEED(70.0000, 101.0000, -20.0000)
STRAIGHT_FEED(85.0000, 101.0000, -20.0000)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -20.0000)
STRAIGHT_FEED(86.0000, 40.0000, -20.0000)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -20.0000)
STRAIGHT_FEED(55.0000, 39.0000, -20.0000)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -20.0000)
STRAIGHT_FEED(54.0000, 100.0000, -20.0000)
```

```
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -20.0000)
STRAIGHT_FEED(70.0000, 101.0000, -20.0000)
STRAIGHT_FEED(70.0000, 101.0000, -18.0000)
STRAIGHT_TRAVERSE(69.9672, 62.1999, -18.0000)
STRAIGHT_FEED(69.9672, 62.1999, -20.0000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -22.0000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -22.0000)
STRAIGHT_FEED(70.0000, 85.0000, -22.0000)
STRAIGHT_FEED(70.0000, 93.0000, -22.0000)
STRAIGHT_FEED(78.0000, 93.0000, -22.0000)
STRAIGHT_FEED(78.0000, 47.0000, -22.0000)
STRAIGHT_FEED(62.0000, 47.0000, -22.0000)
STRAIGHT_FEED(62.0000, 93.0000, -22.0000)
STRAIGHT_FEED(70.0000, 93.0000, -22.0000)
STRAIGHT_FEED(70.0000, 101.0000, -22.0000)
STRAIGHT_FEED(85.0000, 101.0000, -22.0000)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -22.0000)
STRAIGHT_FEED(86.0000, 40.0000, -22.0000)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -22.0000)
STRAIGHT_FEED(55.0000, 39.0000, -22.0000)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -22.0000)
STRAIGHT_FEED(54.0000, 100.0000, -22.0000)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -22.0000)
STRAIGHT_FEED(70.0000, 101.0000, -22.0000)
STRAIGHT_FEED(70.0000, 101.0000, -20.0000)
STRAIGHT_TRAVERSE(69.9672, 62.1999, -20.0000)
STRAIGHT_FEED(69.9672, 62.1999, -22.0000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -24.0000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -24.0000)
STRAIGHT_FEED(70.0000, 85.0000, -24.0000)
STRAIGHT_FEED(70.0000, 93.0000, -24.0000)
STRAIGHT_FEED(78.0000, 93.0000, -24.0000)
STRAIGHT_FEED(78.0000, 47.0000, -24.0000)
STRAIGHT_FEED(62.0000, 47.0000, -24.0000)
STRAIGHT_FEED(62.0000, 93.0000, -24.0000)
```

```
STRAIGHT_FEED(70.0000, 93.0000, -24.0000)
STRAIGHT_FEED(70.0000, 101.0000, -24.0000)
STRAIGHT_FEED(85.0000, 101.0000, -24.0000)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -24.0000)
STRAIGHT_FEED(86.0000, 40.0000, -24.0000)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -24.0000)
STRAIGHT_FEED(55.0000, 39.0000, -24.0000)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -24.0000)
STRAIGHT_FEED(54.0000, 100.0000, -24.0000)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -24.0000)
STRAIGHT_FEED(70.0000, 101.0000, -24.0000)
STRAIGHT_FEED(70.0000, 101.0000, -22.0000)
STRAIGHT_TRAVERSE(69.9672, 62.1999, -22.0000)
STRAIGHT_FEED(69.9672, 62.1999, -24.0000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -26.0000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -26.0000)
STRAIGHT_FEED(70.0000, 85.0000, -26.0000)
STRAIGHT_FEED(70.0000, 93.0000, -26.0000)
STRAIGHT_FEED(78.0000, 93.0000, -26.0000)
STRAIGHT_FEED(78.0000, 47.0000, -26.0000)
STRAIGHT_FEED(62.0000, 47.0000, -26.0000)
STRAIGHT_FEED(62.0000, 93.0000, -26.0000)
STRAIGHT_FEED(70.0000, 93.0000, -26.0000)
STRAIGHT_FEED(70.0000, 101.0000, -26.0000)
STRAIGHT_FEED(85.0000, 101.0000, -26.0000)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -26.0000)
STRAIGHT_FEED(86.0000, 40.0000, -26.0000)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -26.0000)
STRAIGHT_FEED(55.0000, 39.0000, -26.0000)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -26.0000)
STRAIGHT_FEED(54.0000, 100.0000, -26.0000)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -26.0000)
STRAIGHT_FEED(70.0000, 101.0000, -26.0000)
STRAIGHT_FEED(70.0000, 101.0000, -24.0000)
STRAIGHT_TRAVERSE(69.9672, 62.1999, -24.0000)
```

```
STRAIGHT_FEED(69.9672, 62.1999, -26.0000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -28.0000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -28.0000)
STRAIGHT_FEED(70.0000, 85.0000, -28.0000)
STRAIGHT_FEED(70.0000, 93.0000, -28.0000)
STRAIGHT_FEED(78.0000, 93.0000, -28.0000)
STRAIGHT_FEED(78.0000, 47.0000, -28.0000)
STRAIGHT_FEED(62.0000, 47.0000, -28.0000)
STRAIGHT_FEED(62.0000, 93.0000, -28.0000)
STRAIGHT_FEED(70.0000, 93.0000, -28.0000)
STRAIGHT_FEED(70.0000, 101.0000, -28.0000)
STRAIGHT_FEED(85.0000, 101.0000, -28.0000)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -28.0000)
STRAIGHT_FEED(86.0000, 40.0000, -28.0000)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -28.0000)
STRAIGHT_FEED(55.0000, 39.0000, -28.0000)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -28.0000)
STRAIGHT_FEED(54.0000, 100.0000, -28.0000)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -28.0000)
STRAIGHT_FEED(70.0000, 101.0000, -28.0000)
STRAIGHT_FEED(70.0000, 101.0000, -26.0000)
STRAIGHT_TRAVERSE(69.9672, 62.1999, -26.0000)
STRAIGHT_FEED(69.9672, 62.1999, -28.0000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -30.0000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -30.0000)
STRAIGHT_FEED(70.0000, 85.0000, -30.0000)
STRAIGHT_FEED(70.0000, 93.0000, -30.0000)
STRAIGHT_FEED(78.0000, 93.0000, -30.0000)
STRAIGHT_FEED(78.0000, 47.0000, -30.0000)
STRAIGHT_FEED(62.0000, 47.0000, -30.0000)
STRAIGHT_FEED(62.0000, 93.0000, -30.0000)
STRAIGHT_FEED(70.0000, 93.0000, -30.0000)
STRAIGHT_FEED(70.0000, 101.0000, -30.0000)
STRAIGHT_FEED(85.0000, 101.0000, -30.0000)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -30.0000)
```

```
534 STRAIGHT_FEED(86.0000, 40.0000, -30.0000)
535 ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -30.0000)
536 STRAIGHT_FEED(55.0000, 39.0000, -30.0000)
537 ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -30.0000)
538 STRAIGHT_FEED(54.0000, 100.0000, -30.0000)
539 ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -30.0000)
540 STRAIGHT_FEED(70.0000, 101.0000, -30.0000)
541 STRAIGHT_FEED(70.0000, 101.0000, 15.0000)
542 PROGRAM_END()
```

Optimized Canonical Machining Commands File

```
  1 USE_LENGTH_UNITS(CANON_UNITS_MM)
  2 SET_ORIGIN_OFFSETS(0.0000, 0.0000, 0.0000)
  3 SET_FEED_REFERENCE(CANON_XYZ)
  4 SPINDLE_RETRACT()
  5 STOP_SPINDLE_TURNING()
  6 USE_TOOL_LENGTH_OFFSET(0.0000)
  7 CHANGE_TOOL(MILL 18MM)
  8 USE_TOOL_LENGTH_OFFSET(50.0000)
  9 FLOOD_ON()
 10 SET_SPINDLE_SPEED(720.0000)
 11 START_SPINDLE_CLOCKWISE()
 12 SET_FEED_RATE(2400.0000)
 13 COMMENT("WorkingStepID=FINISH␣PLANAR␣FACE1,
␣␣␣OperationName=mill_planar")
 14 STRAIGHT_TRAVERSE(91.9000, -13.5000, 250.0000)
 15 STRAIGHT_TRAVERSE(91.9000, -13.5000, 15.0000)
 16 STRAIGHT_FEED(91.9000, -13.5000, 4.0000)
 17 STRAIGHT_FEED(91.9000, 133.5000, 4.0000)
 18 STRAIGHT_FEED(74.8000, 133.5000, 4.0000)
 19 STRAIGHT_FEED(74.8000, -13.5000, 4.0000)
 20 STRAIGHT_FEED(57.7000, -13.5000, 4.0000)
 21 STRAIGHT_FEED(57.7000, 133.5000, 4.0000)
 22 STRAIGHT_FEED(40.6000, 133.5000, 4.0000)
```

```
STRAIGHT_FEED(40.6000, -13.5000, 4.0000)
STRAIGHT_FEED(23.5000, -13.5000, 4.0000)
STRAIGHT_FEED(23.5000, 133.5000, 4.0000)
STRAIGHT_FEED(6.4000, 133.5000, 4.0000)
STRAIGHT_FEED(6.4000, -13.5000, 4.0000)
STRAIGHT_TRAVERSE(6.4000, -13.5000, 15.0000)
STRAIGHT_TRAVERSE(91.9000, -13.5000, 15.0000)
STRAIGHT_FEED(91.9000, -13.5000, 3.0000)
STRAIGHT_FEED(91.9000, 133.5000, 3.0000)
STRAIGHT_FEED(74.8000, 133.5000, 3.0000)
STRAIGHT_FEED(74.8000, -13.5000, 3.0000)
STRAIGHT_FEED(57.7000, -13.5000, 3.0000)
STRAIGHT_FEED(57.7000, 133.5000, 3.0000)
STRAIGHT_FEED(40.6000, 133.5000, 3.0000)
STRAIGHT_FEED(40.6000, -13.5000, 3.0000)
STRAIGHT_FEED(23.5000, -13.5000, 3.0000)
STRAIGHT_FEED(23.5000, 133.5000, 3.0000)
STRAIGHT_FEED(6.4000, 133.5000, 3.0000)
STRAIGHT_FEED(6.4000, -13.5000, 3.0000)
STRAIGHT_TRAVERSE(6.4000, -13.5000, 15.0000)
STRAIGHT_TRAVERSE(91.9000, -13.5000, 15.0000)
STRAIGHT_FEED(91.9000, -13.5000, 2.0000)
STRAIGHT_FEED(91.9000, 133.5000, 2.0000)
STRAIGHT_FEED(74.8000, 133.5000, 2.0000)
STRAIGHT_FEED(74.8000, -13.5000, 2.0000)
STRAIGHT_FEED(57.7000, -13.5000, 2.0000)
STRAIGHT_FEED(57.7000, 133.5000, 2.0000)
STRAIGHT_FEED(40.6000, 133.5000, 2.0000)
STRAIGHT_FEED(40.6000, -13.5000, 2.0000)
STRAIGHT_FEED(23.5000, -13.5000, 2.0000)
STRAIGHT_FEED(23.5000, 133.5000, 2.0000)
STRAIGHT_FEED(6.4000, 133.5000, 2.0000)
STRAIGHT_FEED(6.4000, -13.5000, 2.0000)
STRAIGHT_TRAVERSE(6.4000, -13.5000, 15.0000)
STRAIGHT_TRAVERSE(91.9000, -13.5000, 15.0000)
```

```
STRAIGHT_FEED(91.9000, -13.5000, 1.0000)
STRAIGHT_FEED(91.9000, 133.5000, 1.0000)
STRAIGHT_FEED(74.8000, 133.5000, 1.0000)
STRAIGHT_FEED(74.8000, -13.5000, 1.0000)
STRAIGHT_FEED(57.7000, -13.5000, 1.0000)
STRAIGHT_FEED(57.7000, 133.5000, 1.0000)
STRAIGHT_FEED(40.6000, 133.5000, 1.0000)
STRAIGHT_FEED(40.6000, -13.5000, 1.0000)
STRAIGHT_FEED(23.5000, -13.5000, 1.0000)
STRAIGHT_FEED(23.5000, 133.5000, 1.0000)
STRAIGHT_FEED(6.4000, 133.5000, 1.0000)
STRAIGHT_FEED(6.4000, -13.5000, 1.0000)
STRAIGHT_TRAVERSE(6.4000, -13.5000, 15.0000)
STRAIGHT_TRAVERSE(91.9000, -13.5000, 15.0000)
STRAIGHT_FEED(91.9000, -13.5000, 0.0000)
STRAIGHT_FEED(91.9000, 133.5000, 0.0000)
STRAIGHT_FEED(74.8000, 133.5000, 0.0000)
STRAIGHT_FEED(74.8000, -13.5000, 0.0000)
STRAIGHT_FEED(57.7000, -13.5000, 0.0000)
STRAIGHT_FEED(57.7000, 133.5000, 0.0000)
STRAIGHT_FEED(40.6000, 133.5000, 0.0000)
STRAIGHT_FEED(40.6000, -13.5000, 0.0000)
STRAIGHT_FEED(23.5000, -13.5000, 0.0000)
STRAIGHT_FEED(23.5000, 133.5000, 0.0000)
STRAIGHT_FEED(6.4000, 133.5000, 0.0000)
STRAIGHT_FEED(6.4000, -13.5000, 0.0000)
STRAIGHT_TRAVERSE(6.4000, -13.5000, 15.0000)
STOP_SPINDLE_TURNING()
FLOOD_OFF()
SPINDLE_RETRACT()
USE_TOOL_LENGTH_OFFSET(0.0000)
CHANGE_TOOL(SPIRAL_DRILL_20MM)
USE_TOOL_LENGTH_OFFSET(70.0000)
COMMENT("WorkingStepID=HOLE1_D=22MM,OperationName=op_drilling")
FLOOD_ON()
```

```
93  START_SPINDLE_CLOCKWISE()
94  SET_FEED_RATE(900.0000)
95  STRAIGHT_TRAVERSE(6.4000, -13.5000, 30.0000)
96  STRAIGHT_TRAVERSE(20.0000, 60.0000, 30.0000)
97  STRAIGHT_TRAVERSE(20.0000, 60.0000, 10.0000)
98  STRAIGHT_FEED(20.0000, 60.0000, -9.1378)
99  SET_SPINDLE_SPEED(960.0000)
100 SET_FEED_RATE(1800.0000)
101 STRAIGHT_FEED(20.0000, 60.0000, -15.1378)
102 SET_SPINDLE_SPEED(480.0000)
103 SET_FEED_RATE(1350.0000)
104 STRAIGHT_FEED(20.0000, 60.0000, -37.1378)
105 SET_FEED_RATE(1800.0000)
106 STRAIGHT_FEED(20.0000, 60.0000, 10.0000)
107 STOP_SPINDLE_TURNING()
108 FLOOD_OFF()
109 SPINDLE_RETRACT()
110 USE_TOOL_LENGTH_OFFSET(0.0000)
111 CHANGE_TOOL(REAMER_22MM)
112 USE_TOOL_LENGTH_OFFSET(50.0000)
113 FLOOD_ON()
114 SET_SPINDLE_SPEED(1080.0000)
115 START_SPINDLE_CLOCKWISE()
116 COMMENT("WorkingStepID=HOLE1,_D=22MM,OperationName=op_reaming")
117 STRAIGHT_FEED(20.0000, 60.0000, -30.0000)
118 STOP_SPINDLE_TURNING()
119 SET_SPINDLE_SPEED(0.0000)
120 STRAIGHT_FEED(20.0000, 60.0000, 10.0000)
121 FLOOD_OFF()
122 SPINDLE_RETRACT()
123 USE_TOOL_LENGTH_OFFSET(0.0000)
124 CHANGE_TOOL(MILL_18MM)
125 USE_TOOL_LENGTH_OFFSET(50.0000)
126 FLOOD_ON()
127 SET_SPINDLE_SPEED(1200.0000)
```

```
128 START_SPINDLE_CLOCKWISE()
129 SET_FEED_RATE(2400.0000)
130 COMMENT("WorkingStepID=ROUGH␣POCKET1,
␣␣OperationName=mill_rectangular_pocket")
131 STRAIGHT_TRAVERSE(20.0000, 60.0000, 30.0000)
132 STRAIGHT_TRAVERSE(68.1502, 61.9583, 30.0000)
133 STRAIGHT_TRAVERSE(68.1502, 61.9583, 15.0000)
134 ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -3, -3.2778)
135 ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -3.2778)
136 STRAIGHT_FEED(70.0000, 85.0000, -3.2778)
137 STRAIGHT_FEED(70.0000, 90.0000, -3.2778)
138 STRAIGHT_FEED(75.0000, 90.0000, -3.2778)
139 STRAIGHT_FEED(75.0000, 50.0000, -3.2778)
140 STRAIGHT_FEED(65.0000, 50.0000, -3.2778)
141 STRAIGHT_FEED(65.0000, 90.0000, -3.2778)
142 STRAIGHT_FEED(70.0000, 90.0000, -3.2778)
143 STRAIGHT_FEED(70.0000, 95.0000, -3.2778)
144 STRAIGHT_FEED(80.0000, 95.0000, -3.2778)
145 STRAIGHT_FEED(80.0000, 45.0000, -3.2778)
146 STRAIGHT_FEED(60.0000, 45.0000, -3.2778)
147 STRAIGHT_FEED(60.0000, 95.0000, -3.2778)
148 STRAIGHT_FEED(70.0000, 95.0000, -3.2778)
149 STRAIGHT_FEED(70.0000, 100.0000, -3.2778)
150 STRAIGHT_FEED(85.0000, 100.0000, -3.2778)
151 STRAIGHT_FEED(85.0000, 40.0000, -3.2778)
152 STRAIGHT_FEED(55.0000, 40.0000, -3.2778)
153 STRAIGHT_FEED(55.0000, 100.0000, -3.2778)
154 STRAIGHT_FEED(70.0000, 100.0000, -3.2778)
155 STRAIGHT_FEED(70.0000, 100.0000, 0.0000)
156 STRAIGHT_TRAVERSE(63.8988, 58.8231, 0.0000)
157 STRAIGHT_FEED(63.8988, 58.8231, -3.2778)
158 ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -6.5556)
159 ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -6.5556)
160 STRAIGHT_FEED(70.0000, 85.0000, -6.5556)
161 STRAIGHT_FEED(70.0000, 90.0000, -6.5556)
```

```
STRAIGHT_FEED(75.0000, 90.0000, -6.5556)
STRAIGHT_FEED(75.0000, 50.0000, -6.5556)
STRAIGHT_FEED(65.0000, 50.0000, -6.5556)
STRAIGHT_FEED(65.0000, 90.0000, -6.5556)
STRAIGHT_FEED(70.0000, 90.0000, -6.5556)
STRAIGHT_FEED(70.0000, 95.0000, -6.5556)
STRAIGHT_FEED(80.0000, 95.0000, -6.5556)
STRAIGHT_FEED(80.0000, 45.0000, -6.5556)
STRAIGHT_FEED(60.0000, 45.0000, -6.5556)
STRAIGHT_FEED(60.0000, 95.0000, -6.5556)
STRAIGHT_FEED(70.0000, 95.0000, -6.5556)
STRAIGHT_FEED(70.0000, 100.0000, -6.5556)
STRAIGHT_FEED(85.0000, 100.0000, -6.5556)
STRAIGHT_FEED(85.0000, 40.0000, -6.5556)
STRAIGHT_FEED(55.0000, 40.0000, -6.5556)
STRAIGHT_FEED(55.0000, 100.0000, -6.5556)
STRAIGHT_FEED(70.0000, 100.0000, -6.5556)
STRAIGHT_FEED(70.0000, 100.0000, -3.2778)
STRAIGHT_TRAVERSE(63.8988, 58.8231, -3.2778)
STRAIGHT_FEED(63.8988, 58.8231, -6.5556)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -9.8333)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -9.8333)
STRAIGHT_FEED(70.0000, 85.0000, -9.8333)
STRAIGHT_FEED(70.0000, 90.0000, -9.8333)
STRAIGHT_FEED(75.0000, 90.0000, -9.8333)
STRAIGHT_FEED(75.0000, 50.0000, -9.8333)
STRAIGHT_FEED(65.0000, 50.0000, -9.8333)
STRAIGHT_FEED(65.0000, 90.0000, -9.8333)
STRAIGHT_FEED(70.0000, 90.0000, -9.8333)
STRAIGHT_FEED(70.0000, 95.0000, -9.8333)
STRAIGHT_FEED(80.0000, 95.0000, -9.8333)
STRAIGHT_FEED(80.0000, 45.0000, -9.8333)
STRAIGHT_FEED(60.0000, 45.0000, -9.8333)
STRAIGHT_FEED(60.0000, 95.0000, -9.8333)
STRAIGHT_FEED(70.0000, 95.0000, -9.8333)
```

```
STRAIGHT_FEED(70.0000, 100.0000, -9.8333)
STRAIGHT_FEED(85.0000, 100.0000, -9.8333)
STRAIGHT_FEED(85.0000, 40.0000, -9.8333)
STRAIGHT_FEED(55.0000, 40.0000, -9.8333)
STRAIGHT_FEED(55.0000, 100.0000, -9.8333)
STRAIGHT_FEED(70.0000, 100.0000, -9.8333)
STRAIGHT_FEED(70.0000, 100.0000, -6.5556)
STRAIGHT_TRAVERSE(63.8988, 58.8231, -6.5556)
STRAIGHT_FEED(63.8988, 58.8231, -9.8333)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -13.1111)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -13.1111)
STRAIGHT_FEED(70.0000, 85.0000, -13.1111)
STRAIGHT_FEED(70.0000, 90.0000, -13.1111)
STRAIGHT_FEED(75.0000, 90.0000, -13.1111)
STRAIGHT_FEED(75.0000, 50.0000, -13.1111)
STRAIGHT_FEED(65.0000, 50.0000, -13.1111)
STRAIGHT_FEED(65.0000, 90.0000, -13.1111)
STRAIGHT_FEED(70.0000, 90.0000, -13.1111)
STRAIGHT_FEED(70.0000, 95.0000, -13.1111)
STRAIGHT_FEED(80.0000, 95.0000, -13.1111)
STRAIGHT_FEED(80.0000, 45.0000, -13.1111)
STRAIGHT_FEED(60.0000, 45.0000, -13.1111)
STRAIGHT_FEED(60.0000, 95.0000, -13.1111)
STRAIGHT_FEED(70.0000, 95.0000, -13.1111)
STRAIGHT_FEED(70.0000, 100.0000, -13.1111)
STRAIGHT_FEED(85.0000, 100.0000, -13.1111)
STRAIGHT_FEED(85.0000, 40.0000, -13.1111)
STRAIGHT_FEED(55.0000, 40.0000, -13.1111)
STRAIGHT_FEED(55.0000, 100.0000, -13.1111)
STRAIGHT_FEED(70.0000, 100.0000, -13.1111)
STRAIGHT_FEED(70.0000, 100.0000, -9.8333)
STRAIGHT_TRAVERSE(63.8988, 58.8231, -9.8333)
STRAIGHT_FEED(63.8988, 58.8231, -13.1111)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -16.3889)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -16.3889)
```

```
STRAIGHT_FEED(70.0000, 85.0000, -16.3889)
STRAIGHT_FEED(70.0000, 90.0000, -16.3889)
STRAIGHT_FEED(75.0000, 90.0000, -16.3889)
STRAIGHT_FEED(75.0000, 50.0000, -16.3889)
STRAIGHT_FEED(65.0000, 50.0000, -16.3889)
STRAIGHT_FEED(65.0000, 90.0000, -16.3889)
STRAIGHT_FEED(70.0000, 90.0000, -16.3889)
STRAIGHT_FEED(70.0000, 95.0000, -16.3889)
STRAIGHT_FEED(80.0000, 95.0000, -16.3889)
STRAIGHT_FEED(80.0000, 45.0000, -16.3889)
STRAIGHT_FEED(60.0000, 45.0000, -16.3889)
STRAIGHT_FEED(60.0000, 95.0000, -16.3889)
STRAIGHT_FEED(70.0000, 95.0000, -16.3889)
STRAIGHT_FEED(70.0000, 100.0000, -16.3889)
STRAIGHT_FEED(85.0000, 100.0000, -16.3889)
STRAIGHT_FEED(85.0000, 40.0000, -16.3889)
STRAIGHT_FEED(55.0000, 40.0000, -16.3889)
STRAIGHT_FEED(55.0000, 100.0000, -16.3889)
STRAIGHT_FEED(70.0000, 100.0000, -16.3889)
STRAIGHT_FEED(70.0000, 100.0000, -13.1111)
STRAIGHT_TRAVERSE(63.8988, 58.8231, -13.1111)
STRAIGHT_FEED(63.8988, 58.8231, -16.3889)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -19.6667)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -19.6667)
STRAIGHT_FEED(70.0000, 85.0000, -19.6667)
STRAIGHT_FEED(70.0000, 90.0000, -19.6667)
STRAIGHT_FEED(75.0000, 90.0000, -19.6667)
STRAIGHT_FEED(75.0000, 50.0000, -19.6667)
STRAIGHT_FEED(65.0000, 50.0000, -19.6667)
STRAIGHT_FEED(65.0000, 90.0000, -19.6667)
STRAIGHT_FEED(70.0000, 90.0000, -19.6667)
STRAIGHT_FEED(70.0000, 95.0000, -19.6667)
STRAIGHT_FEED(80.0000, 95.0000, -19.6667)
STRAIGHT_FEED(80.0000, 45.0000, -19.6667)
STRAIGHT_FEED(60.0000, 45.0000, -19.6667)
```

```
STRAIGHT_FEED(60.0000, 95.0000, -19.6667)
STRAIGHT_FEED(70.0000, 95.0000, -19.6667)
STRAIGHT_FEED(70.0000, 100.0000, -19.6667)
STRAIGHT_FEED(85.0000, 100.0000, -19.6667)
STRAIGHT_FEED(85.0000, 40.0000, -19.6667)
STRAIGHT_FEED(55.0000, 40.0000, -19.6667)
STRAIGHT_FEED(55.0000, 100.0000, -19.6667)
STRAIGHT_FEED(70.0000, 100.0000, -19.6667)
STRAIGHT_FEED(70.0000, 100.0000, -16.3889)
STRAIGHT_TRAVERSE(63.8988, 58.8231, -16.3889)
STRAIGHT_FEED(63.8988, 58.8231, -19.6667)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -22.9444)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -22.9444)
STRAIGHT_FEED(70.0000, 85.0000, -22.9444)
STRAIGHT_FEED(70.0000, 90.0000, -22.9444)
STRAIGHT_FEED(75.0000, 90.0000, -22.9444)
STRAIGHT_FEED(75.0000, 50.0000, -22.9444)
STRAIGHT_FEED(65.0000, 50.0000, -22.9444)
STRAIGHT_FEED(65.0000, 90.0000, -22.9444)
STRAIGHT_FEED(70.0000, 90.0000, -22.9444)
STRAIGHT_FEED(70.0000, 95.0000, -22.9444)
STRAIGHT_FEED(80.0000, 95.0000, -22.9444)
STRAIGHT_FEED(80.0000, 45.0000, -22.9444)
STRAIGHT_FEED(60.0000, 45.0000, -22.9444)
STRAIGHT_FEED(60.0000, 95.0000, -22.9444)
STRAIGHT_FEED(70.0000, 95.0000, -22.9444)
STRAIGHT_FEED(70.0000, 100.0000, -22.9444)
STRAIGHT_FEED(85.0000, 100.0000, -22.9444)
STRAIGHT_FEED(85.0000, 40.0000, -22.9444)
STRAIGHT_FEED(55.0000, 40.0000, -22.9444)
STRAIGHT_FEED(55.0000, 100.0000, -22.9444)
STRAIGHT_FEED(70.0000, 100.0000, -22.9444)
STRAIGHT_FEED(70.0000, 100.0000, -19.6667)
STRAIGHT_TRAVERSE(63.8988, 58.8231, -19.6667)
STRAIGHT_FEED(63.8988, 58.8231, -22.9444)
```

```
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -26.2222)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -26.2222)
STRAIGHT_FEED(70.0000, 85.0000, -26.2222)
STRAIGHT_FEED(70.0000, 90.0000, -26.2222)
STRAIGHT_FEED(75.0000, 90.0000, -26.2222)
STRAIGHT_FEED(75.0000, 50.0000, -26.2222)
STRAIGHT_FEED(65.0000, 50.0000, -26.2222)
STRAIGHT_FEED(65.0000, 90.0000, -26.2222)
STRAIGHT_FEED(70.0000, 90.0000, -26.2222)
STRAIGHT_FEED(70.0000, 95.0000, -26.2222)
STRAIGHT_FEED(80.0000, 95.0000, -26.2222)
STRAIGHT_FEED(80.0000, 45.0000, -26.2222)
STRAIGHT_FEED(60.0000, 45.0000, -26.2222)
STRAIGHT_FEED(60.0000, 95.0000, -26.2222)
STRAIGHT_FEED(70.0000, 95.0000, -26.2222)
STRAIGHT_FEED(70.0000, 100.0000, -26.2222)
STRAIGHT_FEED(85.0000, 100.0000, -26.2222)
STRAIGHT_FEED(85.0000, 40.0000, -26.2222)
STRAIGHT_FEED(55.0000, 40.0000, -26.2222)
STRAIGHT_FEED(55.0000, 100.0000, -26.2222)
STRAIGHT_FEED(70.0000, 100.0000, -26.2222)
STRAIGHT_FEED(70.0000, 100.0000, -22.9444)
STRAIGHT_TRAVERSE(63.8988, 58.8231, -22.9444)
STRAIGHT_FEED(63.8988, 58.8231, -26.2222)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -29.5000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -29.5000)
STRAIGHT_FEED(70.0000, 85.0000, -29.5000)
STRAIGHT_FEED(70.0000, 90.0000, -29.5000)
STRAIGHT_FEED(75.0000, 90.0000, -29.5000)
STRAIGHT_FEED(75.0000, 50.0000, -29.5000)
STRAIGHT_FEED(65.0000, 50.0000, -29.5000)
STRAIGHT_FEED(65.0000, 90.0000, -29.5000)
STRAIGHT_FEED(70.0000, 90.0000, -29.5000)
STRAIGHT_FEED(70.0000, 95.0000, -29.5000)
STRAIGHT_FEED(80.0000, 95.0000, -29.5000)
```

```
STRAIGHT_FEED(80.0000, 45.0000, -29.5000)
STRAIGHT_FEED(60.0000, 45.0000, -29.5000)
STRAIGHT_FEED(60.0000, 95.0000, -29.5000)
STRAIGHT_FEED(70.0000, 95.0000, -29.5000)
STRAIGHT_FEED(70.0000, 100.0000, -29.5000)
STRAIGHT_FEED(85.0000, 100.0000, -29.5000)
STRAIGHT_FEED(85.0000, 40.0000, -29.5000)
STRAIGHT_FEED(55.0000, 40.0000, -29.5000)
STRAIGHT_FEED(55.0000, 100.0000, -29.5000)
STRAIGHT_FEED(70.0000, 100.0000, -29.5000)
STRAIGHT_FEED(70.0000, 100.0000, 15.0000)
COMMENT("WorkingStepID=FINISH␣POCKET1,
␣␣OperationName=mill_rectangular_pocket")
STRAIGHT_TRAVERSE(70.0000, 100.0000, 30.0000)
STRAIGHT_TRAVERSE(63.7411, 58.5590, 30.0000)
STRAIGHT_TRAVERSE(63.7411, 58.5590, 15.0000)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -3, -4.2857)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -4.2857)
STRAIGHT_FEED(70.0000, 85.0000, -4.2857)
STRAIGHT_FEED(70.0000, 93.0000, -4.2857)
STRAIGHT_FEED(78.0000, 93.0000, -4.2857)
STRAIGHT_FEED(78.0000, 47.0000, -4.2857)
STRAIGHT_FEED(62.0000, 47.0000, -4.2857)
STRAIGHT_FEED(62.0000, 93.0000, -4.2857)
STRAIGHT_FEED(70.0000, 93.0000, -4.2857)
STRAIGHT_FEED(70.0000, 101.0000, -4.2857)
STRAIGHT_FEED(85.0000, 101.0000, -4.2857)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -4.2857)
STRAIGHT_FEED(86.0000, 40.0000, -4.2857)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -4.2857)
STRAIGHT_FEED(55.0000, 39.0000, -4.2857)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -4.2857)
STRAIGHT_FEED(54.0000, 100.0000, -4.2857)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -4.2857)
STRAIGHT_FEED(70.0000, 101.0000, -4.2857)
```

```
STRAIGHT_FEED(70.0000, 101.0000, 0.0000)
STRAIGHT_TRAVERSE(62.9965, 53.3293, 0.0000)
STRAIGHT_FEED(62.9965, 53.3293, -4.2857)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -8.5714)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -8.5714)
STRAIGHT_FEED(70.0000, 85.0000, -8.5714)
STRAIGHT_FEED(70.0000, 93.0000, -8.5714)
STRAIGHT_FEED(78.0000, 93.0000, -8.5714)
STRAIGHT_FEED(78.0000, 47.0000, -8.5714)
STRAIGHT_FEED(62.0000, 47.0000, -8.5714)
STRAIGHT_FEED(62.0000, 93.0000, -8.5714)
STRAIGHT_FEED(70.0000, 93.0000, -8.5714)
STRAIGHT_FEED(70.0000, 101.0000, -8.5714)
STRAIGHT_FEED(85.0000, 101.0000, -8.5714)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -8.5714)
STRAIGHT_FEED(86.0000, 40.0000, -8.5714)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -8.5714)
STRAIGHT_FEED(55.0000, 39.0000, -8.5714)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -8.5714)
STRAIGHT_FEED(54.0000, 100.0000, -8.5714)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -8.5714)
STRAIGHT_FEED(70.0000, 101.0000, -8.5714)
STRAIGHT_FEED(70.0000, 101.0000, -4.2857)
STRAIGHT_TRAVERSE(62.9965, 53.3293, -4.2857)
STRAIGHT_FEED(62.9965, 53.3293, -8.5714)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -12.8571)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -12.8571)
STRAIGHT_FEED(70.0000, 85.0000, -12.8571)
STRAIGHT_FEED(70.0000, 93.0000, -12.8571)
STRAIGHT_FEED(78.0000, 93.0000, -12.8571)
STRAIGHT_FEED(78.0000, 47.0000, -12.8571)
STRAIGHT_FEED(62.0000, 47.0000, -12.8571)
STRAIGHT_FEED(62.0000, 93.0000, -12.8571)
STRAIGHT_FEED(70.0000, 93.0000, -12.8571)
STRAIGHT_FEED(70.0000, 101.0000, -12.8571)
```

```
STRAIGHT_FEED(85.0000, 101.0000, -12.8571)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -12.8571)
STRAIGHT_FEED(86.0000, 40.0000, -12.8571)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -12.8571)
STRAIGHT_FEED(55.0000, 39.0000, -12.8571)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -12.8571)
STRAIGHT_FEED(54.0000, 100.0000, -12.8571)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -12.8571)
STRAIGHT_FEED(70.0000, 101.0000, -12.8571)
STRAIGHT_FEED(70.0000, 101.0000, -8.5714)
STRAIGHT_TRAVERSE(62.9965, 53.3293, -8.5714)
STRAIGHT_FEED(62.9965, 53.3293, -12.8571)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -17.1429)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -17.1429)
STRAIGHT_FEED(70.0000, 85.0000, -17.1429)
STRAIGHT_FEED(70.0000, 93.0000, -17.1429)
STRAIGHT_FEED(78.0000, 93.0000, -17.1429)
STRAIGHT_FEED(78.0000, 47.0000, -17.1429)
STRAIGHT_FEED(62.0000, 47.0000, -17.1429)
STRAIGHT_FEED(62.0000, 93.0000, -17.1429)
STRAIGHT_FEED(70.0000, 93.0000, -17.1429)
STRAIGHT_FEED(70.0000, 101.0000, -17.1429)
STRAIGHT_FEED(85.0000, 101.0000, -17.1429)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -17.1429)
STRAIGHT_FEED(86.0000, 40.0000, -17.1429)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -17.1429)
STRAIGHT_FEED(55.0000, 39.0000, -17.1429)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -17.1429)
STRAIGHT_FEED(54.0000, 100.0000, -17.1429)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -17.1429)
STRAIGHT_FEED(70.0000, 101.0000, -17.1429)
STRAIGHT_FEED(70.0000, 101.0000, -12.8571)
STRAIGHT_TRAVERSE(62.9965, 53.3293, -12.8571)
STRAIGHT_FEED(62.9965, 53.3293, -17.1429)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -21.4286)
```

```
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -21.4286)
STRAIGHT_FEED(70.0000, 85.0000, -21.4286)
STRAIGHT_FEED(70.0000, 93.0000, -21.4286)
STRAIGHT_FEED(78.0000, 93.0000, -21.4286)
STRAIGHT_FEED(78.0000, 47.0000, -21.4286)
STRAIGHT_FEED(62.0000, 47.0000, -21.4286)
STRAIGHT_FEED(62.0000, 93.0000, -21.4286)
STRAIGHT_FEED(70.0000, 93.0000, -21.4286)
STRAIGHT_FEED(70.0000, 101.0000, -21.4286)
STRAIGHT_FEED(85.0000, 101.0000, -21.4286)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -21.4286)
STRAIGHT_FEED(86.0000, 40.0000, -21.4286)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -21.4286)
STRAIGHT_FEED(55.0000, 39.0000, -21.4286)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -21.4286)
STRAIGHT_FEED(54.0000, 100.0000, -21.4286)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -21.4286)
STRAIGHT_FEED(70.0000, 101.0000, -21.4286)
STRAIGHT_FEED(70.0000, 101.0000, -17.1429)
STRAIGHT_TRAVERSE(62.9965, 53.3293, -17.1429)
STRAIGHT_FEED(62.9965, 53.3293, -21.4286)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -25.7143)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -25.7143)
STRAIGHT_FEED(70.0000, 85.0000, -25.7143)
STRAIGHT_FEED(70.0000, 93.0000, -25.7143)
STRAIGHT_FEED(78.0000, 93.0000, -25.7143)
STRAIGHT_FEED(78.0000, 47.0000, -25.7143)
STRAIGHT_FEED(62.0000, 47.0000, -25.7143)
STRAIGHT_FEED(62.0000, 93.0000, -25.7143)
STRAIGHT_FEED(70.0000, 93.0000, -25.7143)
STRAIGHT_FEED(70.0000, 101.0000, -25.7143)
STRAIGHT_FEED(85.0000, 101.0000, -25.7143)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -25.7143)
STRAIGHT_FEED(86.0000, 40.0000, -25.7143)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -25.7143)
```

```
STRAIGHT_FEED(55.0000, 39.0000, -25.7143)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -25.7143)
STRAIGHT_FEED(54.0000, 100.0000, -25.7143)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -25.7143)
STRAIGHT_FEED(70.0000, 101.0000, -25.7143)
STRAIGHT_FEED(70.0000, 101.0000, -21.4286)
STRAIGHT_TRAVERSE(62.9965, 53.3293, -21.4286)
STRAIGHT_FEED(62.9965, 53.3293, -25.7143)
ARC_FEED(77.2000, 55.0000, 70.0000, 55.0000, -1, -30.0000)
ARC_FEED(70.0000, 55.0000, 73.6000, 55.0000, -1, -30.0000)
STRAIGHT_FEED(70.0000, 85.0000, -30.0000)
STRAIGHT_FEED(70.0000, 93.0000, -30.0000)
STRAIGHT_FEED(78.0000, 93.0000, -30.0000)
STRAIGHT_FEED(78.0000, 47.0000, -30.0000)
STRAIGHT_FEED(62.0000, 47.0000, -30.0000)
STRAIGHT_FEED(62.0000, 93.0000, -30.0000)
STRAIGHT_FEED(70.0000, 93.0000, -30.0000)
STRAIGHT_FEED(70.0000, 101.0000, -30.0000)
STRAIGHT_FEED(85.0000, 101.0000, -30.0000)
ARC_FEED(86.0000, 100.0000, 85.0000, 100.0000, -1, -30.0000)
STRAIGHT_FEED(86.0000, 40.0000, -30.0000)
ARC_FEED(85.0000, 39.0000, 85.0000, 40.0000, -1, -30.0000)
STRAIGHT_FEED(55.0000, 39.0000, -30.0000)
ARC_FEED(54.0000, 40.0000, 55.0000, 40.0000, -1, -30.0000)
STRAIGHT_FEED(54.0000, 100.0000, -30.0000)
ARC_FEED(55.0000, 101.0000, 55.0000, 100.0000, -1, -30.0000)
STRAIGHT_FEED(70.0000, 101.0000, -30.0000)
STRAIGHT_FEED(70.0000, 101.0000, 15.0000)
PROGRAM_END()
```

face1 product

Unoptimized Canonical Machining Commands File

```
USE_LENGTH_UNITS(CANON_UNITS_MM)
SET_ORIGIN_OFFSETS(0.0000, 0.0000, 0.0000)
SET_FEED_REFERENCE(CANON_XYZ)
SPINDLE_RETRACT()
STOP_SPINDLE_TURNING()
FLOOD_OFF()
SPINDLE_RETRACT()
USE_TOOL_LENGTH_OFFSET(0.0000)
CHANGE_TOOL(MILL 18MM)
USE_TOOL_LENGTH_OFFSET(50.0000)
FLOOD_ON()
SET_SPINDLE_SPEED(720.0000)
START_SPINDLE_CLOCKWISE()
COMMENT("WorkingStepID=FINISH␣PLANAR␣FACE1,
␣␣␣OperationName=mill_planar")
STRAIGHT_TRAVERSE(-13.5000, 8.1000, 200.0000)
STRAIGHT_TRAVERSE(-13.5000, 8.1000, 15.0000)
STRAIGHT_FEED(-13.5000, 8.1000, 2.5000)
STRAIGHT_FEED(113.5000, 8.1000, 2.5000)
STRAIGHT_TRAVERSE(113.5000, 8.1000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 25.2000, 15.0000)
STRAIGHT_FEED(-13.5000, 25.2000, 2.5000)
STRAIGHT_FEED(113.5000, 25.2000, 2.5000)
STRAIGHT_TRAVERSE(113.5000, 25.2000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 42.3000, 15.0000)
STRAIGHT_FEED(-13.5000, 42.3000, 2.5000)
STRAIGHT_FEED(113.5000, 42.3000, 2.5000)
STRAIGHT_TRAVERSE(113.5000, 42.3000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 59.4000, 15.0000)
STRAIGHT_FEED(-13.5000, 59.4000, 2.5000)
STRAIGHT_FEED(113.5000, 59.4000, 2.5000)
STRAIGHT_TRAVERSE(113.5000, 59.4000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 76.5000, 15.0000)
STRAIGHT_FEED(-13.5000, 76.5000, 2.5000)
STRAIGHT_FEED(113.5000, 76.5000, 2.5000)
```

```
STRAIGHT_TRAVERSE(113.5000, 76.5000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 93.6000, 15.0000)
STRAIGHT_FEED(-13.5000, 93.6000, 2.5000)
STRAIGHT_FEED(113.5000, 93.6000, 2.5000)
STRAIGHT_TRAVERSE(113.5000, 93.6000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 110.7000, 15.0000)
STRAIGHT_FEED(-13.5000, 110.7000, 2.5000)
STRAIGHT_FEED(113.5000, 110.7000, 2.5000)
STRAIGHT_TRAVERSE(113.5000, 110.7000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 127.8000, 15.0000)
STRAIGHT_FEED(-13.5000, 127.8000, 2.5000)
STRAIGHT_FEED(113.5000, 127.8000, 2.5000)
STRAIGHT_TRAVERSE(113.5000, 127.8000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 8.1000, 15.0000)
STRAIGHT_FEED(-13.5000, 8.1000, 0.0000)
STRAIGHT_FEED(113.5000, 8.1000, 0.0000)
STRAIGHT_TRAVERSE(113.5000, 8.1000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 25.2000, 15.0000)
STRAIGHT_FEED(-13.5000, 25.2000, 0.0000)
STRAIGHT_FEED(113.5000, 25.2000, 0.0000)
STRAIGHT_TRAVERSE(113.5000, 25.2000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 42.3000, 15.0000)
STRAIGHT_FEED(-13.5000, 42.3000, 0.0000)
STRAIGHT_FEED(113.5000, 42.3000, 0.0000)
STRAIGHT_TRAVERSE(113.5000, 42.3000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 59.4000, 15.0000)
STRAIGHT_FEED(-13.5000, 59.4000, 0.0000)
STRAIGHT_FEED(113.5000, 59.4000, 0.0000)
STRAIGHT_TRAVERSE(113.5000, 59.4000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 76.5000, 15.0000)
STRAIGHT_FEED(-13.5000, 76.5000, 0.0000)
STRAIGHT_FEED(113.5000, 76.5000, 0.0000)
STRAIGHT_TRAVERSE(113.5000, 76.5000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 93.6000, 15.0000)
STRAIGHT_FEED(-13.5000, 93.6000, 0.0000)
```

```
70 STRAIGHT_FEED(113.5000, 93.6000, 0.0000)
71 STRAIGHT_TRAVERSE(113.5000, 93.6000, 15.0000)
72 STRAIGHT_TRAVERSE(-13.5000, 110.7000, 15.0000)
73 STRAIGHT_FEED(-13.5000, 110.7000, 0.0000)
74 STRAIGHT_FEED(113.5000, 110.7000, 0.0000)
75 STRAIGHT_TRAVERSE(113.5000, 110.7000, 15.0000)
76 STRAIGHT_TRAVERSE(-13.5000, 127.8000, 15.0000)
77 STRAIGHT_FEED(-13.5000, 127.8000, 0.0000)
78 STRAIGHT_FEED(113.5000, 127.8000, 0.0000)
79 STRAIGHT_TRAVERSE(113.5000, 127.8000, 15.0000)
80 PROGRAM_END()
```

Optimized Canonical Machining Commands File

```
1 USE_LENGTH_UNITS(CANON_UNITS_MM)
2 SET_ORIGIN_OFFSETS(0.0000, 0.0000, 0.0000)
3 SET_FEED_REFERENCE(CANON_XYZ)
4 SPINDLE_RETRACT()
5 STOP_SPINDLE_TURNING()
6 USE_TOOL_LENGTH_OFFSET(0.0000)
7 CHANGE_TOOL(MILL 18MM)
8 USE_TOOL_LENGTH_OFFSET(50.0000)
9 FLOOD_ON()
10 SET_SPINDLE_SPEED(720.0000)
11 START_SPINDLE_CLOCKWISE()
12 SET_FEED_RATE(2400.0000)
13 COMMENT("WorkingStepID=FINISH␣PLANAR␣FACE1,
␣␣␣OperationName=mill_planar")
14 STRAIGHT_TRAVERSE(-13.5000, 8.1000, 250.0000)
15 STRAIGHT_TRAVERSE(-13.5000, 8.1000, 15.0000)
16 STRAIGHT_FEED(-13.5000, 8.1000, 3.7500)
17 STRAIGHT_FEED(113.5000, 8.1000, 3.7500)
18 STRAIGHT_TRAVERSE(113.5000, 8.1000, 15.0000)
19 STRAIGHT_TRAVERSE(-13.5000, 25.2000, 15.0000)
20 STRAIGHT_FEED(-13.5000, 25.2000, 3.7500)
```

```
STRAIGHT_FEED(113.5000, 25.2000, 3.7500)
STRAIGHT_TRAVERSE(113.5000, 25.2000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 42.3000, 15.0000)
STRAIGHT_FEED(-13.5000, 42.3000, 3.7500)
STRAIGHT_FEED(113.5000, 42.3000, 3.7500)
STRAIGHT_TRAVERSE(113.5000, 42.3000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 59.4000, 15.0000)
STRAIGHT_FEED(-13.5000, 59.4000, 3.7500)
STRAIGHT_FEED(113.5000, 59.4000, 3.7500)
STRAIGHT_TRAVERSE(113.5000, 59.4000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 76.5000, 15.0000)
STRAIGHT_FEED(-13.5000, 76.5000, 3.7500)
STRAIGHT_FEED(113.5000, 76.5000, 3.7500)
STRAIGHT_TRAVERSE(113.5000, 76.5000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 93.6000, 15.0000)
STRAIGHT_FEED(-13.5000, 93.6000, 3.7500)
STRAIGHT_FEED(113.5000, 93.6000, 3.7500)
STRAIGHT_TRAVERSE(113.5000, 93.6000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 110.7000, 15.0000)
STRAIGHT_FEED(-13.5000, 110.7000, 3.7500)
STRAIGHT_FEED(113.5000, 110.7000, 3.7500)
STRAIGHT_TRAVERSE(113.5000, 110.7000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 127.8000, 15.0000)
STRAIGHT_FEED(-13.5000, 127.8000, 3.7500)
STRAIGHT_FEED(113.5000, 127.8000, 3.7500)
STRAIGHT_TRAVERSE(113.5000, 127.8000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 8.1000, 15.0000)
STRAIGHT_FEED(-13.5000, 8.1000, 2.5000)
STRAIGHT_FEED(113.5000, 8.1000, 2.5000)
STRAIGHT_TRAVERSE(113.5000, 8.1000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 25.2000, 15.0000)
STRAIGHT_FEED(-13.5000, 25.2000, 2.5000)
STRAIGHT_FEED(113.5000, 25.2000, 2.5000)
STRAIGHT_TRAVERSE(113.5000, 25.2000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 42.3000, 15.0000)
```

```
STRAIGHT_FEED(-13.5000, 42.3000, 2.5000)
STRAIGHT_FEED(113.5000, 42.3000, 2.5000)
STRAIGHT_TRAVERSE(113.5000, 42.3000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 59.4000, 15.0000)
STRAIGHT_FEED(-13.5000, 59.4000, 2.5000)
STRAIGHT_FEED(113.5000, 59.4000, 2.5000)
STRAIGHT_TRAVERSE(113.5000, 59.4000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 76.5000, 15.0000)
STRAIGHT_FEED(-13.5000, 76.5000, 2.5000)
STRAIGHT_FEED(113.5000, 76.5000, 2.5000)
STRAIGHT_TRAVERSE(113.5000, 76.5000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 93.6000, 15.0000)
STRAIGHT_FEED(-13.5000, 93.6000, 2.5000)
STRAIGHT_FEED(113.5000, 93.6000, 2.5000)
STRAIGHT_TRAVERSE(113.5000, 93.6000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 110.7000, 15.0000)
STRAIGHT_FEED(-13.5000, 110.7000, 2.5000)
STRAIGHT_FEED(113.5000, 110.7000, 2.5000)
STRAIGHT_TRAVERSE(113.5000, 110.7000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 127.8000, 15.0000)
STRAIGHT_FEED(-13.5000, 127.8000, 2.5000)
STRAIGHT_FEED(113.5000, 127.8000, 2.5000)
STRAIGHT_TRAVERSE(113.5000, 127.8000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 8.1000, 15.0000)
STRAIGHT_FEED(-13.5000, 8.1000, 1.2500)
STRAIGHT_FEED(113.5000, 8.1000, 1.2500)
STRAIGHT_TRAVERSE(113.5000, 8.1000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 25.2000, 15.0000)
STRAIGHT_FEED(-13.5000, 25.2000, 1.2500)
STRAIGHT_FEED(113.5000, 25.2000, 1.2500)
STRAIGHT_TRAVERSE(113.5000, 25.2000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 42.3000, 15.0000)
STRAIGHT_FEED(-13.5000, 42.3000, 1.2500)
STRAIGHT_FEED(113.5000, 42.3000, 1.2500)
STRAIGHT_TRAVERSE(113.5000, 42.3000, 15.0000)
```

```
STRAIGHT_TRAVERSE(-13.5000, 59.4000, 15.0000)
STRAIGHT_FEED(-13.5000, 59.4000, 1.2500)
STRAIGHT_FEED(113.5000, 59.4000, 1.2500)
STRAIGHT_TRAVERSE(113.5000, 59.4000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 76.5000, 15.0000)
STRAIGHT_FEED(-13.5000, 76.5000, 1.2500)
STRAIGHT_FEED(113.5000, 76.5000, 1.2500)
STRAIGHT_TRAVERSE(113.5000, 76.5000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 93.6000, 15.0000)
STRAIGHT_FEED(-13.5000, 93.6000, 1.2500)
STRAIGHT_FEED(113.5000, 93.6000, 1.2500)
STRAIGHT_TRAVERSE(113.5000, 93.6000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 110.7000, 15.0000)
STRAIGHT_FEED(-13.5000, 110.7000, 1.2500)
STRAIGHT_FEED(113.5000, 110.7000, 1.2500)
STRAIGHT_TRAVERSE(113.5000, 110.7000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 127.8000, 15.0000)
STRAIGHT_FEED(-13.5000, 127.8000, 1.2500)
STRAIGHT_FEED(113.5000, 127.8000, 1.2500)
STRAIGHT_TRAVERSE(113.5000, 127.8000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 8.1000, 15.0000)
STRAIGHT_FEED(-13.5000, 8.1000, 0.0000)
STRAIGHT_FEED(113.5000, 8.1000, 0.0000)
STRAIGHT_TRAVERSE(113.5000, 8.1000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 25.2000, 15.0000)
STRAIGHT_FEED(-13.5000, 25.2000, 0.0000)
STRAIGHT_FEED(113.5000, 25.2000, 0.0000)
STRAIGHT_TRAVERSE(113.5000, 25.2000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 42.3000, 15.0000)
STRAIGHT_FEED(-13.5000, 42.3000, 0.0000)
STRAIGHT_FEED(113.5000, 42.3000, 0.0000)
STRAIGHT_TRAVERSE(113.5000, 42.3000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 59.4000, 15.0000)
STRAIGHT_FEED(-13.5000, 59.4000, 0.0000)
STRAIGHT_FEED(113.5000, 59.4000, 0.0000)
```

```
STRAIGHT_TRAVERSE(113.5000, 59.4000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 76.5000, 15.0000)
STRAIGHT_FEED(-13.5000, 76.5000, 0.0000)
STRAIGHT_FEED(113.5000, 76.5000, 0.0000)
STRAIGHT_TRAVERSE(113.5000, 76.5000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 93.6000, 15.0000)
STRAIGHT_FEED(-13.5000, 93.6000, 0.0000)
STRAIGHT_FEED(113.5000, 93.6000, 0.0000)
STRAIGHT_TRAVERSE(113.5000, 93.6000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 110.7000, 15.0000)
STRAIGHT_FEED(-13.5000, 110.7000, 0.0000)
STRAIGHT_FEED(113.5000, 110.7000, 0.0000)
STRAIGHT_TRAVERSE(113.5000, 110.7000, 15.0000)
STRAIGHT_TRAVERSE(-13.5000, 127.8000, 15.0000)
STRAIGHT_FEED(-13.5000, 127.8000, 0.0000)
STRAIGHT_FEED(113.5000, 127.8000, 0.0000)
STRAIGHT_TRAVERSE(113.5000, 127.8000, 15.0000)
PROGRAM_END()
```